第七次作业 蒙特卡洛方法

蔡淼 15346001

问题一：

令 D=5，a=2.0， Δ=0.1a，取 Markov 链[1,10000]和[10000,20000]之间的连续点做统计直方图，看是否服从 p(x)分布.

解决方案：

由于使用 C 语言不方便绘制直方图，故使用 C 语言进行数据的获取，利用 matlab 进行直方图的绘制以及结果的展示。

以下结果均以 5 维中的一维作为示例，余下 4 维原理相同，不予赘述。

（1）源代码

C 程序代码展示如下：

• 主程序 HW7_1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
double P_x(double x);
double GetAlpha(double x_1,double x_2);

int main()
{
    srand(time(0));

    double x0,a,temp,deltaX,temp2;
    int i,j,N;
    float x[20000];
    a = 2.0;
    N = 20000;
    x0 = 2*(rand()/(double)RAND_MAX);
    x[0] = x0;
```

```
19.
20.     for (i=0;i<(N-2);i++)
21.     {
22.     temp = rand()/(double)RAND_MAX;
23.     deltaX = 0.2*(temp - 0.5)*2;
24.     x[i+1] = x[i] + deltaX;
25.     if (x[i+1] > a)
26.     {
27.         x[i+1] = x[i+1] - a;
28.     }
29.     if (x[i+1] < 0)
30.     {
31.         x[i+1] = x[i+1] + a;
32.     }
33.     temp2 = rand()/(double)RAND_MAX;
34.     if (GetAlpha(x[i],x[i+1])<temp2)
35.     {
36.         x[i+1] = x[i];
37.     }
38.     }
39.
40.     //printf("The 10000th number of the sequence is %f \n"
   ,x[10000]);
41.     return 0;
42. }
43.
44. //计算 p(x)的函数
45. double P_x(double x)
46. {
47.     double y;
48.     double deno;
49.     deno = 1-exp(-2);
50.     //deno = pow(deno,5);
```

```
51.     y = exp(-1*x)/deno;
52.
53.     return y;
54. }
55.
56. //返回游动判定结果的函数
57. double GetAlpha(double x_1,double x_2)
58. {
59.     double p1,p2,Omega;
60.     p1 = P_x(x_1);
61.     p2 = P_x(x_2);
62.     Omega = p2/p1;
63.
64.     return Omega;
65. }
```

- 用于绘制直方图的 matlab 程序

```
1.  PlotHist(x);
2.  hold on
3.  z = 0:0.002:2;
4.  plot(z,P_x(z));
5.  hold off
```

- matlab 程序中用到的函数定义

```
1.  //函数 PlotHist
2.  //绘制统计直方图
3.  function PlotHist(Mat)
4.  N = 1000;
5.  MaxIntValue = max(Mat);
6.  Count = zeros(1,N);
7.  step = MaxIntValue/1000;
8.  for i =0:1:(N-1)
```

