情報工学実験II

テーマ02
乱数を用いたプログラム

令和５年０７月０６日

イマム　カイリ　ルビス

学籍番号：214071

目 次

1　概要

1.1　乱数とは

乱数とは，ある数字の集合からランダムに選ばれた数字のことである．指定された分布内のすべての数値は，ランダムに選択される確率が等しくなります.

1.2　C 言語の乱数関数

本実験のコードはすべて C 言語で書かれている．C 言語にはすでに乱数関数が用意されているので，本実験ではそれを利用する．

`rand()` の関数は `stdlib.h` ヘッダーに含まれている．しかし，実行するたびに異なる乱数値を得るためには，`srand(time(NULL))` という関数も利用する必要がある．つまり，`time.h` ヘッダーファイルもインクルードする必要がある．[1]

1.3　確率とは

確率とはある事象の確率は，その事象が起こる可能性を示す数値である．その事象が起こる可能性が高ければ高いほど，確率値は高くなる．[2]

確率の基本的な計算は次の公式で計算される.

$$P(A) = \frac{f}{N} \tag{1}$$

事象 A が発生する確率を $P(A)$，事象 A が発生する可能性の数を f，可能な結果の合計数 N.

1.4　実行環境

本実験で使用される実行環境：

- プロセッサ：AMD Ryzen 5 5600X
- メモリー：16.0 GB
- OS：Windows 11 Pro
- コンパイラ：gcc

2　課題１：７個のサイコロを同じ出目になる確率

7 個のサイコロを一緒に振って，すべてのサイコロが同じ出目になる確率を計算する課題である．しかし，結果に変化を与えるために，4 面サイコロ，5 面サイコロ，6 面サイコロの確率を計算してみた.

2.1　同じ出目になる確率の理論値

同じ出目になる確率は以下の公式で計算することができる．

$$P = \left(\frac{1}{n}\right)^{m} \times n \tag{2}$$

サイコロの面の数を n，サイコロの数を m とする．

したがって，各サイコロ類の同じ出目になる確率は以下になる．

4面サイコロ

$$\begin{aligned} P &= \left(\frac{1}{4}\right)^{7} \times 4 \\ P &= \frac{1}{4096} \\ P &= 2.4414 \times 10^{-4} \\ P &= 0.024414\% \end{aligned} \tag{3}$$

5面サイコロ

$$\begin{aligned} P &= \left(\frac{1}{5}\right)^{7} \times 5 \\ P &= \frac{1}{15625} \\ P &= 6.4 \times 10^{-5} \\ P &= 0.006400\% \end{aligned} \tag{4}$$

6面サイコロ

$$\begin{aligned} P &= \left(\frac{1}{6}\right)^{7} \times 6 \\ P &= \frac{1}{46656} \\ P &= 2.1433 \times 10^{-5} \\ P &= 0.002143\% \end{aligned} \tag{5}$$

2.2　7個のサイコロのシミュレーションプログラム

以下は7個のサイコロをシミュレーションするプログラムである．

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

```

```c
#define diceCount 7

#define DEBUG 0

void setValue(int *dst, int value) {
    *dst = value;
}

int checkResult(int diceSide, int *diceResult, int *result) {
    for (int i = 1; i < diceCount; i++) {
        if(diceResult[0] == diceResult[i]) {
            continue;
        } else {
            return 0;
        }
    }
    return 1;
}

void startSimulation(int maxIteration, int count, int diceSide, int *result) {
    for (int i = 0; i < maxIteration; i++) {
        int *diceResult = malloc(diceCount * sizeof(int));
        // printf("%d | ", i+1);
        for (int j = 0; j < count; j++) {
            diceResult[j] = rand() % diceSide + 1;
            // printf("%d ", diceResult[j]);
        }
        *result += checkResult(diceSide, diceResult, result);

        #if DEBUG == 1
        if(checkResult(diceSide, diceResult, result)) {
            for (int j = 0; j < diceSide; j++) {
            printf("%d ", diceResult[j]);
            }
            printf("\n");
        }
        #endif
        // printf("| %d \n", *result);
        free(diceResult);
    }
}

double actual(int diceSide) {
    return 1/pow((double)diceSide, (double)diceCount-1);
}

double errorPercentage(double error, int diceSide) {
    double p = (error / actual(diceSide)) * 100;
    return p;
}

int main(int argc, char **argv) {
    srand(time(NULL));

    int diceSide;
    int maxIteration;
    int result = 0;
    const int repetitionCount = 3;

    if(argc != 2) {
        setValue(&diceSide, atoi(argv[1]));
        setValue(&maxIteration, atoi(argv[2]));
    } else {
        printf("Set dice sided and maxIteration number\n");
        return -2;
    }

```

