ภาคผนวก ก

ภาคผนวก ก

ในการสร้างตัวแปรให้มีคุณสมบัติตามต้องการ วิธีการหนึ่งที่สามารถทำได้คืออาศัยเทคนิคของการผลิตเลขสุ่มโดยการใช้โปรแกรม

1. การผลิตเลขสุ่มโดยการใช้โปรแกรม

เป็นการผลิตเลขสุ่มจากความสัมพันธ์ที่ซ้ำๆ (recurrence relation) กล่าวคือเลขถัดไปเกิดจากการดำเนินการทางคณิตศาสตร์ และตรรกศาสตร์ของตัวเลขปัจจุบันหรือในอดีต อนุบรรพ (sequence) ของตัวเลขซึ่งผลิตได้จึงเป็นอนุบรรพของเลขสุ่มในความหมายที่แท้จริงไม่ได้ แต่อย่างไรก็ตามเลขที่ผลิตในอนุบรรพเหล่านี้อาจจะผ่านการทดสอบความเป็นสุ่มทางสถิติได้หลายอย่าง และเรียกว่าเลขคล้ายสุ่ม (Pseudo-Random Number)

ชุดตัวเลขสุ่มที่ผลิตขึ้นต้องมีคุณสมบัติทางสถิติที่สำคัญ 2 ประการคือ ความสม่ำเสมอ (uniform) และความเป็นอิสระ (independence) ซึ่งวิธีการผลิตชุดตัวเลขสุ่มดังกล่าว อาศัยการใช้ฟังก์ชันมาตรฐานที่อยู่ในโปรแกรมเทอร์โบปาสคาล สำหรับคำสั่งในการเรียกใช้ฟังก์ชันดังกล่าว มีดังนี้

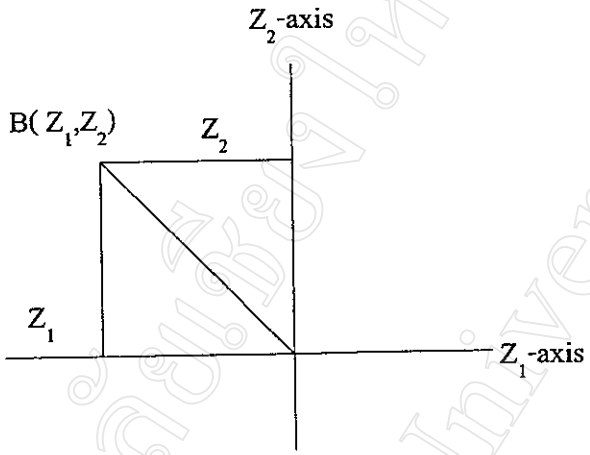
```
randomize;
 for i:=1 to m do
 begin
      r[i]:=random;
 end;
```

2. การผลิตเลขสุ่มที่มีการแจกแจงแบบปกติ

การแจกแจงแบบปกติโดยใช้เทคนิคการแปลงโดยตรงจาก

$$F(X) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \quad ; \quad -\infty < X < \infty$$

Box และ Muller (ค.ศ. 1958) สร้างเลขสุ่มที่มีการแจกแจงแบบปกติมาตรฐานที่มีค่าเฉลี่ยเป็น 0 และค่าความแปรปรวนเป็น 1 พร้อมๆ กัน 2 ค่า ดังนี้



$$Z_1 = B\cos\theta$$

$$Z_2 = B\sin\theta$$

$B^2 = Z_1^2 + Z_2^2$ มีการแจกแจงไคสแควร์ (chi-square distribution) ด้วยระดับความเป็นอิสระเท่ากับ 2 ซึ่งเทียบเท่า (equivalent) กับการแจกแจงเอ็กซ์โปเนนเชียล (exponential distribution) ด้วยค่าเฉลี่ยเท่ากับ 2 ดังนั้นรัศมี B มีค่าดังนี้

$$B = (-2\ln R)^{\frac{1}{2}}$$

โดยการสมมาตร (symmetry) ของการแจกแจงแบบปกติ (normal distribution)