```
    temp1 = size(Mat(Mat<i*step),2);
    temp2 = size(Mat(Mat<(i+1)*step),2);
    Count(i+1) = temp2 - temp1;
end
axis = 0:step:(MaxIntValue-step);
Count = (Count./max(Count))*P_x(0);
bar(axis,Count);
```

（2）结果展示

利用上述代码，可以画出生成的 20000 个点的马尔可夫链的统计分布图，此处绘制直方图的横坐标时将范围[0,a]（a=2）等分成了 1000 块，来统计分布在每一块内的点的数量，并将其进行适当的归一化，以与 p(x)进行直观的比较，其结果如图 1 和图 2 所示。

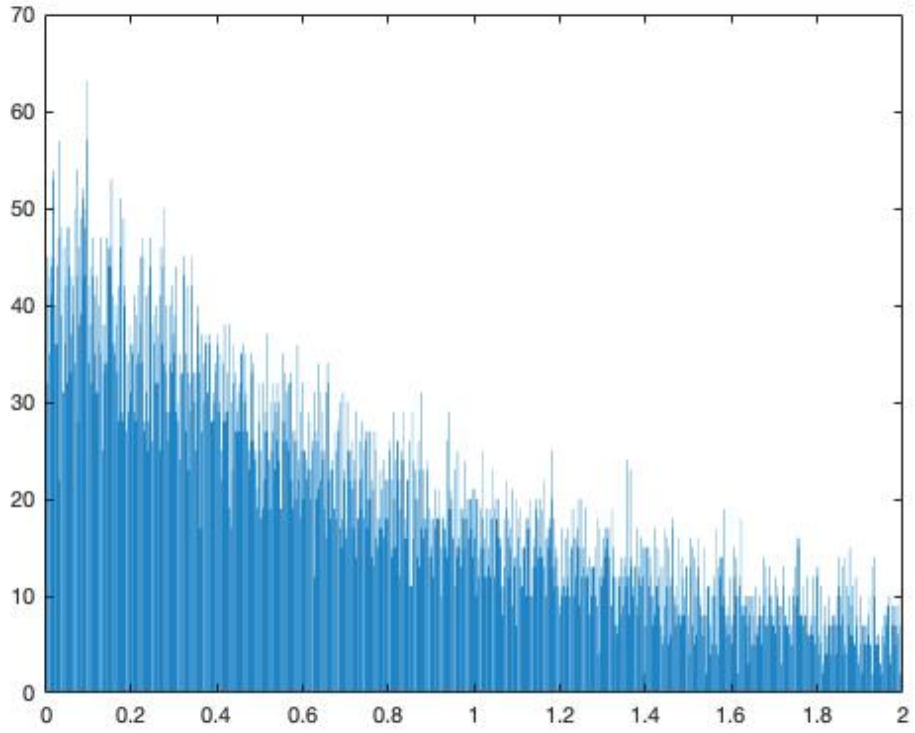


图 1 Markov 链上随机点的分布情况

将其与 p(x)进行对比，可以得到图 2：

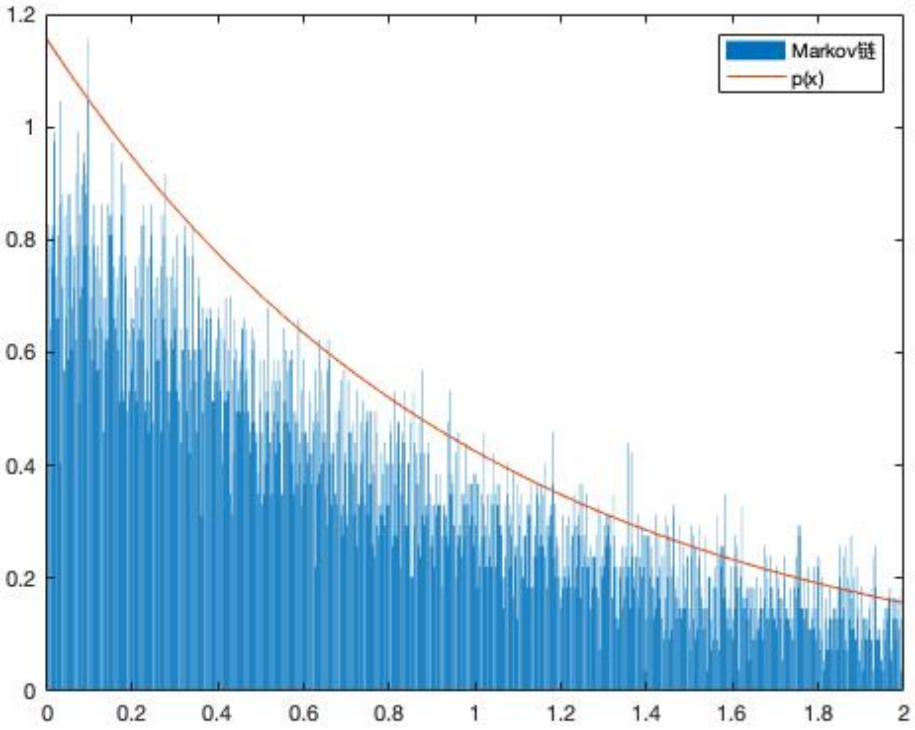


图 2 马尔可夫链上点分布与 p(x)对比示意图

可以看出，生成的马尔可夫链上点的分布与函数 p(x)的趋势是一致的。接下来，我们取[1,10000]和[10000,20000]的连续点分别绘制直方图。

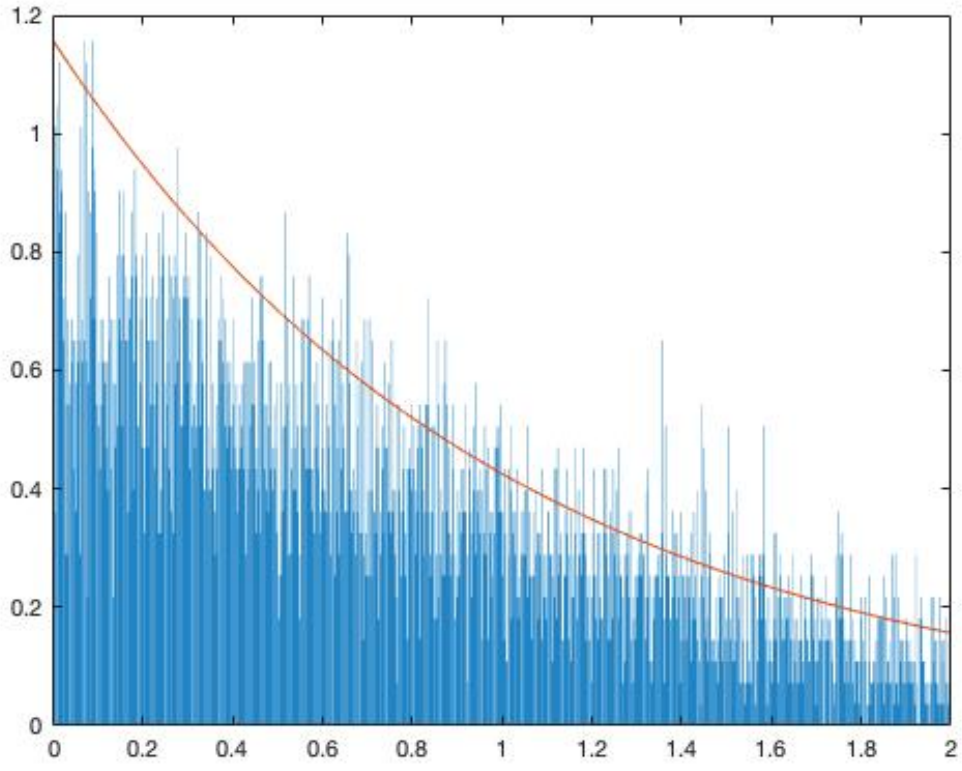


图 3 [1,10000]上的点的分布

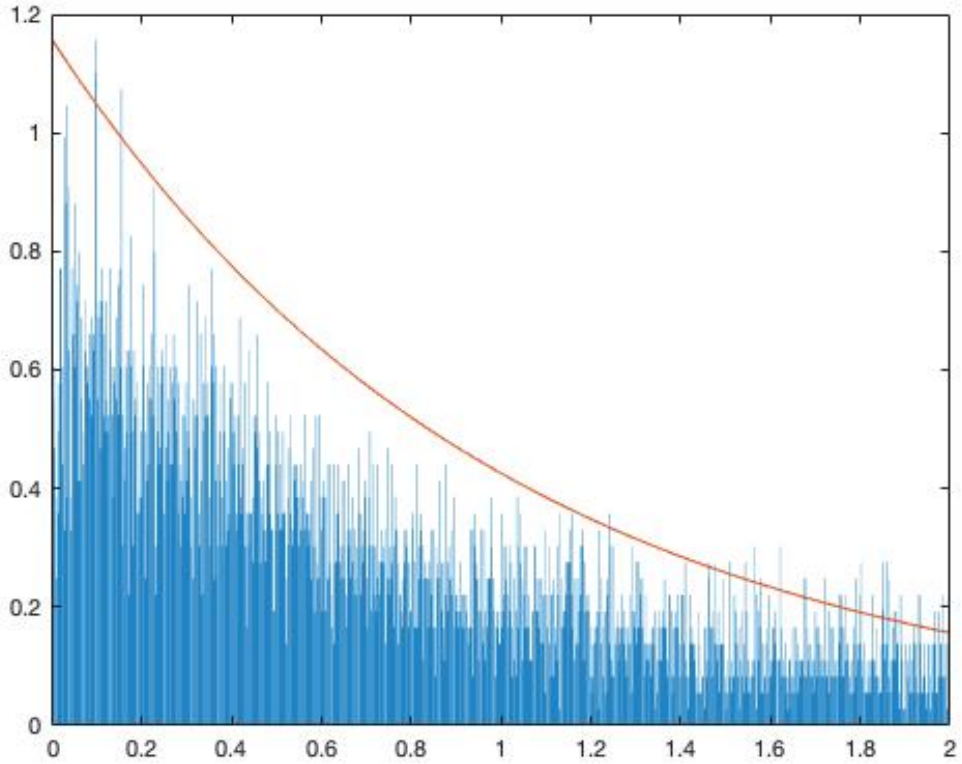


图 4 [10000,20000]上的点的分布

从图 3 和图 4 可以看出，两者基本都服从 p(x)分布，而且[10000,20000]的点分布与 p(x)符合的更好一些，证明了 Markov 链在后期趋向平稳的事实。

问题二：

令 D=5，a=2.0， Δ=0.1a，N_{equi}=10000，N_0=50，m=10000，分别用 L=200，400 做 I_m 的统计直方图，并标出 I_{exact} 位置。

解决方案：

以下结果均以 5 维中的一维作为示例，余下 4 维原理相同，不予赘述。

（1）解决问题所用的 C 程序代码如下所示

主程序 HW7_2.c：

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "MonteCarlo.c"
#define L 200
#define m 10000

int main()
{
    //设置随机种子
    srand(time(0));

    //声明变量
    double a;
    double *Markov_temp;
    double Markov[m+1];
    int i,j,N,N0,step;
    double Intm[L];

    //参数初始化
    a = 2.0;
    N = 510000;
    N0 = 10000;
    step = 50;
```

```
26.
27.     for (i=0;i<L;i++)
28.     {
29.     Markov_temp = GetMarkov(a,N,Markov_temp);
30.     for (j=0;j<m;j++)
31.     {
32.         Markov[j] = Markov_temp[N0+j*step];
33.     }
34.     Intm[i] = ((m*1.0 + ((sum(Markov,m+1))/2.0))*(1.0-exp(-1.0*a)))/m;
35.     printf("L now is %d \n",i);
36.     }
37.
38.     for (i = 0;i<L;i++)
39.     {
40.     printf("The %d th number of Intm is %.8f \n",i+1,Intm[i]);
41.     }
42.     return 0;
43. }
```

其中用到的头文件 MonteCarlo.c 代码如下：

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <time.h>
4.  #include <math.h>
5.
6.  //计算 p(x)的函数
7.  double P_x(double x)
8.  {
9.      double y;
10.     double deno;
11.     deno = 1-exp(-2);
12.     //deno = pow(deno,5);
```