```
73      char filename[20];
74      sprintf(filename, "dice%d.csv", diceSide);
75      printf("%s\n", filename);
76
77      FILE *p = fopen(filename, "w");
78      if(p == NULL) {
79          perror("File open error\n");
80          return -1;
81      }
82
83      const int iteration = (int)log10(maxIteration);
84      double *r = (double *)malloc(iteration * repetitionCount * sizeof(double));
85      double *error = (double *)malloc(repetitionCount * sizeof(double));
86
87      double *q  = r;
88      for (int i = 0; i < repetitionCount; i++) {
89          for (int count = 10; count <= maxIteration; count *= 10) {
90              startSimulation(count, diceCount, diceSide, &result);
91              *q = (double)result/count;
92              result = 0;
93              q++;
94          }
95      }
96
97      printf("count, actual, probability1, errorPercentage1, errorPercentage1,");
98      printf(" probability2, errorPercentage2, errorPercentage2,");
99      printf(" probability3, errorPercentage3, errorPercentage3,\n");
100
101     fprintf(p, "count, actual, probability1, errorPercentage1, errorPercentage1,"
        );
102     fprintf(p, " probability2, errorPercentage2, errorPercentage2,");
103     fprintf(p, " probability3, errorPercentage3, errorPercentage3,\n");
104
105     for (int i = 0; i < iteration; i++) {
106         printf("%d, %.10lf, ", (int)pow(10, i + 1), actual(diceSide)*100);
107         fprintf(p, "%d, %.10lf, ", (int)pow(10, i + 1), actual(diceSide)*100);
108         for (int j = 0; j < repetitionCount; j++) {
109             error[j] = r[i + j*iteration] - actual(diceSide);
110             printf("%.10lf, %.3lf, %.3lf%%, ", (r[i + j*iteration])*100, fabs(
        errorPercentage(error[j], diceSide)), fabs(errorPercentage(error[j], diceSide)
        ));
111             fprintf(p, "%.10lf, %.3lf, %.3lf%%, ", (r[i + j*iteration])*100, fabs
        (errorPercentage(error[j], diceSide)), fabs(errorPercentage(error[j], diceSide
        )));
112         }
113         printf("\n");
114         fprintf(p, "\n");
115     }
116
117     free(r);
118     free(error);
119     fclose(p);
120 }
```

2.3 7個のサイコロのシミュレーションの流れ

サイコロの目がすべて一致するかどうかを確認する手順は以下の通りになる.

1. 最初のサイコロを条件とする．
2. ２個目から７個目まで順次に条件を満たすかどうかをチェックする．
3. 全てのさいころが条件を満たしていれば成功．
4. 条件を満たさないサイコロを１個でも見つければ，チェック処理を中断する．

2.4 ７個のサイコロのシミュレーション結果

各サイコロ類について，最大 10^8 の反復で 3 回シミュレーションを行った．

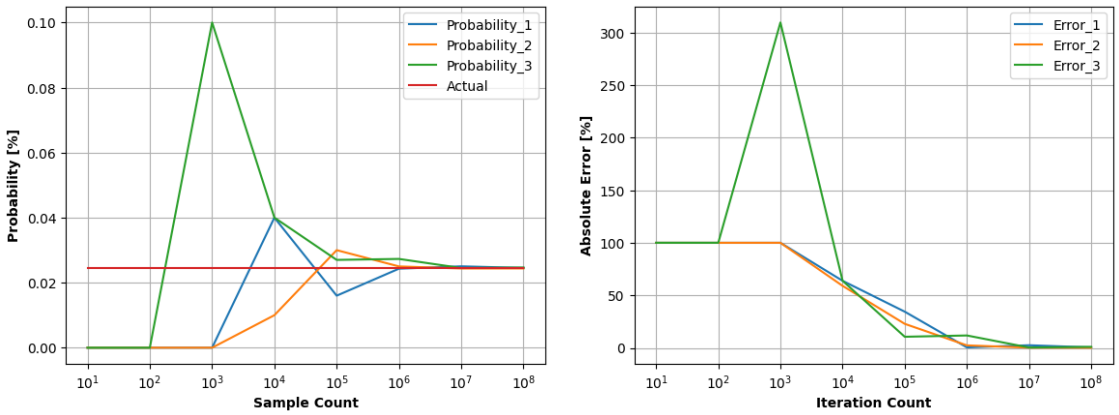
2.4.1 ４面サイコロのシミュレーション結果

以下は４面サイコロのシミュレーション結果である．

表 1: ４面サイコロのシミュレーション結果

Count	Probability_1	Error_1	Probability_2	Error_2	Probability_3	Error_3
10^1	0.000000%	100.00%	0.000000%	100.00%	0.000000%	100.00%
10^2	0.000000%	100.00%	0.000000%	100.00%	0.000000%	100.00%
10^3	0.000000%	100.00%	0.000000%	100.00%	0.100000%	309.60%
10^4	0.040000%	63.84%	0.010000%	59.04%	0.040000%	63.84%
10^5	0.016000%	34.46%	0.030000%	22.88%	0.027000%	10.59%
10^6	0.024300%	0.47%	0.025000%	2.40%	0.027300%	11.82%
10^7	0.025030%	2.52%	0.024360%	0.22%	0.024500%	0.35%
10^8	0.024524%	0.45%	0.024372%	0.17%	0.024659%	1.00%

グラフで表すと以下のグラフになる．



(a) 確率値　　(b) 絶対誤差

図 1: ４面サイコロの確率値と絶対誤差

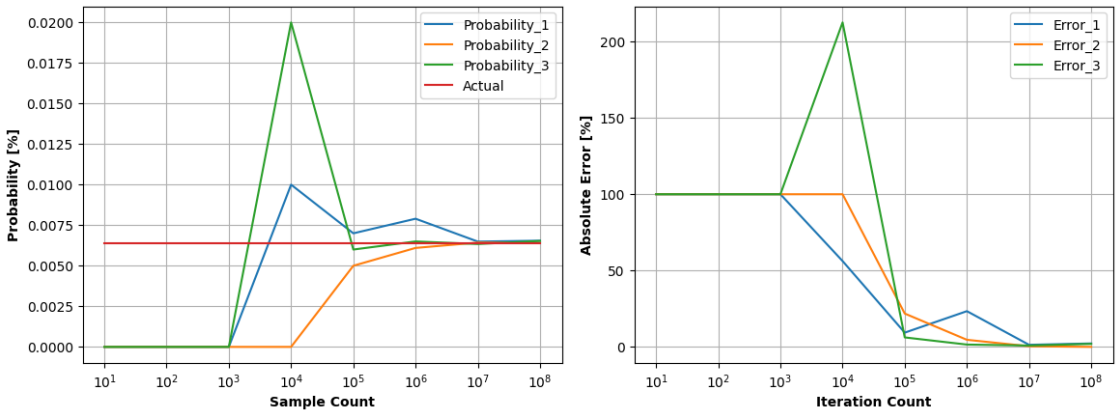
2.4.2　5面サイコロのシミュレーション結果

以下は5面サイコロのシミュレーション結果である.

表 2: 5面サイコロのシミュレーション結果

Count	Probability_1	Error_1	Probability_2	Error_2	Probability_3	Error_3
10^1	0.000000%	100.00%	0.000000%	100.00%	0.000000%	100.00%
10^2	0.000000%	100.00%	0.000000%	100.00%	0.000000%	100.00%
10^3	0.000000%	100.00%	0.000000%	100.00%	0.000000%	100.00%
10^4	0.010000%	56.25%	0.000000%	100.00%	0.020000%	212.50%
10^5	0.007000%	9.38%	0.005000%	21.88%	0.006000%	6.25%
10^6	0.007900%	23.44%	0.006100%	4.69%	0.006500%	1.56%
10^7	0.006490%	1.41%	0.006430%	0.47%	0.006340%	0.94%
10^8	0.006547%	2.30%	0.006403%	0.05%	0.006529%	2.02%

グラフで表すと以下のグラフになる.



(a) 確率値　　(b) 絶対誤差

図 2: 5面サイコロの確率値と絶対誤差

2.4.3　6面サイコロのシミュレーション結果

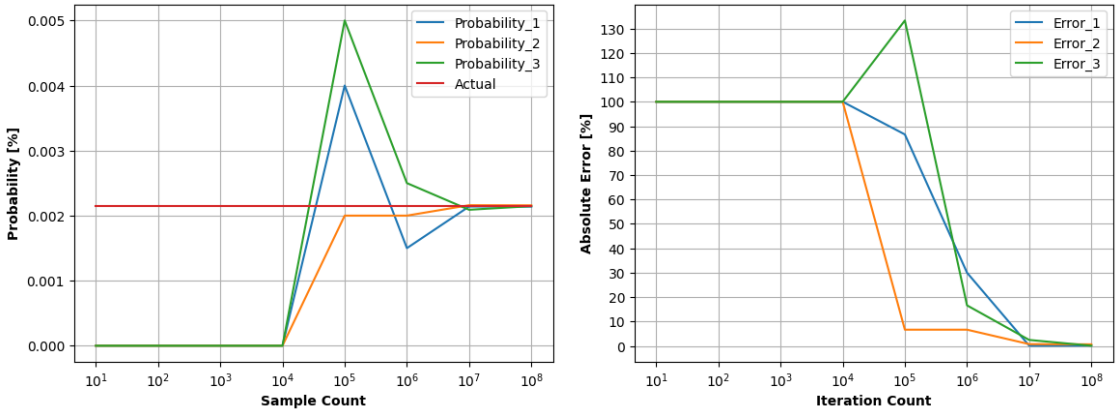
以下は6面サイコロのシミュレーション結果である.

表 3: 6面サイコロのシミュレーション結果

Count	Probability_1	Error_1	Probability_2	Error_2	Probability_3	Error_3
10^1	0.000000%	100.000%	0.000000%	100.000%	0.000000%	100.000%
10^2	0.000000%	100.000%	0.000000%	100.000%	0.000000%	100.000%