θ มีการแจกแจงสม่ำเสมอ (uniform distribution) ระหว่าง 0 กับ 2π เรเดียน ซึ่งค่า B และ θ เป็น mutually independent

$$Z_1 = (-2\ln R_1)^{\frac{1}{2}}\cos(2\pi R_2)$$

$$Z_2 = (-2\ln R_1)^{\frac{1}{2}}\sin(2\pi R_2)$$

การสร้างตัวแปรที่มีการแจกแจงเป็นแบบปกติ มีค่าเฉลี่ย μ และค่าเบี่ยงเบนมาตรฐาน σ โดยอาศัยชุดตัวเลขสุ่ม คำสั่งที่ใช้ดังนี้

```
i:=1; repeat
      z[i]:=sqrt((-2)*ln(r[i]))*(cos(2*pi*r[i+1]));
      z[i+1]:=sqrt((-2)*ln(r[i]))*(sin(2*pi*r[i+1]));
i:=i+2; until i > (m*4)+2;
```

3. การสร้างการแจกแจงแบบลอกนอร์มอล

จากการแจกแจงแบบลอกนอร์มอลมีฟังก์ชันความหนาแน่นอยู่ในรูปของ

$$f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(\ln X-\mu)^2} \quad ; x \geq 0$$

เมื่อ μ และ σ^2 เป็นค่าเฉลี่ยและความแปรปรวนของ Y ซึ่ง $Y = \log_e X$ และ Y มีการแจกแจงแบบปกติ โดยมี $\exp(\sigma^2)$ เป็น scale parameter และ μ เป็น shape parameter

ค่าคาดหวัง ค่าความแปรปรวนและค่าสัมประสิทธิ์ความแปรปรวนของการแจกแจงแบบลอกนอร์มอล คือ

$$E(X) = \exp\{\mu + \frac{\sigma^2}{2}\}$$

$$V(X) = \exp\{2\mu + \sigma^2\}.\{\exp\{\sigma^2\} - 1\}$$

$$CV(X) = \sqrt{\exp\{\sigma^2\} - 1}$$

ดังนั้นในการสร้างตัวแปรให้มีการแจกแจงแบบลอกนอร์มอล โดยอาศัยตัวแปรสุ่มที่มีการแจกแจงแบบปกติ คำสั่งที่ใช้มีดังนี้

```
for i:=1 to m do
    begin
        if (a=1) and (v=1) then  e[i]:=exp((z[i]*0.5));
        if (a=1) and (v=2) then  e[i]:=exp((z[i]*1.0));
    end;
```

เมื่อ z[i] เป็นตัวแปรสุ่มที่มีการแจกแจงแบบปกติมาตรฐาน

4. การสร้างการแจกแจงแบบไคสแควร์

การแจกแจงแบบไคสแควร์ มีฟังก์ชันความหนาแน่นอยู่ในรูปของ

$$f(x) = \frac{1}{2^{\frac{\nu}{2}}\Gamma(\nu/2)} x^{\frac{\nu}{2}-1} e^{\frac{-x}{2}} \qquad ; x \geq 0$$

การสร้างตัวแปรสุ่ม X ที่มีการแจกแจงแบบ chi-square ที่มีระดับองศาแห่งความอิสระเท่ากับ ν โดยอาศัยความสัมพันธ์ระหว่างการแจกแจงดังกล่าวกับการแจกแจงแบบปกติมาตรฐาน (Z) ซึ่งมีรูปแบบความสัมพันธ์ดังนี้

$$X : \nu \sim \sum_{i=1}^{\nu} Z_i^2$$

ค่าคาดหวัง ค่าความแปรปรวนและค่าสัมประสิทธิ์ความแปรปรวนของการแจกแจงแบบไคสแควร์ คือ