```
13.     y = exp(-1*x)/deno;
14.
15.     return y;
16. }
17.
18. //返回游动判定结果的函数
19. double GetAlpha(double x_1,double x_2)
20. {
21.     double p1,p2,Omega;
22.     p1 = P_x(x_1);
23.     p2 = P_x(x_2);
24.     Omega = p2/p1;
25.
26.     return Omega;
27. }
28.
29. //求蒙特卡洛采样点的函数
30. //返回一维数组
31. double* GetMarkov(double a,int N,double y[])
32. {
33.     double x[N];
34.     double x0,temp,deltaX,temp2;
35.     int i,j;
36.
37.     //srand(time(0));
38.
39.     x0 = 2 * (rand()/(double)RAND_MAX);
40.     x[0] = x0;
41.     for (i = 0;i<(N-2);i++)
42.     {
43.     temp = rand()/(double)RAND_MAX;
44.     deltaX = 0.2*(temp - 0.5)*2;
45.     x[i+1] = x[i] + deltaX;
```

```
46.     if (x[i+1]>a)
47.     {
48.         x[i+1] = x[i+1] - a;
49.     }
50.     else if (x[i+1]<0)
51.     {
52.         x[i+1] = x[i+1] + a;
53.     }
54.     temp2 = rand()/(double)RAND_MAX;
55.     if (GetAlpha(x[i],x[i+1])<temp2)
56.     {
57.         x[i+1] = x[i];
58.     }
59.     }
60.     y = x;
61.     return y;
62. }
63.
64. // 数组求和函数
65. // 对数组进行求和，返回求和值
66. double sum(double *Mat,int length)
67. {
68.     double r;
69.     int i;
70.
71.     r = 0.0;
72.     for (i=0;i<length;i++)
73.     {
74.     r = r + Mat[i];
75.     }
76.     return r;
77. }
```

用于绘制直方图的 matlab 代码如下：

```
1.  PlotHist(Intm,10)
```

通过改变主程序 HW7_2.c 中 L 的值，就可以得到题目描述中不同 L 下的 I_m 分布情况并获取统计直方图如图 5&图 6 所示：

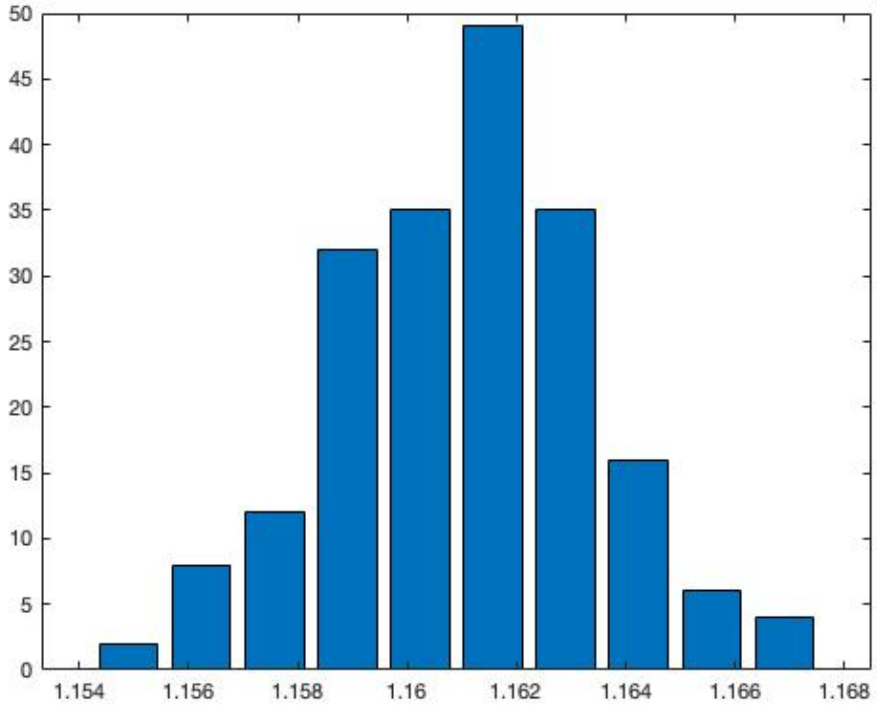


图 5 L = 200 的情况下 Im 的分布

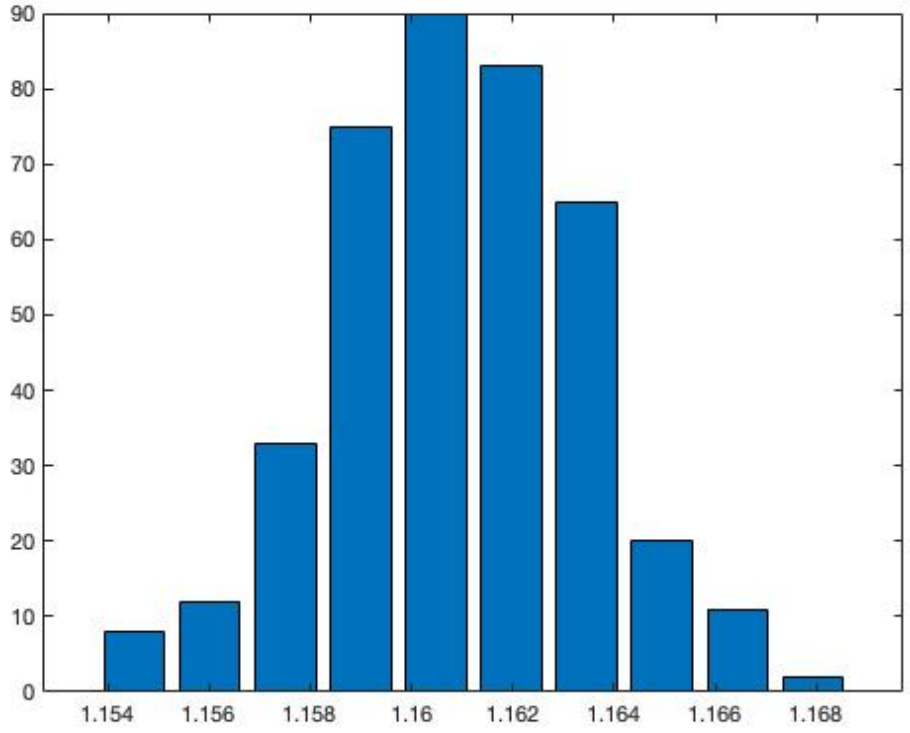


图 6 L=400 时 Im 的分布

而 I_{exact} 的值为：

$$I_{exact}=\frac{3}{2}-\left(\frac{3}{2}+1\right)e^{-2}=1.1617$$

与图 5&图 6 的中心峰值进行比较可知，分布数目最集中的地区正是 1.1617 及其附近的范围，故可以认为取样合理。

问题三：

令 a=2.0，Δ=0.1a，N_{equi}=10000，N_0=50，L=200，改变 m 的值，m=10^4~10^5，做 D=5 和 D=8 的 σ_m 和 m 的关系曲线，验证蒙特卡罗标准差~1/sqrt(m)行为。给出蒙特卡洛积分的结果，比较计算误差 E_r=|I_{MI}-I_{exact}|与 σ_m/sqrt(L)的关系。

（1）代码展示

A. 验证蒙特卡洛标准差~1/sqrt(m)行为

• 主程序 HW7_3_1.c

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

int main()
{
    int m,N0,step,D,L;
    double a,deltaX,sigma;

    srand(time(0));

    a = 2.0;
    deltaX = 0.1*a;
    N0 = 10000;
    step = 50;
    L = 10;
    D = 5;
    m = 30000;

    sigma = GetSigma(a,deltaX,N0,step,m,D,L);
    printf("sigma is %f \n",sigma);

