回胴式遊技機の子役出現率による設定推測の考察

芝浦工業大学　数理科学研究会

～判別分析による誤判別確率の応用～

平成 23 年 11 月 3 日

※何か不明な点や計算ミス等がありましたら加筆修正しますので指摘をお願いします

制作:佐野遼太郎

1　研究背景

スロットマシン (パチスロ) には設定というものがあり, その多くが設定 1 から設定 6 までの 6 段階に分けられている.詳細はスロットマシンの機種により様々だが, 設定 1 だと負け (損する), 設定 6 だと勝つ (儲かる) という仕様になっている.つまり設定は高い方が良いのである.しかし, 設定はスロットマシンの外からは分からない.そこで, 多くの人が子役 (簡単に当たるもの) といわれる物の出現率から設定を推測して, 設定が低いようならやめて, 設定が高いようなら続けて試行するという行動を取っている.しかし, 何回試行した上での子役出現率かによって, 設定推測にどれだけ使えるデータなのかが違ってくる.そこで何回スロットマシンを試行した上で, 子役出現率がどのくらいならば設定 $x(x = 1, 2, \cdots, 6)$ である可能性はどのくらいか?を調べて, スロットマシンの設定看破に役立てて欲しく, この研究テーマを選んだ.

2　実験機種の仕様

今回は実験に使用するスロットマシンの機種を「うるせいやつら 2」(Sammy) とした.選んだ理由は, 機種のシステムを単純化でき, シミュレーションしやすく, 子役出現率の設定差が大きいためである.この機種の詳細は以下の通りとする.

設定	1	2	3	4	5	6
A	7.43	7.31	7.25	7.07	6.98	6.79
B	165	155	146	137	128	114
C	32.2	33.6	33.8	34.4	34.8	35.7

表 1 : 各設定の詳細

このとき, A : 子役出現必要回転数, B : 大当たり出現必要回転数, C : 1000 円あたりの試行回数としている.今回はシミュレーションに十進 BASIC, C + +を使用した.C + +に関しては, 乱数の生成の精度が悪いため rand() の値を一旦 $[0, 1)$ の実数に正規化してから M 倍して整数部をとるようにして精度を上げた.

3　二項分布の正規近似

今回扱う確率分布は二項分布となる.二項分布とは p を確率関数とすると, 次のように表せる (n は正の整数, $0 < \alpha < 1$).

$$p(k) = {}_nC_k\alpha^k(1-\alpha)^{n-k} \quad (k = 0, 1, \cdots, n)$$

二項分布は**中心極限定理**より正規分布に近似できる.これをプログラムを実装して確認してみる.

3.1　プログラムの実装

```
#include<stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#define PI 6*asin( 0.5 )
#define N 10000
#define M 1000
```

```
int a[121],b[121];
double u[2],s[2];

double f(double x,double u,double s)
{
 return exp(-(x-u)*(x-u)*0.5/s)/(double) sqrt(2*(PI)*s);
}

void settei1(int a[])
{
 int i,j,x,y;
 double z,w;
 z=0;w=0;
 for(j=1;j<=N;j++)
  {
   y=0;
   for(i=1;i<=M;i++)
    {
     x=(int) ((rand()/(RAND_MAX+1.0))*743);
     if(x<100){y=y+1;}
    }
   z=z+y;
   w=w+y*y;
   for(i=80;i<=200;i++)
    {
     if(y==i){a[i-79]=a[i-79]+1;}
    }
   u[0]=u[0]+y;
  }
 u[0]=u[0]/(double) N;
 s[0]=(w-(z*z/(double) N))/((double) N-1);
}



void settei6(int b[])
{
 int i,j,x,y;
 double z,w;
 z=0;w=0;
 for(j=1;j<=N;j++)
  {
   y=0;
   for(i=1;i<=M;i++)
    {
```