$$E(X) = \nu$$

$$V(X) = 2\nu$$

$$CV(X) = \sqrt{\frac{2}{\nu}}$$

ดังนั้น คำสั่งในการสร้างตัวแปรให้มีการแจกแจงแบบไคสแควร์ คือ

```
for i:=1 to m do
    begin
        if (a=2) and (v=1) then  e[i]:=(z[i]*z[i]);
        if (a=2) and (v=2) then  e[i]:=(z[i]*z[i])+(z[i+m]*z[i+m])+(z[i+(2*m)]);
        if (a=2) and (v=3)
            then e[i]:=(z[i]*z[i])+(z[i+m]*z[i+m])+(z[i+(2*m)]*(z[i+(2*m)]));
    end;
```

ภาคผนวก ข

ภาคผนวก ข

โปรแกรมที่ใช้ในการวิจัย

```
program  powertest(alpha,beta);
uses     printer,crt;
type     matrix=array[0..105,0..15] of real;
         mat=array[0..15,0..15] of real;
         ma=array[1..2,2..6,4..100] of real;
         vector=array[0..108] of real;  vect=array[0..452] of real;
         vec=array[0..11] of real;
const    pi=3.141592654;
var      a,t,v,i,j,k,m,n,q,h,check,che,ch,loop,count1,coun1,count2,coun2:integer;
         count3,count4,count5,count6,coun3,coun4,coun5,coun6,v1,v2,loo:integer;
         sum1,sum2,sum3,sum4,mea1,mea2,mea3,mea4,ssf,ssr,sst,dif,sum,mea:real;
         f1,f2,f3,alpha1,beta1,alpha2,beta2,alpha3,alpha4,beta3,beta4,ss:real;
         alpha5,alpha6,beta5,beta6:real;
         xf,xr,xw:matrix;     ff:ma;   xxr,xxf:mat;      r,z:vect;
         e,x1,x2,x3,x4,w,y,yy,ee,diff,ox1,ox2,ox3,ox4:vector;
         orx1,orx2,orx3,orx4,yf:vector;     re,betaf,betar,xyf,xyr:vec;

procedure initializ(var xyf:vec; ch:integer);
var      j:integer;
begin
        for j:=1 to ch do
        begin xyf[j]:=0; end;
end; {of procedure initializ}
```

```
procedure crossmatrix(ch,m:integer;  xx,ff:matrix;  var xxf:mat);
var     s,j,i:integer;
        begin
            for s:=1 to ch do
      begin
        for j:=1 to ch do
        begin
           xxf[s,j]:=0;
           for i:=1 to m do
           begin xxf[s,j]:=xxf[s,j]+xx[i,j]*ff[i,s];end;
        end;
      end;
end; {of crossmatrix}

procedure crossmat(ch,m:integer;  xf:matrix;  y:vector; var xyf:vec);
var     j,i:integer;
begin
     for j:=1 to ch do
     begin
        xyf[j]:=0;
        for i:=1 to m do
        begin xyf[j]:=xyf[j]+xf[i,j]*y[i]; end;
      end;
end; {of crossmat}

procedure iterate(ch:integer; xxf:mat; xyf:vec; var betaf:vec);
var     i,j:integer;      approxf,sumf:real;
begin
        for i:=1 to ch do
        begin
```

```
        sumf:=0;
            for j:=1 to ch do
                    begin
                    if i<>j then
            sumf:=sumf+xxf[i,j]*betaf[j]; end;
            betaf[i]:=(xyf[i]-sumf)/xxf[i,i];
        end;
end; {of procedure iterate}

procedure Gauss_Seidel_Driver(xxf,xxr:mat; xyf,xyr:vec; var betaf,betar:vec);
var     t:integer;
begin
        t:=1; repeat
        iterate(ch,xxf,xyf,betaf);
        iterate(h,xxr,xyr,betar);
        t:=t+1; until t=200;
end; {of procedure Gauss_Seidiel_Driver}

procedure cal_ee(var ee:vector; m,ch:integer; xf:matrix; betaf:vec; y:vector);
var     i,j:integer;  sum:real;
begin
        for i:=1 to m do
        begin