```

```
25.     return 0;
26. }
```

• 头文件 MonteCarlo.c 见附录。

（2）结果展示

a）D = 5 的情况下，m 从 10000，以 5000 的步长变化至 100000 的过程中，σ_m 随着 m 变化的情况。

m	σ_m	m	σ_m
10000	0.009636	60000	0.004219
15000	0.008033	65000	0.004144
20000	0.00727	70000	0.003875
25000	0.006681	75000	0.003781
30000	0.005897	80000	0.003699
35000	0.005243	85000	0.003478
40000	0.005135	90000	0.00351
45000	0.00553	95000	0.003161
50000	0.005045	100000	0.003393
55000	0.004485	105000	0.003019

表 1 数据记录表

将其绘制成图可得：

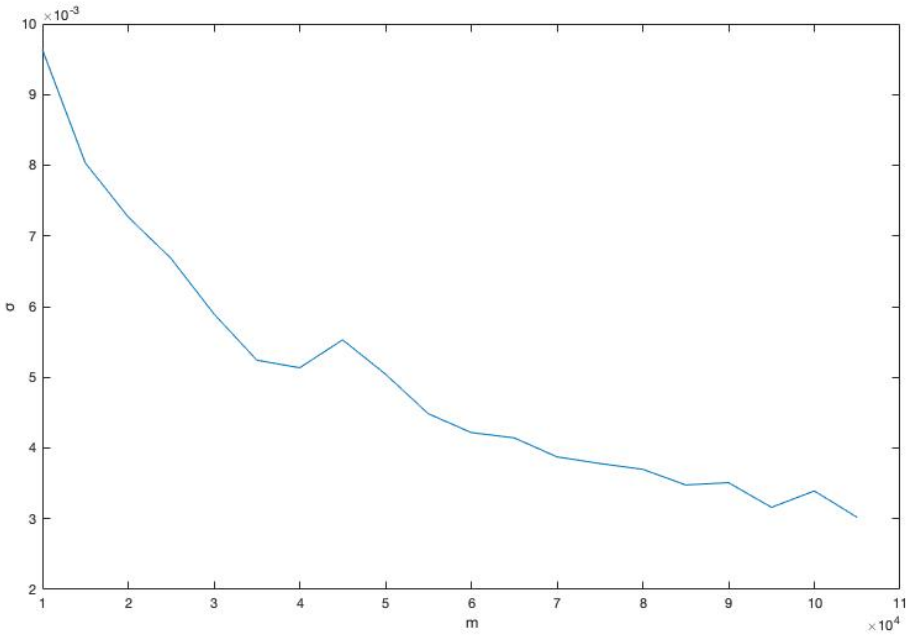


图 7

从图 7 可以看出，随着 m 的增加，σ 持续下降，说明 m 越大，结果越稳定。将 1/sqrt(m)的趋势图与图 7 中曲线进行对比，可得：

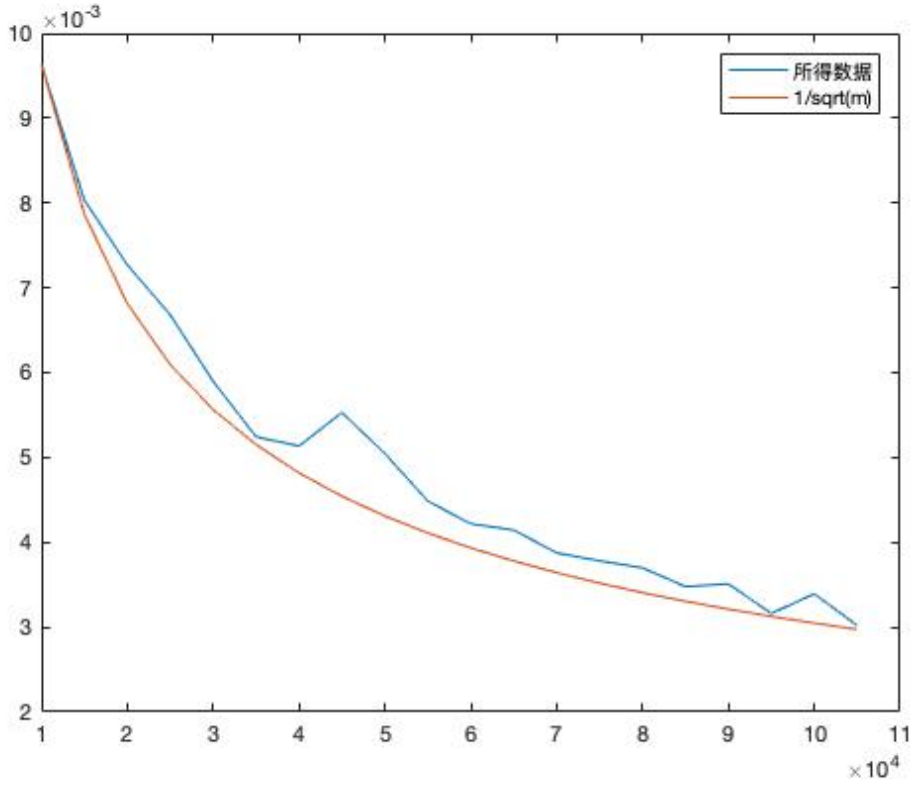


图 8

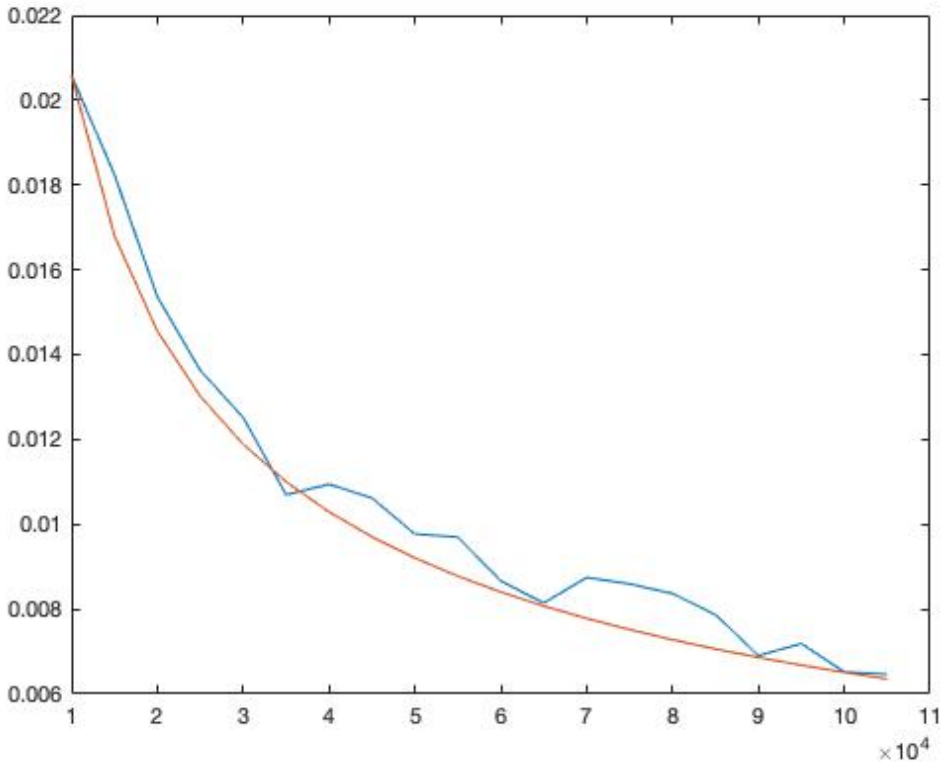
故可以认为，在 D=5 的情况下，成功验证了蒙特卡罗标准差~1/sqrt(m)的行为。

b）D = 8 的情况下，m 从 10000，以 5000 的步长变化至 100000 的过程中，σ_m 随着 m 变化的情况。

m	σ_m	m	σ_m
10000	0.020586	60000	0.008663
15000	0.018251	65000	0.008145
20000	0.015365	70000	0.008745
25000	0.013630	75000	0.008595
30000	0.012517	80000	0.008366
35000	0.010695	85000	0.007870
40000	0.010945	90000	0.006906
45000	0.010620	95000	0.007186
50000	0.009770	100000	0.006519
55000	0.009699	105000	0.006465