10^3	0.000000%	100.000%	0.000000%	100.000%	0.000000%	100.000%
10^4	0.000000%	100.000%	0.000000%	100.000%	0.000000%	100.000%

10^5	0.004000%	86.624%	0.002000%	6.688%	0.005000%	133.280%
10^6	0.001500%	30.016%	0.002000%	6.688%	0.002500%	16.640%
10^7	0.002140%	0.156%	0.002160%	0.777%	0.002090%	2.489%
10^8	0.002139%	0.203%	0.002158%	0.684%	0.002146%	0.124%

グラフで表すと以下のグラフになる．



(a) 確率値　　(b) 絶対誤差

図 3: 6 面サイコロの確率値と絶対誤差

2.5　7 個のサイコロのシミュレーションの考察

4 面サイコロ，5 面サイコロ，6 面サイコロのどのシミュレーションも，反復回数が増えるにつれて誤差が小さくなっています．図 (1)，図 (2)，図 (3) を見ると，反復回数が 10^7 とき，全ての誤差が小さくなっていることがわかる．

しかし，4 面サイコロの 1 回目と 2 回目のシミュレーションでは，図 (1(b)) ように反復回数 10^6 のとき，誤差が小さくなるものもある．しかし，3 回目のシミュレーションでは誤差が大きくなっている．同じことが 5 面サイコロでも起こり，反復回数が 10^6 最初のシミュレーションでは大きな誤差が出ました．最後の 6 面サイコロでは，誤差は小さくならなかった．つまり，正確な結果を得るためには，反復回数が 10^6 では十分ではないという結論になる．

3　課題２：円の面積

円の面積を計算するには，乱数によって点をプロットし，その点が円の中にあるかあるかどうかをを確認する．そして，四角形の面積と円の面積を比較する．計算過程は以下のようになる．

$$\frac{\text{円の面積}}{\text{四角形の面積}} = \frac{a}{a+b} \tag{6}$$

$$\text{円の面積} = \text{四角形の面積} \times \frac{a}{a+b} \tag{7}$$

円の内側の点の数を a，円の外側の点の数を b とする．

3.1 円の面積のシミュレーションプログラム

以下は7個のサイコロをシミュレーションするプログラムである．

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

double getPercentage() {
    double x = rand() % 200;
    x -= 100;
    return x/100;
}

void setPoint(double *point, int r) {
    for (int i = 0; i < 2; i++) {
        double percentage = getPercentage();
        point[i] = percentage*r;
    }
}

double square(double r) {
    return r*r;
}

double circle(double *point){
    return point[0]*point[0] + point[1]*point[1];
}

int insideCircle(double *point, int r){
    if (circle(point) <= r*r) return 1;
    else return 0;
}

double circleAreaApprox(int insideCount, int outsideCount, int r) {
    return ((double)(insideCount)/(double)(insideCount + outsideCount)) * square
    (2*r);
}

void printResult(int insideCount, int outsideCount, int r) {
    printf("in: %d, out: %d\n", insideCount, outsideCount);
    printf("area : %lf\n", circleAreaApprox(insideCount, outsideCount, r));
}

double actualArea(int r) {
    return M_PI*r*r;
}

void startCalculation(int *insideCount, int *outsideCount, int r, int
    maxIteration) {
    double *point = (double *)malloc(2 * sizeof(double));

    for (int i = 0; i < maxIteration; i++) {
        setPoint(point, r);
```

```
        if(insideCircle(point, r)) *insideCount = *insideCount + 1;
        else *outsideCount = *outsideCount + 1;
    }

    free(point);
}

void resetValue(int *inside, int *outside) {
    *inside = 0;
    *outside = 0;
}

double errorPercentage(double error, int r) {
    return (error / actualArea(r)) * 100;
}

int main (int argc, char **argv) {
    srand(time(NULL));

    int r;
    int maxIteration;
    int insideCount = 0;
    int outsideCount = 0;
    const int repetitionCount = 3;

    if (argc == 3) {
        r = atoi(argv[1]);
        maxIteration = atoi(argv[2]);
    } else {
        printf("Argument are missing\n");
        return 1;
    }

    char filename[20];
    sprintf(filename, "area%d.csv", r);
    printf("%s\n", filename);

    FILE *p = fopen(filename, "w");

    if(p == NULL) {
        perror("File open error\n");
        return -1;
    }

    const int iteration = (int)log10(maxIteration);
    double *area = (double *)malloc(iteration * repetitionCount * sizeof(double))
    ;
    double *error = (double *)malloc(repetitionCount * sizeof(double));

    double *q = area;
    for (int i = 0; i < repetitionCount; i++) {
        for (int count = 10; count <= maxIteration; count *= 10) {
            startCalculation(&insideCount, &outsideCount, r, count);
            *q = circleAreaApprox(insideCount, outsideCount, r);
            q++;
            resetValue(&insideCount, &outsideCount);
        }
    }

    printf("Sample Count, Actual, Result_1, Error_1,  Error_1,");
    printf("Result_2, Error_2, Error_2, ");
    printf("Result_3, Error_3, Error_3, \n");

    fprintf(p, "Sample Count, Actual, Result_1, Error_1,  Error_1,");
    fprintf(p, "Result_2, Error_2, Error_2, ");
    fprintf(p, "Result_3, Error_3, Error_3, \n");