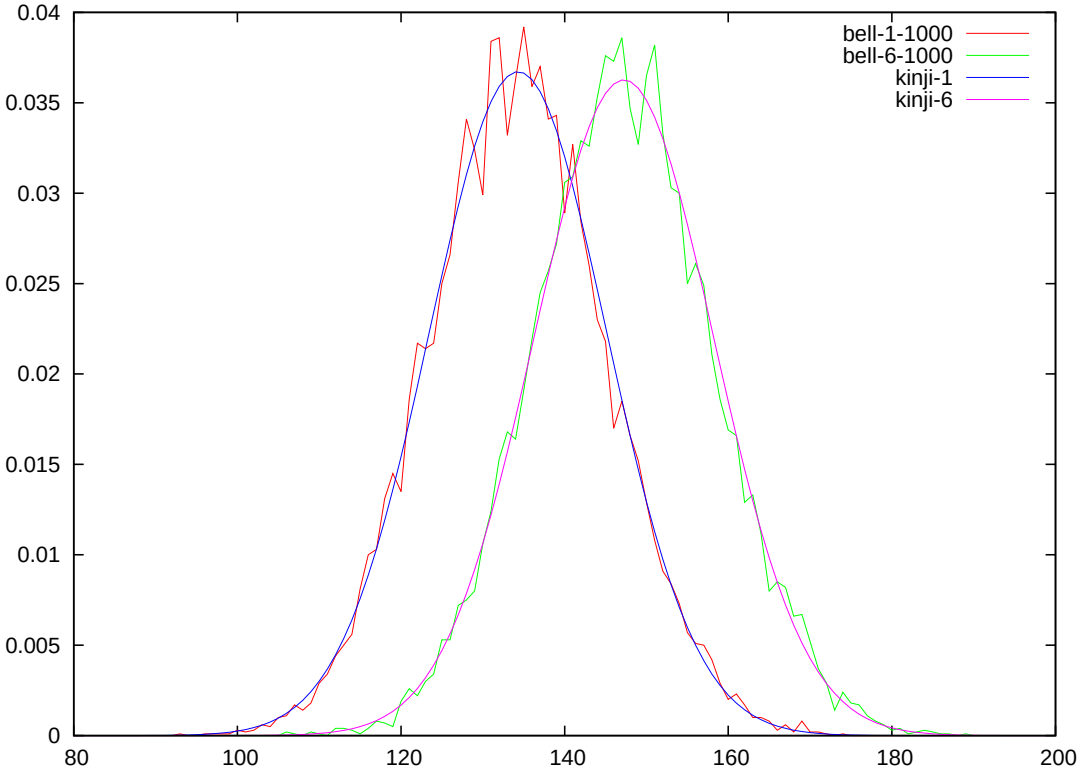
```
     x=(int) ((rand()/(RAND_MAX+1.0))*679);
     if(x<100){y=y+1;}
    }
   z=z+y;
   w=w+y*y;
   for(i=80;i<=200;i++)
    {
     if(y==i){b[i-79]=b[i-79]+1;}
    }
   u[1]=u[1]+y;
  }
 u[1]=u[1]/(double) N;
 s[1]=(w-(z*z/(double) N))/((double) N-1);
}

void main()
{
 double x;
 int i;
 srand((unsigned int)time(NULL));
 for(i=0;i<=120;i++){a[i]=0; b[i]=0;}
 settei1(a); settei6(b);
 for(i=1;i<=120;i++)
  {
   printf("%d %lf %lf %lf %lf\n",
          i+79,a[i]/(double) N,b[i]/(double) N,f(i+79,u[0],s[0]),f(i+79,u[1],s[1]));
  }
}
```