表 2 数据记录表

与 a）中类似，可以绘制得到下图：



故可以认为，在 D=8 的情况下，成功验证了蒙特卡罗标准差~1/sqrt(m)的行为。

B. 比较计算误差 $E_r=|I_{Ml}-I_{exact}|$与 $\sigma_m/sqrt(L)$的关系

以下计算以 D=5 为例。

（1）代码展示

主程序 HW7_3_2.c

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

int main()
{
    int i,m,N0,step,D,L,n,temp;
    double a,deltaX,sigma,I_exact,error;
    double* answer;
    double* axis;

    //设置随机种子
    srand(time(0));

    I_exact = pow(1.1617,5);
    a = 2.0;
    deltaX = 0.1*a;
    N0 = 10000;
    step = 50;
    D = 5;
    m = 10000;
    n = 200;
    error = 0.0;
    sigma = 0.0;

    //给 answer 创建储存空间
```

```
29.     answer = (double*)malloc(sizeof(double) * n);
30.     axis = (double*)malloc(sizeof(double) * n);
31.
32.     for (i = 0;i<n;i++)
33.     {
34.         L = i+1;
35.         sigma = GetSigma(a,deltaX,N0,step,m,D,L);
36.         axis[i] = sigma/sqrt((double)L);
37.         error = I_exact - (GetIntm(a,deltaX,N0,step,m,D,L));
38.         error = fabs(error);
39.         answer[i] = error;
40.         printf("The i now is %d \n",i);
41.         fflush(stdout);
42.     }
43.
44.
45.     FILE *fpWrite = fopen("HW7_3_2.txt","w");
46.     if(fpWrite==NULL)
47.     {
48.         return 0;
49.     }
50.     for(i=0;i<n;i++)
51.     {
52.         fprintf(fpWrite,"%8f ;%8f \n",axis[i],answer[i]);
53.     }
54.     //释放内存
55.     free(answer);
56.     free(axis);
57.
58.     return 0;
59. }
```

（2）结果展示

由主程序 HW7_3_2.c 可以看出，运行程序所得到的结果是 L 从 1 变化到 200，对应的 $E_r=|I_{Ml}-I_{exact}|$ 与 $\sigma_m/\mathrm{sqrt}(L)$ 的值，所得数据可以绘制成图，如下图所示：

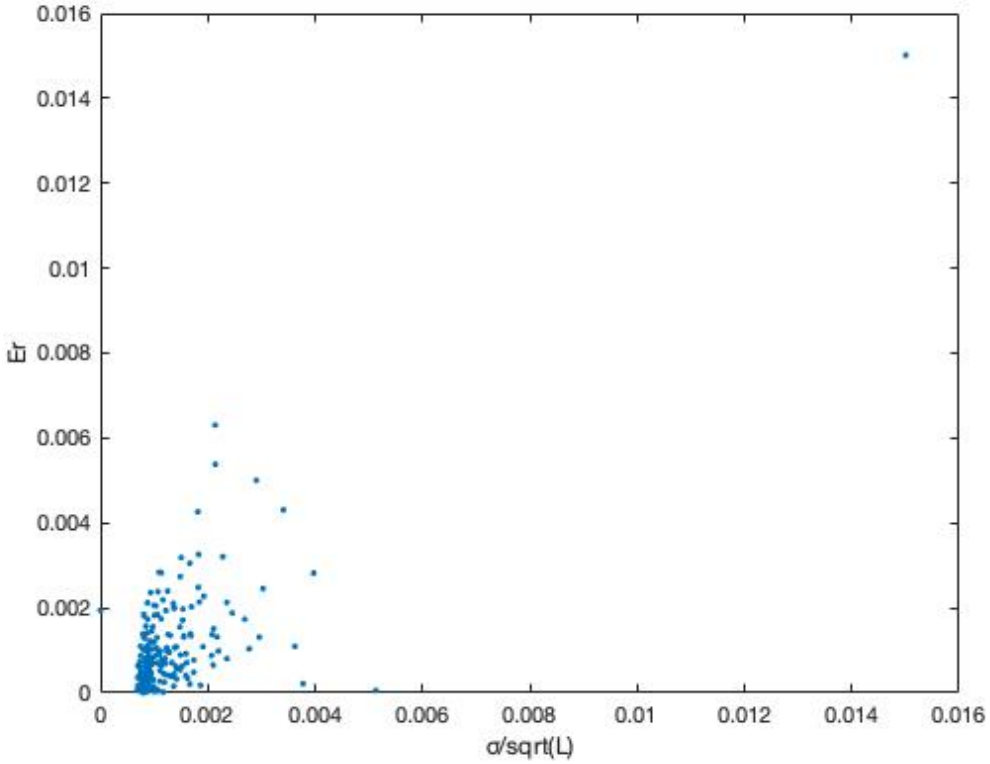


图 9

问题四：

考察 N_{equi} 选取对结果的影响。令 D=5，a=2.0，Δ=0.1a，N_0=50，m=10000，L=200，选取 N_{equi}=2000，4000，8000，16000，看 σ_m 的变化。

（1）代码展示

• 主程序 HW7_4.c

```
1. //
2. //  HW7_4.c
3. //
4. //
5. //  Created by 三水 on 2018/12/4.
6. //
7.
8. #include <stdio.h>
9. #include <stdlib.h>
10. #include <time.h>
11. #include <math.h>
12. #include "MonteCarlo.c"
13.
14. int main()
15. {
16.     int i,m,N0,step,D,L,n,temp;
17.     double a,deltaX,sigma;
18.     double answer[4];
19.
20.     srand(time(0));
21.
22.     a = 2.0;
23.     deltaX = 0.1*a;
24.     step = 50;
25.     L = 200;
26.     D = 5;
```