```

```
116     for (int i = 0; i < iteration; i++) {
117         printf("%d, %.3lf, ", (int)pow(10, i + 1), actualArea(r));
118         fprintf(p, "%d, %.3lf, ", (int)pow(10, i + 1), actualArea(r));
119         for (int j = 0; j < repetitionCount; j++) {
120             error[j] = area[i + j*iteration] - actualArea(r);
121             printf("%.3lf, %.3lf, %.3lf%%, ", area[i + j*iteration], fabs(
    errorPercentage(error[j], r)), fabs(errorPercentage(error[j], r)));
122             fprintf(p, "%.3lf, %.3lf, %.3lf%%, ", area[i + j*iteration], fabs(
    errorPercentage(error[j], r)), fabs(errorPercentage(error[j], r)));
123         }
124         printf("\n");
125         fprintf(p, "\n");
126     }
127
128     free(area);
129     free(error);
130     fclose(p);
131 }
```

3.2 円の面積のシミュレーションの流れ

サイコロの目がすべて一致するかどうかを確認する手順は以下の通りになる．

1. 乱数によって点をプロットする．
2. 円の内側と外側にプロットされている点を数える．
3. 円の面積と四角形を比較して，円の面積を得る．

3.3 円の面積のシミュレーション結果

本実験で計算された円の半径は $7m$ と $10m$ とする．各円の半径の大きさに分れて，最大 10^8 の反復で 3 回シミュレーションを行った．

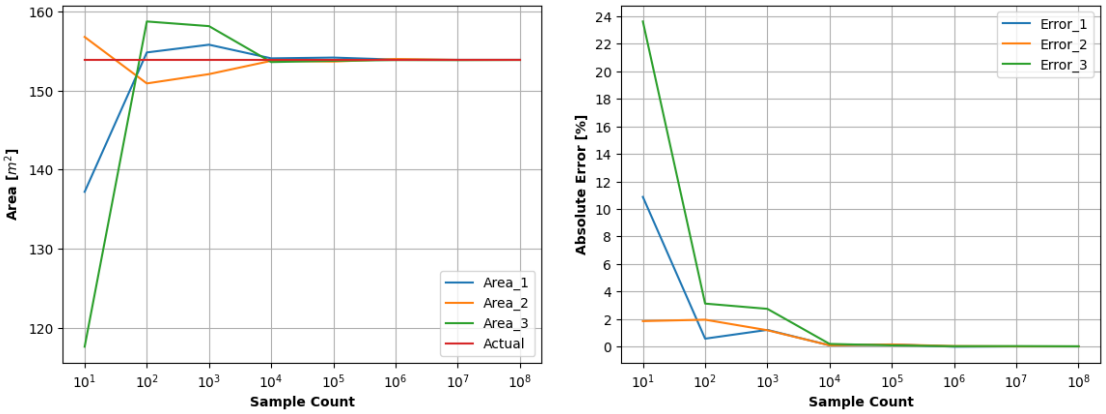
3.3.1 半径 $7m$ の円のシミュレーション結果

以下は半径 $7m$ の円のシミュレーション結果である．

表 4: 半径 $7m$ の円のシミュレーション結果

Sample Count	Result_1 $[m^2]$	Error_1	Result_2 $[m^2]$	Error_2	Result_3 $[m^2]$	Error_3
10^1	137.200	10.873%	156.800	1.859%	117.600	23.606%
10^2	154.840	0.586%	150.920	1.961%	158.760	3.132%
10^3	155.820	1.223%	152.096	1.197%	158.172	2.750%
10^4	154.095	0.102%	153.782	0.102%	153.625	0.203%
10^5	154.193	0.166%	153.688	0.163%	153.787	0.098%
10^6	153.940	0.001%	154.009	0.046%	153.879	0.038%
10^7	153.907	0.020%	153.901	0.024%	153.870	0.044%
10^8	153.907	0.020%	153.903	0.023%	153.890	0.031%

グラフで表すと以下のグラフになる．



(a) 面積　　(b) 絶対誤差

図 4: 半径 $7m$ の円の面積と絶対誤差

3.3.2　半径 $10m$ の円のシミュレーション結果

以下は半径 $10m$ の円のシミュレーション結果である．

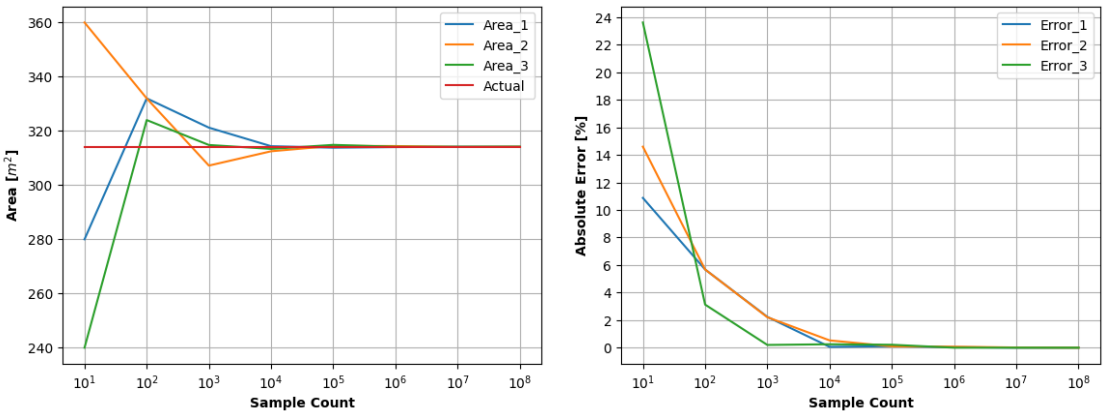
表 5: 半径 $10m$ の円のシミュレーション結果

Sample Count	Result_1	Error_1	Result_2	Error_2	Result_3	Error_3
10^1	280.000	10.873%	360.000	14.592%	240.000	23.606%
10^2	332.000	5.679%	332.000	5.679%	324.000	3.132%
10^3	321.200	2.241%	307.200	2.215%	314.800	0.204%
10^4	314.360	0.064%	312.480	0.535%	313.360	0.254%
10^5	313.804	0.113%	314.480	0.102%	314.840	0.217%
10^6	314.026	0.042%	314.402	0.077%	314.145	0.004%
10^7	314.149	0.003%	314.104	0.018%	314.139	0.006%
10^8	314.144	0.005%	314.168	0.003%	314.168	0.003%

グラフで表すと以下のグラフになる．

3.4　円の面積のシミュレーションの考察

図 (4) と図 (5) からシミュレーションの結果は，サンプル数が増えるにつれて，より正確になっていくことがわかる．サンプル数が 10^4 のとき，絶対誤差が 1% 未満になる．サンプル数を 10^6 ポイントでシミュレーションを行うと，誤差は 0.1% 以下になる．しかし，サンプルをそれ以上に増やしても，誤差は有意な減少を示さない．



(a) 面積　　(b) 絶対誤差

図 5: 半径 $10m$ の円の面積と絶対誤差

したがって，作業負荷と精度を考慮すると，10^6 サンプルを使用するのが最も効率的．つまり作業量が少なく精度が高い．

4　課題３：１０名の男女横一列に並べる

女性 5 人と男性 5 人の計 10 人が横一列に並べる．順番は完全に乱数によって決まる．しかし，期待される並び方は，各女性に対して，自身より背の高い男性が左に少なくとも一人いる．ということ，この課題はこの並び方が起こった確率計算する課題である．

4.1　横一列に並べるシミュレーションプログラム

以下は１０名を横一列に並べるをシミュレーションするプログラムである．

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

#define SIZE 5

#define DEBUG 0

void allocateValue(int *dst) {
    for (int i = 1; i <= 2*SIZE; i++) {
        *dst = i;
        dst++;
    }
}

void printData(int *dst) {
    for (int i = 0; i < 2 * SIZE; i++) {
        printf("%d ", *dst);
        dst++;
    }
    printf("\n");
}

```

```
25 void swapElement(int *dst, int a, int b) {
26     int tmp = dst[a];
27     dst[a] = dst[b];
28     dst[b] = tmp;
29 }
30
31 void shuffle(int *dst) {
32     for (int i = 2*SIZE - 1; i >= 0; i--) {
33         int target = rand() % SIZE;
34         swapElement(dst, i, target);
35     }
36 }
37
38 void combineValue(int *dst, int *m, int *w) {
39     for (int i = 0; i < SIZE; i++) {
40         *dst = *m;
41         dst++;
42         *dst = *w;
43         dst++;
44
45         m++;
46         w++;
47     }
48 }
49
50 int check(int *dst) {
51     int max;
52     if (dst[0] == 10) return 1;
53     if (dst[0] % 2 == 1) return 0;
54     else {
55         max = dst[0];
56         for (int i = 1; i < 2*SIZE; i++) {
57             if (dst[i] % 2 == 1) {
58                 if (max > dst[i]) continue;
59                 else return 0;
60             } else {
61                 if (dst[i] > max) max = dst[i];
62             }
63         }
64     }
65     return 1;
66 }
67
68 void startCalculation(int maxIteration, int *result) {
69     int *combine = (int *)malloc(2*SIZE * sizeof(int));
70     allocateValue(combine);
71
72     for (int i = 0; i < maxIteration; i++) {
73         shuffle(combine);
74
75         *result += check(combine);
76         #if DEBUG == 1
77             if(!check(combine)) {
78                 for(int j = 0; j < 10; j++) {
79                     printf("%d ", combine[j]);
80                 }
81                 printf("\n\n");
82             }
83         #endif
84     }
85
86     free(combine);
87 }
88
89 double actual() {
90     return (double)(1*3*5*7*9)/(2*4*6*8*10);
91 }
```