            sum:=0;
            for j:=1 to ch do
            begin
            sum:=sum+xf[i,j]*betaf[j]; end;
            ee[i]:=y[i]-sum;
        end;
end; {of procedure cal_ee}
```

```
procedure sor(var item:vector ; count:integer);
        var     l,j,k:integer; x:real;
        begin
       for l:=1 to count-1 do
       begin
        k:=l;
        x:=item[l];
        for j:=l+1 to count do
        if item[j]<x then
          begin
          k:=j;
          x:=item[j];
          end;
          item[k]:=item[l];
           item[l]:=x;
      end;
end; {of procedure sor}

procedure  cal_ss(var diff:vector; var ee:vector; xf:matrix; betaf:vec; y:vector; m,ch:integer);
var      i,c:integer; ss:real;
begin
        for i:=1 to m do
        begin
           sor(ee,m);
           if odd(m) then
           begin
           c:=(m+1) div 2; ss:=ee[c]; end
              else
              begin
```

```
            c:=m div 2;
            ss:=(ee[c]+ee[c+1])/2;
                    end;
                    cal_ee(ee,m,ch,xf,betaf,y);
            diff[i]:=abs(ee[i]-ss);
        end;
end; {of procedure cal_ss}

procedure   cal_yf(var yf:vector; diff,ee:vector; m:integer);
var         i,c:integer;
begin
        sor(diff,m);
        for i:=1 to m do
        begin
          if odd(m) then
          begin
          c:=(m+1) div 2;
          ss:=diff[c]/0.6745; end
          else
            begin
            c:=m div 2;
            ss:=(diff[c]+diff[c+1])/2;
            ss:=ss/0.6745;
            end;
            yf[i]:=(-0.3)*(abs(ee[i]*ss));
            yf[i]:=(ee[i]/ss)*(exp(yf[i]));
      end;
end;{of procedure cal_yf}

procedure calw(m:integer; ss:real; ee,yf:vector; var wf:vector);
```

```
var     i,j:integer; appro,sum:vector;   d:real;
begin
                for i:=1 to m do
                begin
          d:=ee[i]/ss;
          if d=0 then wf[i]:=1
            else  wf[i]:=(yf[i]*(abs(d))/d);
       end;
end; {of procedure calw}

procedure rank(var ord:vector; x,y:vector;  m:integer);
var     i,j:integer;
begin    for i:=1 to m do
       begin
          for j:=1 to m do
          begin
          if x[i]=y[j] then ord[i]:=j;
          end;
       end;
end;{of procedure rank}

procedure calstat_f(ch,h,m:integer; e,ee,y:vector; betaf,betar:vec;
       xf,xr:matrix; var f:real);
var     i:integer; sst,sstr:real;
begin
       sst:=0; sstr:=0;
       cal_ee(ee,m,ch,xf,betaf,y);        cal_ee(e,m,h,xr,betar,y);
       for i:=1 to m do
       begin sst:=sst+ee[i]*ee[i]; end;
       for i:=1 to m do
```

```
        begin sstr:=sstr+e[i]*e[i]; end;
            f:=(sstr-sst)/(k-1);
                    f:=f/(sst/(m-ch));
          end;  {of procedure calsts_f}

begin {Main program}
clrscr;
a:=1; repeat  t:=1; repeat v:=1; repeat q:=2; repeat k:=3; repeat   n:=5; repeat
       count1:=0; coun1:=0; count2:=0; coun2:=0;    count3:=0;  coun3:=0;
       count4:=0; coun4:=0; count5:=0; coun5:=0;    count6:=0;  coun6:=0;