```
27.     m = 20000;
28.     n = 200;
29.     temp = 0;
30.     //N0 = 10000;
31.
32.     N0 = 2000;
33.     answer[0] = GetSigma(a,deltaX,N0,step,m,D,L);
34.     N0 = 4000;
35.     answer[1] = GetSigma(a,deltaX,N0,step,m,D,L);
36.     N0 = 8000;
37.     answer[2] = GetSigma(a,deltaX,N0,step,m,D,L);
38.     N0 = 16000;
39.     answer[3] = GetSigma(a,deltaX,N0,step,m,D,L);
40.
41.     for (i=0;i<4;i++)
42.     {
43.         printf("the %d th answer is %f \n",i,answer[i]);
44.     }
45.
46.
47.     return 0;
48. }
```

其中用到的头文件 MonteCarlo.c 与问题三中给出的一致。

（2）结果展示

运行主程序 HW7_4.c 得到的结果如下表所示：

N_{equi}	2000	4000	8000	16000
σ_m	0.011318	0.011617	0.010274	0.011257

表 3 运行结果

从表 1 可以看出，N_{equi} 的变化对 σ_m 的影响并不明显，说明 Markov 链至少在 2000 步以内就已经基本达到了平衡，故改变 N_{equi} 的选取对实验的结果影响不大。

问题五：

考察 N_0 选取对结果的影响。令 D=5，a=2.0， Δ =0.1a，m=10000，L=1，N_{equi}=10000，用平衡后的点，计算协方差 $Cov(x^{(l)},x^{(l+N0+1)})$作为 N_0 的函数，由此确定 N_0 的最佳取值。

（1）代码展示

主程序 HW7_5.c

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

int main()
{
    //设置随机种子
    srand(time(0));

    //声明变量
    double a;
    double* Cov;
    int m,i,j,N,N0,step,number,step_Max;
    int StartOfM2,EndOfM1;
    double* Markov;
    double* Markov1;
    double* Markov2;

    //初始化参量
    a = 2.0;
    N0 = 10000;
    m = 10000;
    number = 50000;
```

```
26.     step_Max = 1000;
27.     N = N0 + 1000000;
28.     //给一维数组 Cov 设置存储空间
29.     Cov = (double *)malloc(sizeof(double) * step_Max);
30.
31.     for (step=2;step<step_Max;step++)
32.     {
33.         StartOfM2 = N0 + step +1;
34.         EndOfM1 = N - step - 1;
35.
36.         //给一维数组 Markov 设置存储空间
37.         Markov = (double *)malloc(sizeof(double) * N);
38.         Markov1 = (double *)malloc(sizeof(double) * EndOfM1);
39.         Markov2 = (double *)malloc(sizeof(double) * EndOfM1);
40.
41.         //求长度为 N 的 Markov 链
42.         Markov = GetMarkov(a,N,Markov);
43.
44.         //求 x_l 与 x_(l+N0+1)
45.         for (i=N0;i<EndOfM1;i++)
46.         {
47.             Markov1[i-N0] = Markov[i];
48.         }
49.         for (j=StartOfM2;j<N;j++)
50.         {
51.             Markov2[j-StartOfM2] = Markov[j];
52.         }
53.
54.         //求协方差
55.         Cov[step-2] = GetCov(Markov1,Markov2,number);
56.         free(Markov);
57.         free(Markov1);
58.         free(Markov2);
```

```
59.         printf("The step now is %d \n",step);
60.         fflush(stdout);
61.     }
62.     //将得到的数据写入 txt 文件
63.     FILE *fpWrite = fopen("HW7_5.txt","w");
64.     if(fpWrite==NULL)
65.     {
66.         return 0;
67.     }
68.     for(i=2;i<step_Max;i++)
69.     {
70.         fprintf(fpWrite,"%8f \n",Cov[i-2]);
71.     }
72.
73.     return 0;
74.
75. }
```

其中用到的新函数 GetCov()为，其作用为计算两个数组之间的协方差：

```
1.  //求两个一维数组前 m 位的协方差
2.  double GetCov(double Markov1[],double Markov2[],int m)
3.  {
4.      double temp1[m];
5.      int i,j;
6.      double E1,E2,answer;
7.
8.      for (i=0;i<m;i++)
9.      {
10.         temp1[i] = Markov1[i]*Markov2[i];
11.     }
12.     E1 = sum(temp1,m)/m;
13.     E2 = (sum(Markov1,m)/m)*(sum(Markov2,m)/m);
```

```
14.
15.     answer = E1 - E2;
16.
17.     return answer;
18. }
```

从上述代码可以看出，主程序 HW7_5.c 的作用是以一维为例，求得 N_0（在代码中为 step 变量）从 2 变化到 1000 中，x^l 与 x^{l+N_0+1} 的协方差。

（2）结果展示

运行 HW7_5.c 后得到数据 HW7_5.txt，将其用 matlab 进行绘图展示可得图 7：

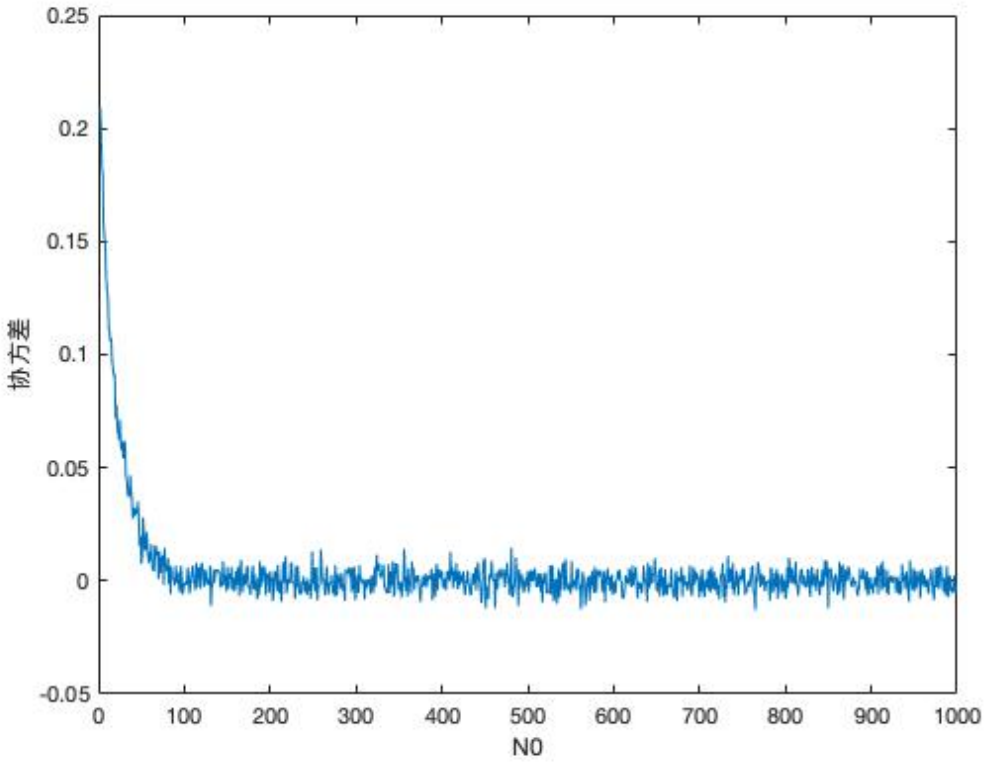


图 10 协方差大小与 N_0 的关系图

由图 7 可以看出，在 N_0 属于 1-100 的范围时，随着 N_0 的增加，协方差迅速的降低，在 N_0 达到 100 左右之后，N_0 继续增加，协方差在 0 的附近进行振荡，故可以认为 N_0 大于 100 的值都可以满足在 Markov 链上取点的要求。

问题六：

考察Δ选取对结果的影响。令D=5，a=2.0，N_0=50，N_{equi}=10000，m=10000，L=200，计算接受率a，做两个图：

a）σ_m与Δ的关系图

b）接受率a与Δ的关系图

a）σ_m与Δ的关系

（1）代码展示

主程序 HW7_6_1.c

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

int main()
{
    int i,j,m,N0,step,D,L,n;
    double a,Delta,sigma;
    double answer[40];

    srand(time(0));

    a = 2.0;
    N0 = 10000;
    step = 50;
    L = 200;
    D = 5;
    m = 10000;
    n = 20;