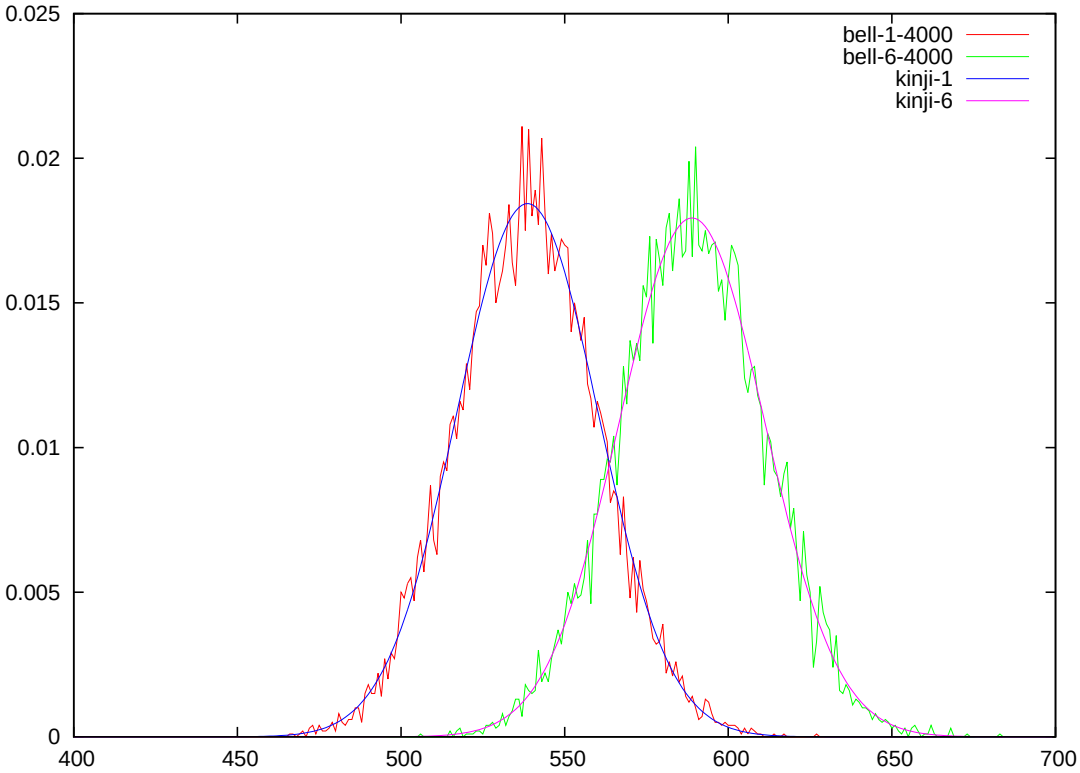
このプログラムは試行回数 1000 における, 子役出現回数の確率分布 (二項分布) と, この分布と同じ平均と分散の正規分布を出力するものである. また, このプログラムを少し書き換える事で試行回数 4000,10000,100000 の場合, ボーナス出現回数における同様の場合の出力が可能である.

3.2 プログラムの実行結果

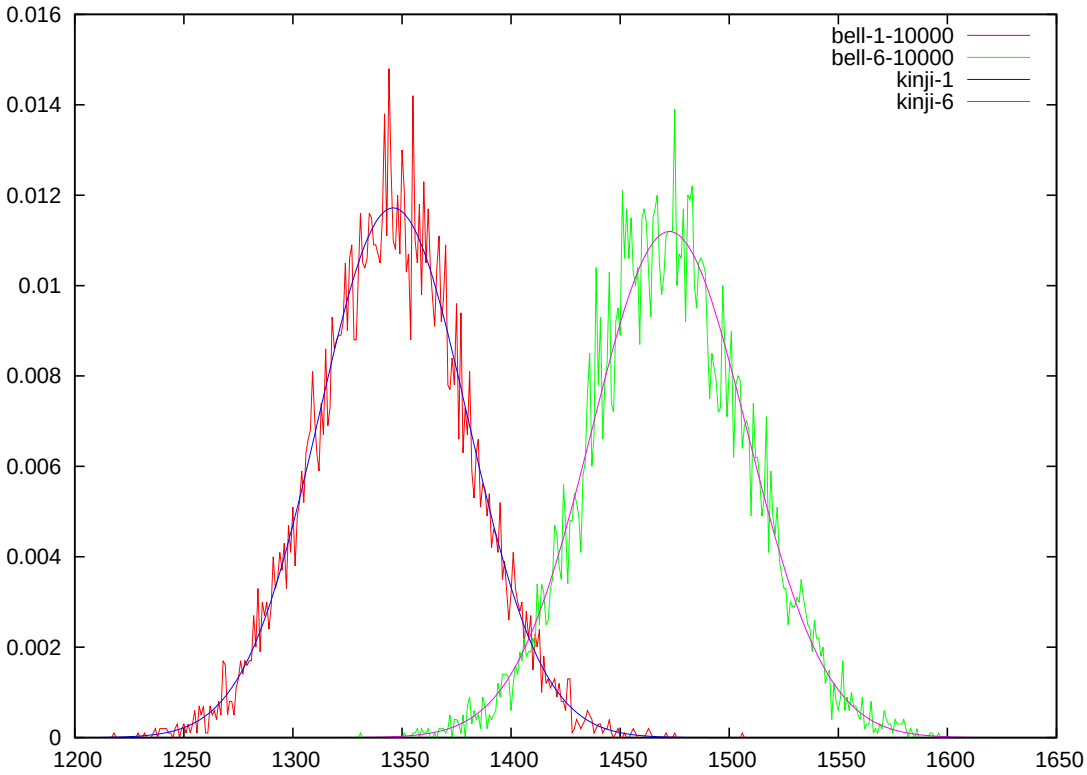
上のプログラムにおける試行回数を $1000, 4000, 10000, 100000$ としたグラフ, ボーナス出現回数における同様の試行回数のグラフを出力した.