       loop:=1; repeat
       sum1:=0; sum2:=0; sum3:=0; sum4:=0;  m:=n*k;  ch:=k+q; h:=q+1;
       randomize;
       for i:=0 to (m*4)+2 do
       begin r[i]:=random; end;

    {Generate random number which have lognormal and chi-square distribution}
       i:=1; repeat
           z[i]:=sqrt((-2)*ln(r[i]))*(cos(2*pi*r[i+1]));
           z[i+1]:=sqrt((-2)*ln(r[i]))*(sin(2*pi*r[i+1]));
       i:=i+2; until i > (m*4)+2;

        for i:=1 to m do
        begin
           if (a=1) and (v=1) then  e[i]:=exp((z[i]*0.5));
           if (a=1) and (v=2) then  e[i]:=exp((z[i]*1.0));
           if (a=2) and (v=1) then  e[i]:=(z[i]*z[i]);
           if (a=2) and (v=2) then  e[i]:=(z[i]*z[i])+(z[i+m]*z[i+m]);
           if (a=2) and (v=3) then  e[i]:=(z[i]*z[i])+(z[i+m]*z[i+m])+(z[i+(2*m)]*(z[i+(2*m)]));
```

```
                {Generate x which have normal distribution}
        x1[i]:=5+z[i]*0.5;                    x2[i]:=5+z[i+m]*1.0;
    x3[i]:=5+z[i+(2*m)]*1.5;                  x4[i]:=5+z[i+(3*m)]*2.0;

      sum1:=sum1+x1[i];                       sum2:=sum2+x2[i];
      sum3:=sum3+x3[i];                       sum4:=sum4+x4[i];
   end;
      mea1:=sum1/m;                           mea2:=sum2/m;
      mea3:=sum3/m;                           mea4:=sum4/m;

                {Generate matrix of full model x}
for i:=1 to m do
begin
   check:=1;  repeat
      if (check=1) then xf[i,1]:=1;
      if (check=2) then  if (i<=n) then xf[i,2]:=1 else xf[i,2]:=0;
      if (check=3) then  if (n<i) and (i<=(2*n))          then xf[i,3]:=1 else  xf[i,3]:=0;
      if (check=4) then  if (i>(2*n)) and(i<=(3*n))       then xf[i,4]:=1 else  xf[i,4]:=0;
      if (check=5) then  if (i>(3*n)) and(i<=(4*n))       then xf[i,5]:=1 else  xf[i,5]:=0;
      if (check=6) then  if (i>(4*n)) and(i<=(5*n))       then xf[i,6]:=1 else  xf[i,6]:=0;
      if (check=7) then  if (i>(5*n)) and(i<=(6*n))       then xf[i,7]:=1 else  xf[i,7]:=0;
   check:=check+1;  until check=k+1;
   check:=k+1;
   repeat
      if (check=4)  then xf[i,4] := x1[i] - mea1;     if (check=5)  then xf[i,5] := x2[i] - mea2;
      if (check=6)  then xf[i,6] := x3[i] - mea3;     if (check=7)  then xf[i,7] := x4[i] - mea4;
      if (check=8)  then xf[i,8] := x1[i] - mea1;     if (check=9)  then xf[i,9] := x2[i] - mea2;
      if (check=10) then xf[i,10]:= x3[i] - mea3;     if (check=11) then xf[i,11]:= x4[i] - mea4;
   check:=check+1;  until check =ch+1;
```

```
                                {Generate vector of treatment}
        che:=1;
              repeat
         if (t=1) and (che=1) then tre[1]:=15;
    if (t=1) and (che>1) then tre[che]:= 0;        if (t=2) and (che=1) then tre[1]:= 15;
if (t=2) and (che=2) then tre[2]:= -0.7;           if (t=2) and (che=3) then tre[3]:= 0.7;
if (t=2) and (che=4) then tre[4]:= -0.6;           if (t=2) and (che=5) then tre[5]:= 0.6;
if (t=2) and (che=6) then tre[6]:= -0.5;           if (t=2) and (che=7) then tre[7]:= 0.5;