    for (i=0;i<n;i++)
```

```
24.     {
25.         Delta = (0.04 + 0.04*i)*a;
26.         answer[i] = GetSigma(a,Delta,N0,step,m,D,L);
27.         printf("now is the %d th circle \n",i);
28.         fflush(stdout);
29.     }
30.     //将得到的数据写入 txt 文件
31.     FILE *fpWrite = fopen("HW7_6_1.txt","w");
32.     if(fpWrite==NULL)
33.     {
34.         return 0;
35.     }
36.     for(i=0;i<n;i++)
37.     {
38.         fprintf(fpWrite,"%8f \n",answer[i]);
39.     }
40.
41.     return 0;
42. }
```

主程序 HW7_6_1.c 的作用是令 Δ 从 0.04a 开始，以 0.04a 为步长，逐步增加到 0.8a，并记录下每一个 Δ 对应的 σ_m，并将其写入 HW7_6_1.txt 文件中。

（2）结果展示

Δ	σm	Δ	σm	Δ	σm
0.01a	0.074798	0.11a	0.010764	0.21a	0.010582
0.02a	0.041177	0.12a	0.008819	0.22a	0.010107
0.03a	0.027102	0.13a	0.010709	0.23a	0.00968
0.04a	0.017237	0.14a	0.009458	0.24a	0.009292
0.05a	0.014654	0.15a	0.009301	0.25a	0.009756
0.06a	0.01407	0.16a	0.009796	0.26a	0.009784
0.07a	0.012877	0.17a	0.009713	0.27a	0.009923
0.08a	0.011041	0.18a	0.009821	0.28a	0.009857
0.09a	0.01071	0.19a	0.008963	0.29a	0.009386

0.1a	0.010061	0.2a	0.009846	0.3a	0.009551

表 4 数据记录表

从表 2 可以看出，随着 Δ 的增加， σ_m 开始下降，然后趋向于稳定，稳定值在 0.009 附近。

b）接受率 a 与 Δ 的关系

以一维为例。

（1）代码展示

主程序 HW7_6_2.c

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

int main()
{
    srand(time(0));

    double a,Delta;
    int i,N,N0,step,m,n,temp;
    double* answer;

    a = 2.0;
    N0 = 10000;
    step = 50;
    m = 10000;
    N = N0 + m*step;
    n = 625;
    //Delta = 0.4;
    answer = (double *)malloc(sizeof(double) * n);

```

```
    for (i=0;i<n;i++)
    {
        Delta = 0.04 + 0.04*i;
        temp = GetPassRate(a,N,Delta);
        answer[i] = (double)temp/(double)N;
        printf("the %d th answer is %8f \n",i+1,answer[i]);
        fflush(stdout);
    }
    //写入文件
    FILE *fpWrite = fopen("HW7_6_2.txt","w");
    if(fpWrite==NULL)
    {
        return 0;
    }
    for(i=0;i<n;i++)
    {
        fprintf(fpWrite,"%8f \n",answer[i]);
    }
    //释放内存
    free(answer);

    return 0;

}
```

其中用到的新函数 GetPassRate()为：

```
int GetPassRate(double a,int N,double Delta)
{
    double* x;
    double x0,temp,deltaX,temp2;
    int i,j,Fail_count,Success_count;

```

```
7.      Fail_count = 0;
8.      Success_count = 0;
9.      //srand(time(0));
10.     //给一维数组 x 创建储存空间
11.     x = (double *)malloc(sizeof(double) * N);
12.
13.     x0 = 2 * (rand()/(double)RAND_MAX);
14.     x[0] = x0;
15.     for (i = 0;i<(N-2);i++)
16.     {
17.         temp = rand()/(double)RAND_MAX;
18.         deltaX = Delta*(temp - 0.5)*2;
19.         x[i+1] = x[i] + deltaX;
20.         while (x[i+1]>a)
21.         {
22.             x[i+1] = x[i+1] - a;
23.         }
24.         while (x[i+1]<0)
25.         {
26.             x[i+1] = x[i+1] + a;
27.         }
28.         temp2 = rand()/(double)RAND_MAX;
29.         if (GetAlpha(x[i],x[i+1])<temp2)
30.         {
31.             x[i+1] = x[i];
32.             Fail_count = Fail_count + 1;
33.         }
34.     }
35.     free(x);
36.     Success_count = N - Fail_count;
37.     return Success_count;
38. }
```

从上述代码可以看出，主程序 HW7_6_2.c 的作用是求得Δ从 0.04 开始，以 0.04 的步长一直增长到 25 的过程中，通过率的值，并写入 HW7_6_2.txt 中。

（2）结果展示

将（1）中获得的数据进行关系图的绘制，可得下图：

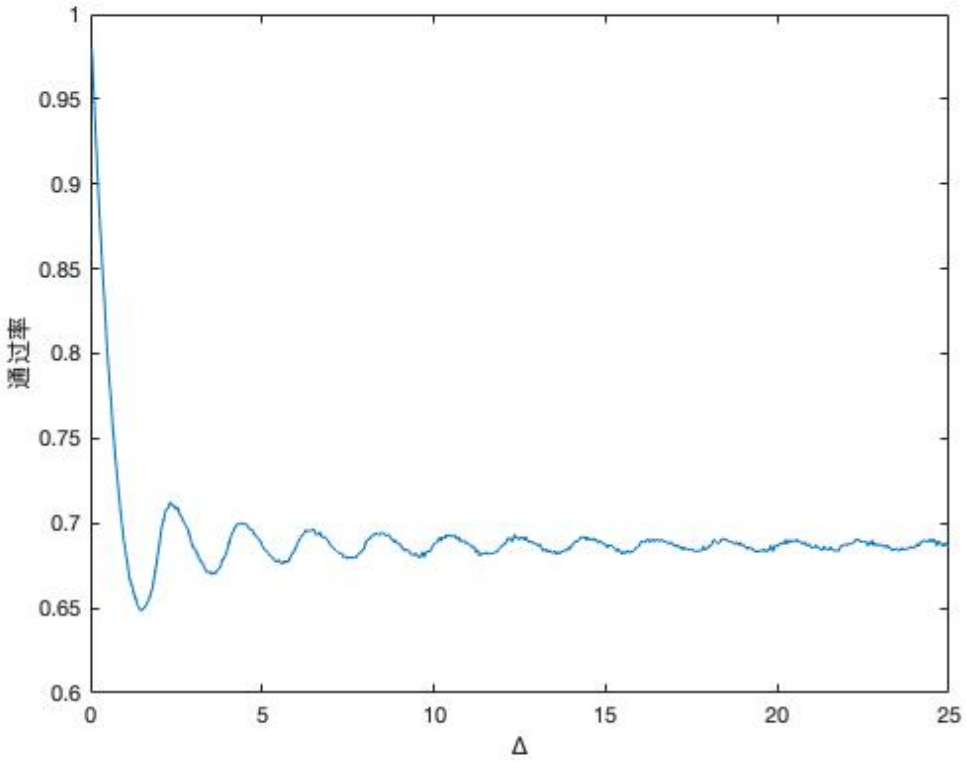


图 11 通过率与Δ的值之间关系图

可见，随着Δ的增长，通过率在开始时迅速下降，然后开始一边振荡一边收敛至某一个特定的值。

附录

- MonteCarlo.c

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


//计算 p(x)的函数
double P_x(double x)
{
    double y;
    double deno;
    deno = 1-exp(-2);
    //deno = pow(deno,5);
    y = exp(-1*x)/deno;

    return y;
}

//返回游动判定结果的函数
double GetAlpha(double x_1,double x_2)
{
    double p1,p2,Omega;
    p1 = P_x(x_1);
    p2 = P_x(x_2);
    Omega = p2/p1;

    return Omega;
}

//求蒙特卡洛采样点的函数
```

```
31. //返回一维数组
32. double* GetMarkov(double a,int N,double Delta,double y[])
33. {
34.     double* x;
35.     double x0,temp,deltaX,temp2;
36.     int i,j;
37. 
38.     //srand(time(0));
39.     x = (double *)malloc(sizeof(double) * N);
40. 
41.     x0 = 2 * (rand()/(double)RAND_MAX);
42.     x[0] = x0;
43.     for (i = 0;i<(N-2);i++)
44.     {
45.         temp = rand()/(double)RAND_MAX;
46.         deltaX = Delta*(temp - 0.5)*2;
47.         x[i+1] = x[i] + deltaX;
48.         while (x[i+1]>a)
49.         {
50.             x[i+1] = x[i+1] - a;
51.         }
52.         while (x[i+1]<0)
53.         {
54.             x[i+1] = x[i+1] + a;
55.         }
56.         temp2 = rand()/(double)RAND_MAX;
57.         if (GetAlpha(x[i],x[i+1])<temp2)
58.         {
59.             x[i+1] = x[i];
60.         }
61.     }
62.     y = x;
63.     free(x);
```