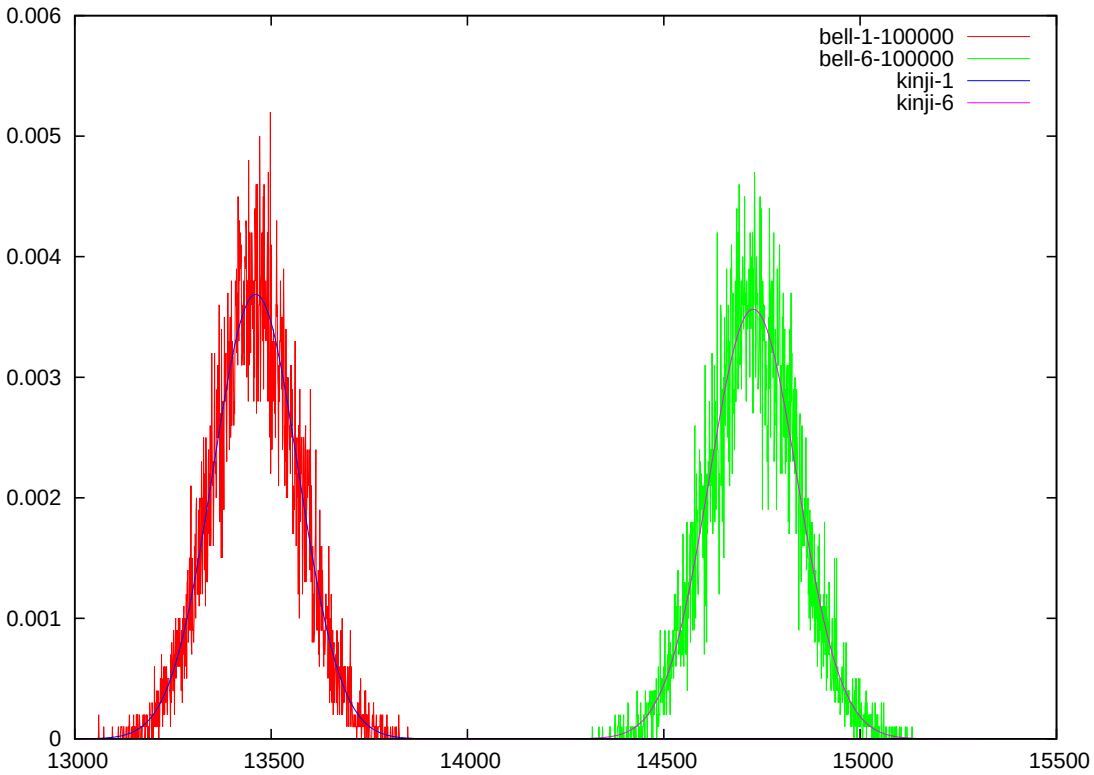
グラフ 1 : 試行回数 1000 の二項分布と, 近似した正規分布



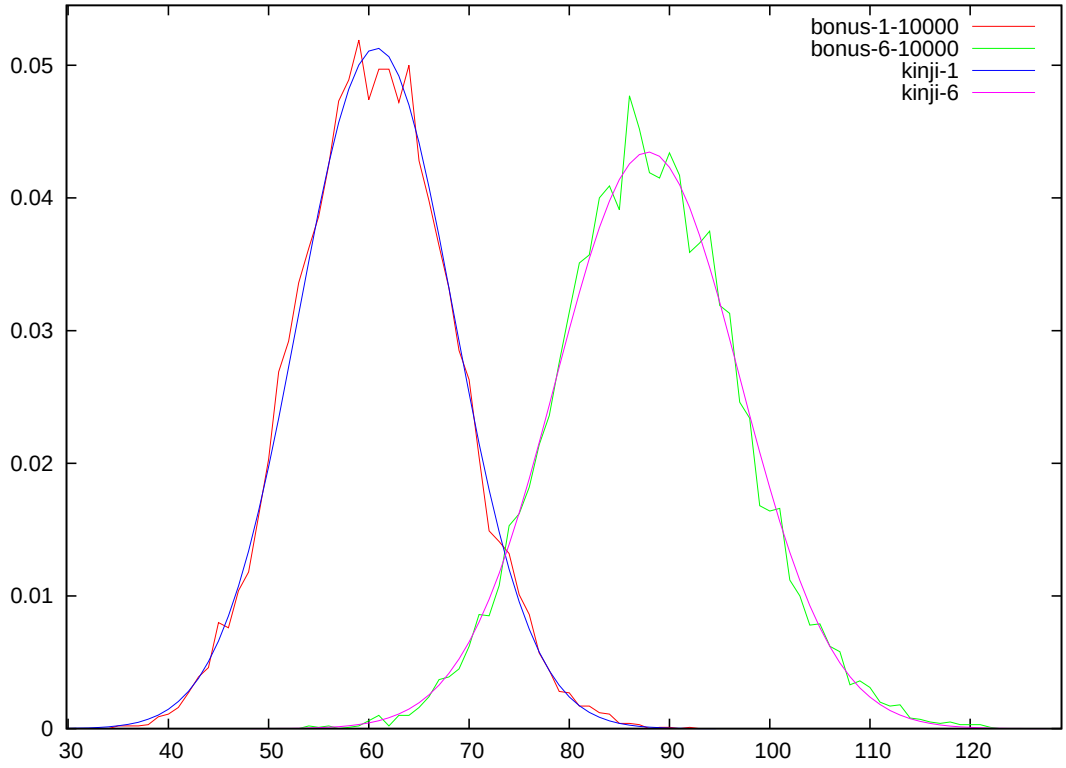
グラフ 2 : 試行回数 4000 の二項分布と, 近似した正規分布



グラフ 3 : 試行回数 10000 の二項分布と, 近似した正規分布



グラフ 4 : 試行回数 100000 の二項分布と, 近似した正規分布



グラフ 5 : 試行回数 10000 の二項分布と, 近似した正規分布 (※ボーナス)

この実行結果から確かに正規分布に近似されている事が確認できた.

4 判別分析の適用判断

4.1 判別分析

平方和を次の様に定義する.

$$S_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} {x_i}^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}$$

判別分析は, 2 つの母集団を設定して, あるサンプルがどちらの母集団に属するのかを推測するための方法である.次に, マハラノビスの距離の 2 乗を次のように定義する.

$$D^{[1]2} = \frac{(x_1 - {\mu_1}^{[1]})^2}{\sigma^2}, \qquad D^{[2]2} = \frac{(x_1 - {\mu_1}^{[2]})^2}{\sigma^2}$$

線形判別関数