```
92
93  double errorPercentage(double error) {
94      double p = (error / actual()) * 100;
95      return p;
96  }
97
98  int main(int argc, char **argv) {
99      srand(time(NULL));
100
101     int maxIteration;
102
103     if (argc == 2) {
104         maxIteration = atoi(argv[1]);
105     } else {
106         printf("Set Iteration Count");
107         return 1;
108     }
109
110     int result = 0;
111     FILE *p = fopen("height.csv", "w");
112
113     const int repetitionCount = 3;
114     const int iteration = (int)log10(maxIteration);
115     double *r = (double *)malloc(iteration * repetitionCount * sizeof(double));
116     double *error = (double *)malloc(repetitionCount * sizeof(double));
117
118     double *q = r;
119     for (int i = 0; i < repetitionCount; i++) {
120         for (int count = 10; count <= maxIteration; count *= 10) {
121             startCalculation(count, &result);
122             *q = (double)result/(double)count;
123             result = 0;
124             q++;
125         }
126     }
127
128     printf("count, actual, ");
129     printf("probability_1, error_1, error_1, ");
130     printf("probability_2, error_2, error_2, ");
131     printf("probability_3, error_3, error_3,\n");
132
133     fprintf(p, "count, actual, ");
134     fprintf(p, "probability_1, error_1, error_1, ");
135     fprintf(p, "probability_2, error_2, error_2, ");
136     fprintf(p, "probability_3, error_3, error_3,\n");
137     for (int i = 0; i < iteration; i++) {
138         printf("%d, %.10lf, ", (int)pow(10, i+1), actual()*100);
139         fprintf(p, "%d, %.10lf, ", (int)pow(10, i+1), actual()*100);
140         for (int j = 0; j < repetitionCount; j++) {
141             error[j] = r[i + j*iteration] - actual();
142             printf("%.10lf, %.3lf, %.3lf%%, ", (r[i + j*iteration])*100, fabs(
    errorPercentage(error[j])), fabs(errorPercentage(error[j])));
143             fprintf(p, "%.10lf, %.3lf, %.3lf%%, ", (r[i + j*iteration])*100, fabs
    (errorPercentage(error[j])), fabs(errorPercentage(error[j])));
144         }
145         printf("\n");
146         fprintf(p, "\n");
147     }
148
149     free(r);
150     free(error);
151     fclose(p);
152 }
```

4.2　１０名を横一列に並べるシミュレーションの流れ

乱数できめた並び方を確認する手順は以下の通りになる．

1. 乱数で全員の位置を決定する．
2. 各女性に左に立っている男性と身長を比較する．
3. 条件を満たさない女性を１人でも見つければ，チェック処理を中断する．

4.3　１０名を横一列に並べるの理論値

この問題で期待される条件の並び方が起こった確率を数学的に計算すると，次のようになる．

$$P = \frac{1 \times 3 \times 5 \times 7 \times 9}{2 \times 4 \times 6 \times 8 \times 10} \tag{8}$$

$$P = 0.24609 \tag{9}$$

$$P = 24.609375\% \tag{10}$$

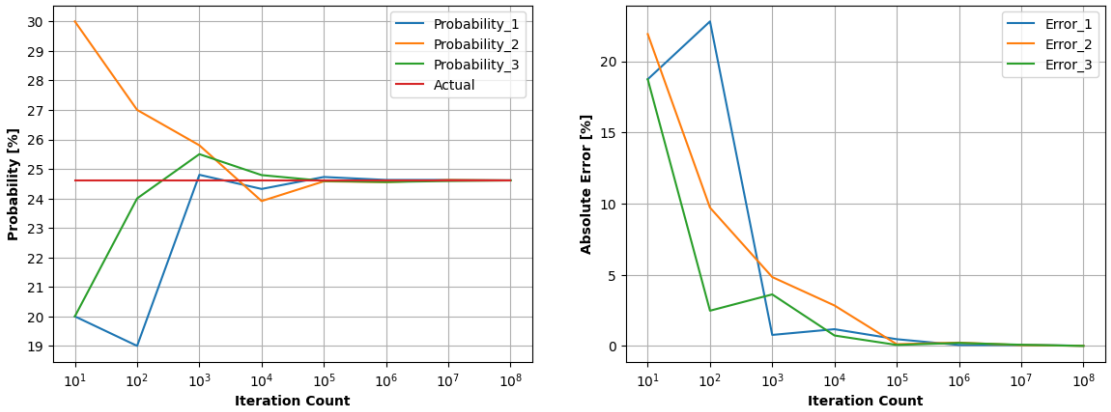
4.4　１０名を横一列に並べるシミュレーションの結果

この課題をシミュレーションを行うとき，最大 10^8 の反復で 3 回シミュレーションを行った．その結果は以下になる．

表 6: １０名を横一列に並べるシミュレーションの結果

Count	Probability_1	Error_1	Probability_2	Error_2	Probability_3	Error_3
10^1	20.000000%	18.730%	30.000000%	21.905%	20.000000%	18.730%
10^2	19.000000%	22.794%	27.000000%	9.714%	24.000000%	2.476%
10^3	24.800000%	0.775%	25.800000%	4.838%	25.500000%	3.619%
10^4	24.320000%	1.176%	23.910000%	2.842%	24.790000%	0.734%
10^5	24.726000%	0.474%	24.579000%	0.123%	24.592000%	0.071%
10^6	24.626300%	0.069%	24.551900%	0.234%	24.557400%	0.211%
10^7	24.626050%	0.068%	24.624070%	0.060%	24.589280%	0.082%
10^8	24.612748%	0.014%	24.611035%	0.007%	24.608419%	0.004%

グラフで表すと以下のグラフになる．



(a) 確率値　　(b) 絶対誤差

図 6:　１０名の男女横一列に並べる確率

4.5　１０名を横一列に並べるシミュレーションの考察

表 (6) をから見ると，最初にすべての絶対誤差が 0.1% 未満になったのは，反復回数が 10^7 回のときである．つまり，この課題は 24.626% の確率で発生し，正確な結果を得るには 10^7 回のシミュレーションが必要という結論になる．

5　課題４：コインゲーム

サイコロを使ったゲームで，1 か 2 の目が出るたびに B が A にコインを 1 枚渡し，それ以外出たら A が B にコインを 1 枚渡す．これを繰り返して，最後にコインを全部持っている方が勝つ．この課題では，A がゲームに勝つ確率を計算することが目的である．

5.1　コインゲームのシミュレーションプログラム

以下はコインゲームをシミュレーションするプログラムである．

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

#define DEBUG 0

int rollDice() {
    return rand() % 6 + 1;
}

int check(int dice) {
    if (dice == 1 || dice == 2) return 1;
    else return 0;
}