        che:=che+1; until che = k+1;
        che:=k+1;
        repeat  tre[che] :=1.2;                    che:=che+1; until che =ch+1;

                                {Generate vector of y}
      z[0]:=0;
      for j:=1 to ch do
      begin z[j]:=z[j-1]+xf[i,j]*tre[j]; end;
      y[i]:=z[j]+e[i];          yy[i]:= y[i];

                                {Generate reduce model of x}
      check:=1;  repeat
    if (k=3) or (k=7) then begin
          if (check=1) then xr[i,1]:=1;
          if (check=2) then xr[i,2]:=x1[i]-mea1;       if (check=3) then xr[i,3]:=x2[i]-mea2;
          if (check=4) then xr[i,4]:=x3[i]-mea3;       if (check=5) then xr[i,5]:=x4[i]-mea4;
      end;

    if k=5 then begin
      if (check=1) then xr[i,1]:=1;
      if (check=2) then xr[i,2]:=x3[i]-mea3;       if (check=3) then xr[i,3]:=x4[i]-mea4;
      if (check=4) then xr[i,4]:=x1[i]-mea1;       if (check=5) then xr[i,5]:=x2[i]-mea2;
```

```
end;
        check:=check+1;        until check =q+2;
         end;

                                {OLS method}

crossmatrix(ch,m,xf,xf,xxf);                    crossmatrix(h,m,xr,xr,xxr);
crossmat(ch,m,xf,y,xyf);                        crossmat(h,m,xr,y,xyr);
initializ(betaf,ch);                            initializ(betar,h);
Gauss_Seidel_Driver(xxf,xxr,xyf,xyr,betaf,betar);
calstat_f(ch,h,m,e,ee,y,betaf,betar,xf,xr,f1);

                                {M-estimation method}

loo:=1; repeat
cal_ee(ee,m,ch,xf,betaf,y);                     cal_ss(diff,ee,xf,betaf,y,m,ch);
cal_yf(yf,diff,ee,m);                           calw(m,ss,ee,yf,w);
for i:=1 to m do
begin
    for j:=1 to ch do
    begin xw[i,j]:=xf[i,j]*w[i];  end;
end;
crossmatrix(ch,m,xw,xf,xxf);                    crossmat(ch,m,xw,y,xyf);
cal_ee(e,m,h,xr,betar,y);                       cal_ss(diff,e,xr,betar,y,m,h);
cal_yf(yf,diff,e,m);                            calw(m,ss,e,yf,w);
for i:=1 to m do
begin
     for j:=1 to h do
     begin xw[i,j]:=xr[i,j]*w[i];  end;
end;
crossmatrix(h,m,xw,xr,xxr);                     crossmat(h,m,xw,y,xyr);
initializ(betaf,ch);                            initializ(betar,h);
```

```
   Gauss_Seidel_Driver(xxf,xxr,xyf,xyr,betaf,betar);
loo:=loo+1; until loo > 10;
   calstat_f(ch,h,m,e,ee,y,betaf,betar,xf,xr,f2);

                          {Rank transformation method}
   sor(yy,m);                                  rank(w,y,yy,m);
   for i:=1 to m do
   begin
      x1[i]:=x1[i]-mea1;                       ox1[i]:=x1[i];
      x2[i]:=x2[i]-mea2;                       ox2[i]:=x2[i];
      x3[i]:=x3[i]-mea3;                       ox3[i]:=x3[i];
      x4[i]:=x4[i]-mea4;                       ox4[i]:=x4[i];
   end;
      sor(ox1,m);                              rank(orx1,x1,ox1,m);
      sor(ox2,m);                              rank(orx2,x2,ox2,m);
      sor(ox3,m);                              rank(orx3,x3,ox3,m);
      sor(ox4,m);                              rank(orx4,x4,ox4,m);