$$z = \frac{(\mu_1^{[1]} - \mu_1^{[2]})}{\sigma^2}(x_1 - \bar{\mu}), \qquad \text{ただし} \bar{\mu} = \frac{(\mu_1^{[1]} + \mu_1^{[2]})}{2}$$

線形判別関数を用いた判別方式

$$z \geq 0 \ \Leftrightarrow \ D^{[1]2} \leq D^{[2]2} \ \Leftrightarrow \ \text{母集団 [1] に属する}$$

$$z < 0 \ \Leftrightarrow \ D^{[1]2} > D^{[2]2} \ \Leftrightarrow \ \text{母集団 [2] に属する}$$

母分散

$$\sigma^2 = \frac{S_{11}^{[1]} + S_{11}^{[2]}}{(n^{[1]} - 1) + (n^{[2]} - 1)}$$

誤判別の確率

$$Pr\left(u \geq \frac{\delta}{2\sigma}\right) \qquad \text{ただし} \delta = \mu_1^{[1]} - \mu_1^{[2]} > 0$$

判別効率

$$D_{x_1}^2([1],[2]) = \frac{(\mu_1^{[1]} - \mu_1^{[2]})^2}{\sigma^2}$$

4.2 適用判断

子役出現回数の確率分布は上の結果より, 正規分布と見なせるので判別分析を適用させることができる. ただし判別分析は同じ分散の正規分布 2 つに対して使える物であり, 今回は設定 1 の正規分布と設定 6 の正規分布の分散には, わずかに違いがある.これによりどのくらい誤差が生じるのかを調べて判別分析を適用させるかどうかを検討していく.