```

```
17 void giveCoin(int *dst, int *src) {
18     *dst = *dst + 1;
19     *src = *src - 1;
20 }
21
22 void startSimulation(int *A, int *B) {
23     while(*A != 0 && *B != 0) {
24         int dice = rollDice();
25         if (check(dice)) giveCoin(A, B);
26         else giveCoin(B, A);
27
28         #if DEBUG == 1
29             printf("A = %d, B = %d\n", *A, *B);
30         #endif
31     }
32 }
33
34 void updateWinCount(int coin, int *winCount) {
35     if(coin) *winCount = *winCount + 1;
36     else return;
37 }
38
39 void resetCoin(int *A, int *B, int startA, int startB) {
40     *A = startA;
41     *B = startB;
42 }
43
44 double printAWinPercentage(int winA, int count) {
45     return (double)(winA)/(double)(count);
46 }
47
48 void startCalculation(int *A, int *B, int *winA, int *winB, int maxIteration, int
       startA, int startB) {
49     for (int i = 0; i < maxIteration; i++) {
50         resetCoin(A, B, startA, startB);
51         startSimulation(A, B);
52         updateWinCount(*A, winA);
53         updateWinCount(*B, winB);
54     }
55 }
56
57 void resetValue(int *coinA, int *coinB, int *winA, int *winB, int startA, int
       startB) {
58     *coinA = startA;
59     *coinB = startB;
60     *winA = 0;
61     *winB = 0;
62 }
63
64 int main(int argc, char **argv) {
65     srand(time(NULL));
66
67     int maxIteration;
68     int startA;
69     if (argc == 3) {
70         maxIteration = atoi(argv[1]);
71         startA = atoi(argv[2]);
72     } else {
73         printf("Set maxIteration count\n");
74         return 1;
75     }
76
77     int startB = 8 - startA;
78
79     int coinA;
80     int coinB;
81     int winA;
```

```
82      int winB;
83      resetValue(&coinA, &coinB, &winA, &winB, startA, startB);
84
85      char filename[20];
86      sprintf(filename, "coin%d.csv", startA);
87      printf("%s\n", filename);
88
89      FILE *p = fopen(filename, "w");
90
91      if(p == NULL) {
92          perror("File open error\n");
93          return -1;
94      }
95
96      const int repetitionCount = 3;
97      const int iteration = (int)log10(maxIteration);
98      double *r = (double *)malloc(iteration * repetitionCount * sizeof(double));
99
100     double *q = r;
101     for (int i = 0; i < repetitionCount; i++) {
102         for (int count = 10; count <= maxIteration; count *= 10) {
103             startCalculation(&coinA, &coinB, &winA,
104             winB, count, startA, startB);
105             *q = printAWinPercentage(winA, count);
106             q++;
107             resetValue(&coinA, &coinB, &winA, &winB, startA, startB);
108         }
109     }
110
111     printf("Count, Probability_1, Probability_2, Probability_3,\n");
112     fprintf(p, "Count, Probability_1, Probability_2, Probability_3,\n");
113
114     for (int i = 0; i < iteration; i++) {
115         printf("%d, ", (int)pow(10, i+1));
116         fprintf(p, "%d, ", (int)pow(10, i+1));
117         for (int j = 0; j < repetitionCount; j++) {
118             printf("%.10lf, ", r[i + j*iteration]*100);
119             fprintf(p, "%.10lf, ", r[i + j*iteration]*100);
120         }
121         printf("\n");
122         fprintf(p, "\n");
123     }
124
125
126     free(r);
127     fclose(p);
128
129     return 0;
130 }
```

5.2 コインゲームのシミュレーション結果

A のコインの枚数の違いによる影響を知るために，コインの枚数で分けて，シミュレーションを行った．各シミュレーションは 3 回行った．

5.2.1 A が 5 枚コインで始まる

以下は A が 5 枚コインで始まったゲームのシミュレーション結果である．

表 7: A が 5 枚コインで始まる結果

Count	Probability_1	Probability_2	Probability_3
10^1	0.00000%	20.00000%	20.00000%
10^2	15.00000%	6.00000%	13.00000%
10^3	11.90000%	11.90000%	12.50000%
10^4	11.60000%	12.27000%	12.52000%
10^5	12.04900%	12.06100%	12.25900%
10^6	12.16000%	12.14950%	12.14660%
10^7	12.13316%	12.15369%	12.17129%
10^8	12.16059%	12.15940%	12.15675%

グラフで表すと以下のグラフになる.

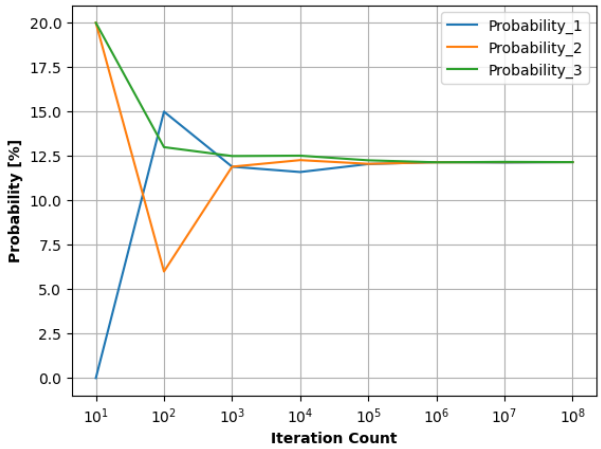


図 7: A が 5 枚コインで始まった勝つ確率

5.2.2 A が 6 枚コインで始まる

以下は A が 6 枚コインで始まったゲームのシミュレーション結果である.

表 8: A が 6 枚コインで始まる結果

count	Probability_1	Probability_2	Probability_3
10^1	10.000000%	30.000000%	40.000000%
10^2	20.000000%	29.000000%	21.000000%
10^3	23.700000%	24.900000%	26.700000%
10^4	25.450000%	24.530000%	24.890000%
10^5	24.631000%	24.569000%	24.560000%
10^6	24.606900%	24.746600%	24.748100%

10^7	24.699580%	24.708590%	24.720960%
10^8	24.712858%	24.701537%	24.711713%

グラフで表すと以下のグラフになる.

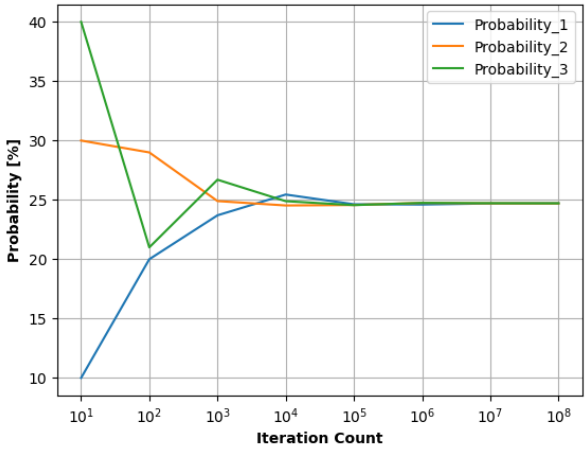


図 8: A が 6 枚コインで始まった勝つ確率

5.2.3 A が 7 枚コインで始まる

以下は A が 7 枚コインで始まったゲームのシミュレーション結果である.

表 9: A が 7 枚コインで始まる結果

count	Probability_1	Probability_2	Probability_3
10^1	50.000000%	40.000000%	60.000000%
10^2	51.000000%	51.000000%	52.000000%
10^3	47.200000%	50.700000%	49.400000%
10^4	50.200000%	50.450000%	49.400000%
10^5	49.906000%	49.861000%	49.743000%
10^6	49.863900%	49.828300%	49.811800%
10^7	49.803250%	49.803660%	49.798380%
10^8	49.809443%	49.814332%	49.802907%

グラフで表すと以下のグラフになる．

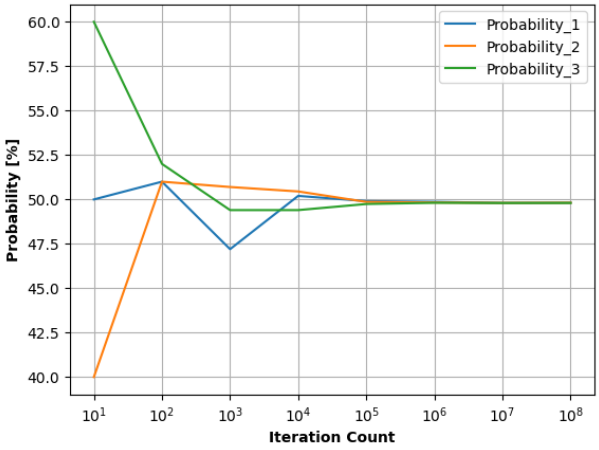


図 9: A が 7 枚コインで始まった勝つ確率

5.3　コインゲームの考察

表 (7)，表 (8)，表 (9) から見ると，コイン数が変わると A が勝つ確率は以下のようになる．

- A が 5 枚コインで始まるかつ確率：12.16%
- A が 6 枚コインで始まるかつ確率：24.70%
- A が 7 枚コインで始まるかつ確率：49.80%