   for i:=1 to m do
   begin
   check:=k+1;
   repeat
      if (check=4)  then xf[i,4] :=  orx1[i];     if (check=5)  then xf[i,5] :=  orx2[i];
      if (check=6)  then xf[i,6] :=  orx3[i];     if (check=7)  then xf[i,7] :=  orx4[i];
      if (check=8)  then xf[i,8] :=  orx1[i];     if (check=9)  then xf[i,9] :=  orx2[i];
      if (check=10) then xf[i,10]:= orx3[i];      if (check=11) then xf[i,11]:= orx4[i];
   check:=check+1;
   until check =ch+1;

   check:=1;   repeat
   if (k=3) or (k=7) then begin
```

```
    if (check=1) then xr[i,1]:=1;
    if (check=2) then xr[i,2]:=orx1[i];          if (check=3) then xr[i,3]:=orx2[i];
          if (check=4) then xr[i,4]:=orx3[i];          if (check=5) then xr[i,5]:=orx4[i];
       end;
  if k=5 then begin
    if (check=1) then xr[i,1]:=1;
    if (check=2) then xr[i,2]:=orx3[i];          if (check=3) then xr[i,3]:=orx4[i];
    if (check=4) then xr[i,4]:=orx1[i];          if (check=5) then xr[i,5]:=orx2[i];
  end;
  check:=check+1;
    until check =q+2;
  end;
  crossmatrix(ch,m,xf,xf,xxf);                   crossmatrix(h,m,xr,xr,xxr);
  crossmat(ch,m,xf,w,xyf);                       crossmat(h,m,xr,w,xyr);
  initializ(betaf,ch);                           initializ(betar,h);
  Gauss_Seidel_Driver(xxf,xxr,xyf,xyr,betaf,betar);
  calstat_f(ch,h,m,e,ee,w,betaf,betar,xf,xr,f3);

    v1:=k-1; v2:=m-k-q;
     if t=1 then  begin

if f1 > ff[1,v1,v2] then count1:=count1+1;     if f1 > ff[2,v1,v2] then count2:=count2+1;
if f2 > ff[1,v1,v2] then count3:=count3+1;     if f2 > ff[2,v1,v2] then count4:=count4+1;
if f3 > ff[1,v1,v2] then count5:=count5+1;     if f3 > ff[2,v1,v2] then count6:=count6+1;
        end;
if t=2 then  begin
if f1 <= ff[1,v1,v2] then coun1:=coun1+1;      if1 <= ff[2,v1,v2] then coun2:=coun2+1;
if f2 <= ff[1,v1,v2] then coun3:=coun3+1;      if2 <= ff[2,v1,v2] then coun4:=coun4+1;
if f3 <= ff[1,v1,v2] then coun5:=coun5+1;      if3 <= ff[2,v1,v2] then coun6:=coun6+1;
        end;
```

```
        loop:=loop+1; until loop = 1001;
         alpha1:=count1/(loop-1);                    beta1:=1-(coun1/(loop-1));
                alpha2:=count2/(loop-1);                    beta2:=1-(coun2/(loop-1));
         alpha3:=count3/(loop-1);                    beta3:=1-(coun3/(loop-1));
alpha4:=count4/(loop-1);                    beta4:=1-(coun4/(loop-1));
         alpha5:=count5/(loop-1);                    beta5:=1-(coun5/(loop-1));
         alpha6:=count6/(loop-1);                    beta6:=1-(coun6/(loop-1));

        n:=n+5;until n=20;     k:=k+2; until k=9;     q:=q+2; until q=6;    v:=v+1; until v=4;
        t:=t+1; until t=3;  a:=a+1; until a=3;
      end.
```

ประวัติผู้เขียน

ชื่อ – สกุล	นางสาวปาริฉัตร ทวีบุรุษ
วัน เดือน ปีเกิด	5 พฤษภาคม 2520
ประวัติการศึกษา	สำเร็จการศึกษามัธยมศึกษาตอนปลาย ศูนย์การศึกษานอกโรงเรียน จังหวัดสตูล ปีการศึกษา 2536 สำเร็จการศึกษาปริญญาวิทยาศาสตรบัณฑิต สาขาสถิติ มหาวิทยาลัยขอนแก่น 2541