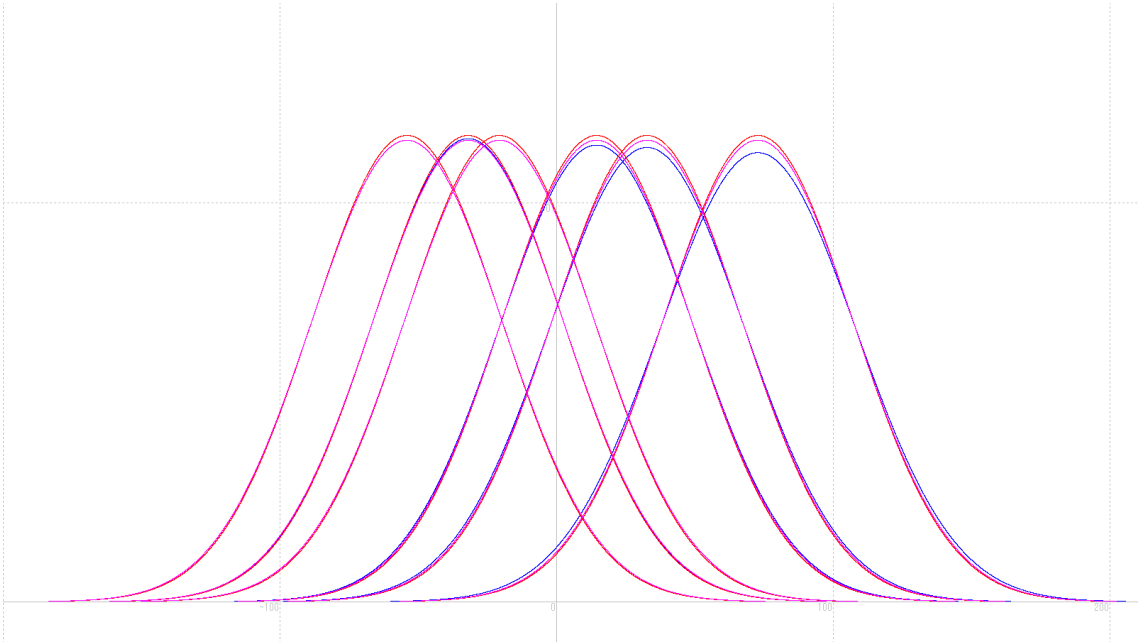
4.2.1 プログラムの実装

十進 BASIC によりプログラムを実装した.

```
SET bitmap SIZE 1600,900
SET WINDOW -200,210,-0.001,0.015
DRAW grid(100,0.01)
OPTION BASE 0
DIM p(6)
LET p(0)=1/7.43
LET p(1)=1/7.31
LET p(2)=1/7.25
LET p(3)=1/7.07
LET p(4)=1/6.98
LET p(5)=1/6.79
FOR j=0 TO 5
   LET m=p(j)
   FOR i=1200 TO 1650 STEP 0.01
      LET z=1/SQR(2*PI*10000*m*(1-m))*EXP(-(i-10000*m)
            *(i-10000*m)/(2*10000*m*(1-m)))
      IF z>0.00001 THEN
         SET LINE COLOR 2
         PLOT LINES:  i-1400,z;
      END IF
   NEXT i
   PLOT LINES
NEXT j
FOR j=0 TO 5
   LET m=p(j)
```

```
   FOR i=1200 TO 1650 STEP 0.01
      LET z=1/SQR(2*PI*10000*p(0)*(1-p(0)))*EXP(-(i-10000*m)
            *(i-10000*m)/(2*10000*p(0)*(1-p(0))))
      IF z>0.00001 THEN
         SET LINE COLOR 4
         PLOT LINES:  i-1400,z;
      END IF
   NEXT i
   PLOT LINES
NEXT j
FOR j=0 TO 5
   LET m=p(j)
   FOR i=1200 TO 1650 STEP 0.01
      LET z=1/SQR(2*PI*10000*p(2)*(1-p(2)))*EXP(-(i-10000*m)
            *(i-10000*m)/(2*10000*p(2)*(1-p(2))))
      IF z>0.00001 THEN
         SET LINE COLOR 7
         PLOT LINES:  i-1400,z;
      END IF
   NEXT i
   PLOT LINES
NEXT j
END
```

4.2.2 プログラムの実行結果



グラフ 6 : 設定 1 から設定 6 までの試行回数 10000 における子役出現回数の正規分布

青色のグラフが設定 1 から設定 6 までの試行回数 10000 における子役出現回数の正規分布であり, 赤色のグラフがすべて設定 1 の分散に合わせた場合のグラフで, 桃色のグラフがすべて設定 3 の分散に合わせた場合のグラフとなっている.したがって設定 1 と設定 3 では, 青色のグラフは重なっているので見えない.
判別分析が適用できるか考える.異なる設定の 2 つのグラフの重なっている面積が判別分析において重要である.いまグラフ 6 を見ると, 異なる設定の 2 つのグラフの重なっている面積は真の値と分散を同じに変えたものとで大きな誤差は見られないので判別分析が適用できると判断する.従って判別分析を行っていく.

5　判別分析の実行

正規分布の面積はシンプソン公式にて数値積分を行った値により求めていく.