A が最初に持たせるコインの数が最大でも，B がゲームに勝つ確率の方が高い．つまり，サイコロの目が 1 か 2 のときだけ A にコインを出すという条件が，ゲームの勝率に最も大きな影響を与えていることがわかる．

6　課題５：ビンゴゲーム

この課題の目的は，ビンゴをシミュレートし，4 回以内に当たる確率，7 回以内に当たる確率，ビンゴゲームで起こる最大回数，最大回数が起こる確率を計算することである．

ビンゴになるには，横・館・斜めのいずれか 1 列にある 5 マスが揃って有効になった場合に勝利となり．

6.1　ビンゴのシミュレーションプログラム

以下はコインゲームをシミュレーションするプログラムである．

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define SIZE 5
#define INTERVAL 15
// #define MAXVALUE 500
#define MAXVALUE 75

void printBoard(int *board) {
    for (int i = 1; i <= SIZE * SIZE; i++) {
        printf("%2d ", *board);
        board++;
        if(i % 5 == 0) printf("\n");
    }
}

void initiateDummy(int *dummy, int scale) {
    for (int i = 0; i < INTERVAL; i++) {
        dummy[i] = i + INTERVAL*scale + 1;
    }
}

void swapElement(int *dst, int a, int b) {
    int tmp = dst[a];
    dst[a] = dst[b];
    dst[b] = tmp;
}

void shuffle(int *dummy, int size) {
    for (int i = size - 1; i >= 0; i--) {
        int target = rand() % size;
        swapElement(dummy, i, target);
    }
}

void makeBoard(int *board) {
    int *dummy = (int *)malloc(INTERVAL * sizeof(int));

    for (int i = 0; i < SIZE; i++) {
        initiateDummy(dummy, i);
        shuffle(dummy, INTERVAL);

        int *p = dummy;
        int *q = board;

        for (int j = 0; j < SIZE; j++) {
            *board = *p;
            p++;
            board += SIZE;
        }

        board = ++q;
    }
    free(dummy);

    board += SIZE + 2;
    *board = 0;
}

void printCheck(int *boardCheck) {
    int *p = boardCheck;

    printf("==============\n");

    for (int i = 1; i <= SIZE * SIZE; i++) {
```

```
67          printf("%2d ", *p);
68          p++;
69          if(i % 5 == 0) printf("\n");
70      }
71
72      printf("==============\n");
73  }
74
75  int checkRow(int *boardCheck) {
76      int *p = boardCheck;
77      int *q = boardCheck;
78      int count = 0;
79      for (int i = 0; i < SIZE; i++) {
80          for (int j = 0; j < SIZE; j++) {
81              if(*p == 1) {
82                  p++;
83                  count ++;
84                  continue;
85              } else break;
86          }
87          if (count == 5) return 1;
88          count = 0;
89          q += SIZE;
90          p = q;
91      }
92      return 0;
93  }
94
95  int checkCol(int *boardCheck) {
96      int *p = boardCheck;
97      int *q = boardCheck;
98      int count = 0;
99      for (int i = 0; i < SIZE; i++) {
100         for (int j = 0; j < SIZE; j++) {
101             if(*p == 1) {
102                 p += SIZE;
103                 count ++;
104                 continue;
105             } else break;
106         }
107         if (count == 5) return 1;
108         count = 0;
109         q ++;
110         p = q;
111     }
112     return 0;
113 }
114
115 int checkDia(int *boardCheck) {
116     int *p = boardCheck;
117     int *q = boardCheck + 4;
118
119     int count = 0;
120     int inc = 1;
121
122     for (int i = 0; i < 2; i++) {
123         for (int j = 0; j < SIZE; j++) {
124             if (*p == 1) count ++;
125             else break;
126             p += SIZE + inc;
127         }
128         if (count == 5) return 1;
129         count = 0;
130         p = q;
131         inc = -1;
132     }
133
```

```
134     return 0;
135 }
136 
137 void updateCheck(int *boardCheck, int offset) {
138     if (offset < SIZE*SIZE){
139         int *p = boardCheck;
140         p += offset;
141         *p = 1;
142     }
143 }
144 
145 int searchBoard(int *boardValue, int target) {
146     int *p = boardValue;
147     int *q = boardValue;
148     int row = (target - 1) / 15;
149 
150     p += row;
151     for (int i = 0; i < SIZE; i++) {
152         if(*p == target) break;
153         p += SIZE;
154     }
155 
156     int dif = p - q;
157 
158     return dif;
159 }
160 
161 void initiateBoardCheck(int *boardCheck) {
162     for (int i = 0; i < SIZE * SIZE; i++) {
163         boardCheck[i] = 0;
164     }
165     boardCheck[2*SIZE + 2] = 1;
166 }
167 
168 void initiateCheckValue(int *checkValue) {
169     for (int i = 0; i < MAXVALUE; i++) {
170         checkValue[i] = i+1;
171     }
172 }
173 
174 void printValue(int *checkValue) {
175     for (int i = 0; i < MAXVALUE; i++) {
176         printf("%d \n", checkValue[i]);
177     }
178 }
179 
180 int startBingo() {
181     int *boardValue = (int *)malloc(SIZE*SIZE * sizeof(int));
182     makeBoard(boardValue);
183 
184     int *boardCheck = (int *)malloc(SIZE*SIZE * sizeof(int));
185     initiateBoardCheck(boardCheck);
186 
187 
188     int *checkValue = (int *)malloc(MAXVALUE * sizeof(int));
189     initiateCheckValue(checkValue);
190     shuffle(checkValue, MAXVALUE);
191 
192     int count = 0;
193 
194     while (!checkCol(boardCheck) && !checkRow(boardCheck) && !checkDia(boardCheck
    )) {
195         count ++ ;
196         int dif = searchBoard(boardValue, checkValue[count]);
197         updateCheck(boardCheck, dif);
198     }
199 
```

```
200     free(boardValue);
201     free(boardCheck);
202     free(checkValue);
203 
204     return count;
205 }
206 
207 void printResult(int *result) {
208     for (int i = 0;  i < 500; i++) {
209         if(result[i] != 0) printf("%d, %d\n", i, result[i]);
210         else continue;
211     }
212 }
213 
214 void probUnderX(int iteration, int *result, int x)
215 {
216     int total = 0;
217     for (int i = 1; i <= x; i++) {
218         total += result[i];
219     }
220 
221     printf("Probability under %d = %lf\n", x, (double)(total)/(double)(iteration)
    );
222 }
223 
224 void probWhenX(int iteration, int *result, int x)
225 {
226     printf("Probability when %d = %lf\n", x, (double)(result[x])/(double)(
    iteration));
227 }
228 
229 int getMax(int *result) {
230     int max = 0;
231     int index = 0;
232     for (int i = 0; i < MAXVALUE; i++) {
233         if (result[i] != 0) index = i;
234         else continue;
235     }
236     return index;
237 }
238 
239 void resetResult(int *result) {
240     for (int i = 0; i < MAXVALUE; i++) {
241         result[i] = 0;
242     }
243 }
244 
245 int main(int argc, char **argv) {
246     srand(time(NULL));
247 
248     int iteration;
249     if (argc == 2) {
250         iteration = atoi(argv[1]);
251     } else {
252         printf("Iteration value set to 5000\n");
253         iteration = 5000;
254     }
255 
256 
257     const int repetitionCount = 3;
258     int *result = (int *)malloc(MAXVALUE * repetitionCount * sizeof(int));
259     int *flag = (int *)malloc(repetitionCount * sizeof(int));
260 
261 
262     FILE *p = fopen("bingo.csv", "w");
263     for (int i = 0; i < repetitionCount; i++) {
264         for (int j = 0; j < iteration; j++) {
```