5.1　プログラムの実装

```
#include<stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#define PI 6*asin( 0.5 )

int b[1];
double a[2],s[5],u[20],sigma[15],c[15],d[15];

double f(double x)
{
 double f;
 f=(double) exp((double) -x*x*0.5)/(double) sqrt(2*(PI));
 return f;
}

double simpson(double e)
{
 int n,i;
 double h,s;
 n=(int)(e/0.001)+3;
 h=e/(double) (n-1); s=0;
 for(i=1;i<n;i=i+2)
  {
   s=s+((double) ((f(h*(i-1))+4*f(h*i)+f(h*(i+1)))*h)/(double) 3);
  }
 return s;
}

void hanbetu(double s[],double u[],int b[])
{
```

```
 int i,j,k;
 k=0;
 for(i=0;i<5;i++)
  {
   for(j=i+1;j<=5;j++)
    {
     k=k+1;
     u[k+5]=(u[i]+u[j])/2.0;
     sigma[k-1]=(s[i]+s[j])/(20000-2);
     c[k-1]=(u[i]-u[j])/sigma[k-1];
     d[k-1]=u[i]-u[j];if(d[k-1]<0){d[k-1]=-d[k-1];}
     printf("設定%d と設定%d の線形判別関数  z=%fx+%f\n",i+1,j+1,c[k-1],-c[k-1]*u[k+5]);
     printf("判別値:     %f\n",u[k+5]);
     printf("誤判別の確率     %f\n",0.5-simpson(d[k-1]/(2*sqrt(sigma[k-1]))));
    }
  }
}

void settei1(double a[],int b[])
{
 int i,j,x;
 double y;
 for(j=1;j<10001;j++)
  {
   y=0;
   for(i=1;i<b[0]+1;i++)
    {
     x=(int) ((rand()/(RAND_MAX+1.0))*743);
     if(x<100){y=y+1;}
    }
   a[0]=a[0]+y;
   a[1]=a[1]+y*y;
  }
}
void settei2(double a[],int b[])
{
 int i,j,x;
 double y;
 for(j=1;j<10001;j++)
  {
   y=0;
   for(i=1;i<b[0]+1;i++)
    {
     x=(int) ((rand()/(RAND_MAX+1.0))*731);
```

```
     if(x<100){y=y+1;}
    }
   a[0]=a[0]+y;
   a[1]=a[1]+y*y;
  }
}
void settei3(double a[],int b[])
{
 int i,j,x;
 double y;
 for(j=1;j<10001;j++)
  {
   y=0;
   for(i=1;i<b[0]+1;i++)
    {
     x=(int) ((rand()/(RAND_MAX+1.0))*725);
     if(x<100){y=y+1;}
    }
   a[0]=a[0]+y;
   a[1]=a[1]+y*y;
  }
}

void settei4(double a[],int b[])
{
 int i,j,x;
 double y;
 for(j=1;j<10001;j++)
  {
   y=0;
   for(i=1;i<b[0]+1;i++)
    {
     x=(int) ((rand()/(RAND_MAX+1.0))*707);
     if(x<100){y=y+1;}
    }
   a[0]=a[0]+y;
   a[1]=a[1]+y*y;
  }
}

void settei5(double a[],int b[])
{
 int i,j,x;
 double y;
```

```
 for(j=1;j<10001;j++)
  {
   y=0;
   for(i=1;i<b[0]+1;i++)
    {
     x=(int) ((rand()/(RAND_MAX+1.0))*698);
     if(x<100){y=y+1;}
    }
   a[0]=a[0]+y;
   a[1]=a[1]+y*y;
  }
}

void settei6(double a[],int b[])
{
 int i,j,x;
 double y;
 for(j=1;j<10001;j++)
  {
   y=0;
   for(i=1;i<b[0]+1;i++)
    {
     x=(int) ((rand()/(RAND_MAX+1.0))*679);
     if(x<100){y=y+1;}
    }
   a[0]=a[0]+y;
   a[1]=a[1]+y*y;
  }
}

void main()
{
 int i;
 double j,k;
 srand((unsigned int)time(NULL));
 printf("試行回数は何回ですか？");
 scanf("%d",&b[0]);
 a[0]=0;a[1]=0;
 for(i=0;i<3;i++){s[i]=0;u[i]=0;}
 settei1(a,b);
 s[0]=a[1]-(a[0]*a[0]/(double) 10000);
 u[0]=a[0]/(double) 10000; a[0]=0; a[1]=0;
 settei2(a,b);
 s[1]=a[1]-(a[0]*a[0]/(double) 10000);
```

```
 u[1]=a[0]/(double) 10000; a[0]=0; a[1]=0;
 settei3(a,b);
 s[2]=a[1]-(a[0]*a[0]/(double) 10000);
 u[2]=a[0]/(double) 10000; a[0]=0; a[1]=0;
 settei4(a,b);
 s[3]=a[1]-(a[0]*a[0]/(double) 10000);
 u[3]=a[0]/(double) 10000; a[0]=0; a[1]=0;
 settei5(a,b);
 s[4]=a[1]-(a[0]*a[0]/(double) 10000);
 u[4]=a[0]/(double) 10000; a[0]=0; a[1]=0;
 settei6(a,b);
 s[5]=a[1]-(a[0]*a[0]/(double) 10000);
 u[5]=a[0]/(double) 10000; a[0]=0; a[1]=0;
 hanbetu(s,u,b);
}
```