```
265             int r = startBingo();
266             result[r + i*MAXVALUE]++;
267             if (flag[i] == 0 && r == 4) {
268                 flag[i] = j;
269             }
270         }
271     }
272 
273     printf("Winning Round, Win Count, Win Count,\n");
274     fprintf(p, "Winning Round, Win Count, Win Count,\n");
275     for (int i = 0; i < MAXVALUE; i++) {
276         printf("%d, ",i);
277         fprintf(p, "%d, ",i);
278         for (int j = 0; j < repetitionCount; j++) {
279             printf("%d, %d, ", flag[j], result[i + j*MAXVALUE]);
280             fprintf(p, "%d, %d, ", flag[j], result[i + j*MAXVALUE]);
281         }
282         printf("\n");
283         fprintf(p, "\n");
284     }
285 
286     free(flag);
287     free(result);
288     fclose(p);
289 
290     return 0;
291 }
```

6.2 ビンゴのシミュレーションの流れ

ビンゴをシミュレートする手順は以下の通りになる．

1. ビンゴカードを生成する．
2. 乱数で番号をひき，ビンゴカードをチェックする．
3. 横・館・斜めのいずれか 1 列にある 5 マスが揃うかどうかをチェックする．
4. 揃っていない場合，２番に帰る．

6.3 ビンゴのシミュレーション結果

反復回数が 10^8 で 3 回のシミュレーションを行った．シミュレーション結果は以下のようになる．

6.3.1　１回目のシミュレーション結果

１回目のシミュレーション結果をグラフで表すと以下のグラフになる．

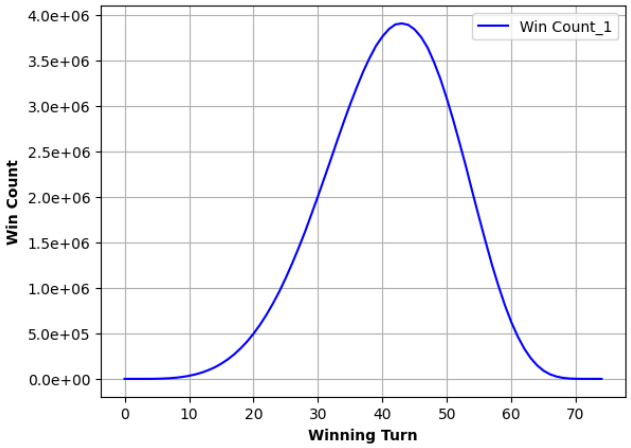


図 10:　１回目の当たる回数の分布

6.3.2　２回目のシミュレーション結果

２回目のシミュレーション結果をグラフで表すと以下のグラフになる．

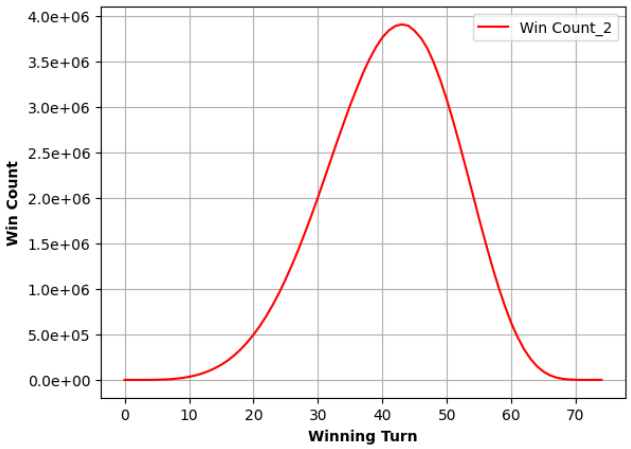


図 11:　２回目の当たる回数の分布

6.3.3　３回目のシミュレーション結果

３回目のシミュレーション結果をグラフで表すと以下のグラフになる．

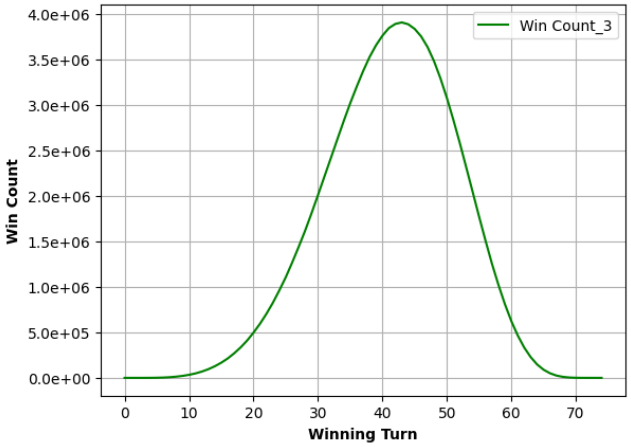


図 12:　３回目の当たる回数の分布

6.3.4　４回で当たる確率

各シミュレーションの結果から，４回で当たる確率は以下になる．

- シミュレーション１回目：0.000345%
- シミュレーション２回目：0.000318%
- シミュレーション３回目：0.000339%

6.3.5　７回で当たる確率

各シミュレーションの結果から，７回で当たる確率は以下になる．

- シミュレーション１回目：0.012380%
- シミュレーション２回目：0.012543%
- シミュレーション３回目：0.012326%

6.3.6　最も遅いゲームの回数

各シミュレーションの結果から，最も遅いゲームの回数は以下になる．

- シミュレーション１回目：71 回
- シミュレーション２回目：71 回
- シミュレーション３回目：71 回

6.3.7　最も遅いゲームが起こる確率

各シミュレーションの結果から，最も遅いゲームが起こる確率は以下になる．

- シミュレーション１回目：0.000133%
- シミュレーション２回目：0.000131%
- シミュレーション３回目：0.000126%

6.4　ビンゴのシミュレーションの考察

シミュレーションの結果から，以下のような結論が得られた．

- 図 (10)，図 (11)，図 (12) から見ると，ビンゴになるまでの回数の分布は似ている．
- ４回以内で当たる確率は 0.0003% である．
- ７回以内で当たる確率は 0.012% である．
- 最も遅いゲームの回数は 71 回である．
- 最も遅いゲームが起こる確率は 0.00013% である．

7　発表感想

3 人の発表者のうち，制限時間をオーバーしたのは 1 人だけだった．パワーポイントのスライドもわかりやすいし，面白い画像を使っている人もいて，より興味深い．このテーマが難しいから、事前準備の時間が短くなったのかもしれない．

質問者については、全員が自分の仕事をきちんとこなしていた．以前より良くなったと言える．今回はみんなもっと準備していたようで，質疑応答も質問者のおかげでスムーズに進んだ．

今回の発表は前回より少し良くなったが，まだ改善すべき点はたくさんある．だからこそ，最後の発表がより良いの発表になることを期待している．

参考文献

[1] https://www.scaler.com/topics/random-number-generator-in-c/（参照 2023-07-04）

[2] https://en.wikipedia.org/wiki/Probability（参照 2023-07-04）