5.2 プログラムの実行

実行結果をまとめると次の様になる.

設定	1	2	3	4	5	6
1		135.844450	136.289150	138.053500	138.898850	140.999750
2	135.844450		137.435300	139.199650	140.045000	142.145900
3	136.289150	137.435300		139.644350	140.489700	142.590600
4	138.053500	139.199650	139.644350		142.254050	144.354950
5	138.898850	140.045000	140.489700	142.254050		145.200300
6	140.999750	142.145900	142.590600	144.354950	145.200300	

表 2 : 各設定間の判別値 (試行回数 1000)

設定	1	2	3	4	5	6
1		0.457412	0.441591	0.378836	0.350130	0.283401
2	0.457412		0.483667	0.419611	0.389700	0.319808
3	0.441591	0.483667		0.435619	0.405844	0.334857
4	0.378836	0.419611	0.435619		0.469470	0.395570
5	0.350130	0.389700	0.405844	0.469470		0.425118
6	0.283401	0.319808	0.334857	0.395570	0.425118	

表 3 : 各設定間の誤判別の確率 (試行回数 1000)

設定	1	2	3	4	5	6
1		542.822300	544.926100	552.150700	555.961050	563.914900
2	542.822300		549.390700	556.615300	560.425650	568.379500
3	544.926100	549.390700		558.719100	562.529450	570.483300
4	552.150700	556.615300	558.719100		569.754050	577.707900
5	555.961050	560.425650	562.529450	569.754050		581.518250
6	563.914900	568.379500	570.483300	577.707900	581.518250	

表 4 : 各設定間の判別値 (試行回数 4000)

設定	1	2	3	4	5	6
1		0.418262	0.381012	0.264118	0.211084	0.122579
2	0.418262		0.461441	0.334281	0.273956	0.168415
3	0.381012	0.461441		0.371104	0.307832	0.194669
4	0.264118	0.334281	0.371104		0.431652	0.298003
5	0.211084	0.273956	0.307832	0.431652		0.360002
6	0.122579	0.168415	0.194669	0.298003	0.360002	

表 5 : 各設定間の誤判別の確率 (試行回数 4000)

上の表の使い方は, 試行回数 1000(4000) の時の子役出現確率が理論値と最も近い物をそのスロットマシンの設定であるとする.その上で, その設定と他の設定との誤判別の確率を各組み合わせごとに見て判断する.

6 結論

以上の結果から子役出現率による設定判別の優位性を考える.(このとき, 誤判別の確率が試行回数 4000 までなのは, 一日の試行回数が 8000 回程度が限界ということから, 設定を判別するならば 4000 回までには判断しなければあまり意味のない判別となることによる.) 表 3, 表 5 より試行回数 1000 回の時点では優位になるほどの判断はできず, 試行回数 4000 回の時点では高設定 (設定 $4, 5, 6$) か低設定 $(1, 2, 3)$ に関しては判断が優位になるが個別の設定を判断するほどの優位性は無いことが分かる.

7 今後の課題

今回使用した判別分析は, 二つの同じ分散の正規分布に対して適用できる物なので, 多数の正規分布の場合に適用できる重判別分析を行うプログラムを実装して, より精密な判別を行い, 個別の設定推測を行えるようにしていきたい.また, 実験対象とする機種を増やし, 多くの人に役立つデータを出していけたらと思っている.

8 参考文献

[1] 永田靖・棟近雅彦:多変量解析法入門, 2001 年, サイエンス社
[2]http://slot-navi.com/10/uruseiyatsura2.php