```
64.     return y;
65. }
66.
67. // 数组求和函数
68. // 对数组进行求和，返回求和值
69. double sum(double *Mat,int length)
70. {
71.     double r;
72.     int i;
73.
74.     r = 0.0;
75.     for (i=0;i<length;i++)
76.     {
77.     r = r + Mat[i];
78.     }
79.     return r;
80. }
81.
82. //求均方根的函数
83. double GetAQS(double* Intm,int L)
84. {
85.     int i;
86.     double QS,AQS;
87.
88.     QS = 0.0;
89.
90.     for (i=0;i<L;i++)
91.     {
92.     QS = QS + pow(Intm[i],2);
93.     }
94.     AQS = QS/L;
95.
96.     return AQS;
```

```
97. }
98.
99.
100. //求 sigma 的函数
101. //返回 sigma 的值
102. double GetSigma(double a,double Delta,int N0,int step,int m,int D,int L)
103. {
104.     double*  Markov_temp;
105.     double*  Markov_temp1;
106.     //double*  Markov_temp2;
107.     double**   Markov;
108.     long int i,j,k,l,s;
109.     int N;
110.     double Intm[L];
111.     double Intm_aver,AQS,sigma;
112.     double temp1,temp2;
113.
114.     //参数初始化
115.     N = N0 + m*step;
116.     temp1 = 0.0;
117.     temp2 = 0.0;
118.     //为一维数组 Markov_temp 分配空间
119.     //Markov_temp = (double *)malloc(sizeof(double) * N);
120.     //为一维数组 Markov_temp1 分配空间
121.     Markov_temp1 = (double *)malloc(sizeof(double) * m);
122.     //为二维数组 Markov 分配空间
123.     Markov = (double **)malloc(sizeof(double *) * D);
124.     for (i=0;i<D;i++)
125.     {
126.         Markov[i] = (double *)malloc(sizeof(double) * m);
127.     }
128.
```

```
129.     for (i=0;i<L;i++)
130.     {
131.         for (j=0;j<D;j++)
132.         {
133.             Markov_temp = GetMarkov(a,N,Delta,Markov_temp);
134.             for(k=0;k<m;k++)
135.             {
136.                 Markov[j][k] = Markov_temp[N0 + k*step];
137.             }
138.         }
139.         for (l=0;l<D;l++)
140.         {
141.             for (s=0;s<m;s++)
142.             {
143.                 Markov[l][s] = 1 + (Markov[l][s]/2);
144.             }
145.         }
146.         for (l=0;l<m;l++)
147.         {
148.             Markov_temp1[l] = Markov[0][l];
149.         }
150.
151.         //Markov_temp = Equal(m,Markov,Markov_temp);
152.         for (l=0;l<(D-1);l++)
153.         {
154.             for (s=0;s<m;s++)
155.             {
156.                 Markov_temp1[s] = Markov_temp1[s] * Markov[l+1][s];
157.             }
158.         }
159.         temp1 = sum(Markov_temp1,m);
160.         temp2 = 1.0 - exp((-1)*a);
161.         Intm[i] = (temp1*(pow(temp2,D)))/m;
```

```
162.         printf("i now is %ld \n",i);
163.         //fflush(stdout);
164.     }
165.     for (i=0; i<D; i++)
166.     {
167.         free(*(Markov + i));
168.     }
169.     Intm_aver = (sum(Intm,L))/L;
170.     AQS = GetAQS(Intm,L);
171.     sigma = sqrt(AQS - pow(Intm_aver,2));
172.     //free(Markov_temp);
173.     free(Markov_temp1);
174.
175.     return sigma;
176. }
177.
178. //求两个一维数组前 m 位的协方差
179. double GetCov(double Markov1[],double Markov2[],int m)
180. {
181.     double temp1[m];
182.     int i,j;
183.     double E1,E2,answer;
184.
185.     for (i=0;i<m;i++)
186.     {
187.         temp1[i] = Markov1[i]*Markov2[i];
188.     }
189.     E1 = sum(temp1,m)/m;
190.     E2 = (sum(Markov1,m)/m)*(sum(Markov2,m)/m);
191.
192.     answer = E1 - E2;
193.
194.     return answer;
```

```
195. }
196.
197. int GetPassRate(double a,int N,double Delta)
198. {
199.     double* x;
200.     double x0,temp,deltaX,temp2;
201.     int i,j,Fail_count,Success_count;
202.
203.     Fail_count = 0;
204.     Success_count = 0;
205.     //srand(time(0));
206.     //给一维数组 x 创建储存空间
207.     x = (double *)malloc(sizeof(double) * N);
208.
209.     x0 = 2 * (rand()/(double)RAND_MAX);
210.     x[0] = x0;
211.     for (i = 0;i<(N-2);i++)
212.     {
213.         temp = rand()/(double)RAND_MAX;
214.         deltaX = Delta*(temp - 0.5)*2;
215.         x[i+1] = x[i] + deltaX;
216.         while (x[i+1]>a)
217.         {
218.             x[i+1] = x[i+1] - a;
219.         }
220.         while (x[i+1]<0)
221.         {
222.             x[i+1] = x[i+1] + a;
223.         }
224.         temp2 = rand()/(double)RAND_MAX;
225.         if (GetAlpha(x[i],x[i+1])<temp2)
226.         {
227.             x[i+1] = x[i];
```

```
228.             Fail_count = Fail_count + 1;
229.         }
230.     }
231.     free(x);
232.     Success_count = N - Fail_count;
233.     return Success_count;
234. }
235.
236. double GetIntm(double a,double Delta,int N0,int step,int m,int D,int L)
237. {
238.     double*  Markov_temp;
239.     double*  Markov_temp1;
240.     //double*  Markov_temp2;
241.     double**   Markov;
242.     long int i,j,k,l,s;
243.     int N;
244.     double Intm[L];
245.     double Intm_aver;
246.     double temp1,temp2;
247.
248.     //参数初始化
249.     N = N0 + m*step;
250.     temp1 = 0.0;
251.     temp2 = 0.0;
252.     //为一维数组 Markov_temp 分配空间
253.     //Markov_temp = (double *)malloc(sizeof(double) * N);
254.     //为一维数组 Markov_temp1 分配空间
255.     Markov_temp1 = (double *)malloc(sizeof(double) * m);
256.     //为二维数组 Markov 分配空间
257.     Markov = (double **)malloc(sizeof(double *) * D);
258.     for (i=0;i<D;i++)
259.     {
```

```
260.         Markov[i] = (double *)malloc(sizeof(double) * m);
261.     }
262.
263.     for (i=0;i<L;i++)
264.     {
265.         for (j=0;j<D;j++)
266.         {
267.             Markov_temp = GetMarkov(a,N,Delta,Markov_temp);
268.             for(k=0;k<m;k++)
269.             {
270.                 Markov[j][k] = Markov_temp[N0 + k*step];
271.             }
272.         }
273.         for (l=0;l<D;l++)
274.         {
275.             for (s=0;s<m;s++)
276.             {
277.                 Markov[l][s] = 1 + (Markov[l][s]/2);
278.             }
279.         }
280.         for (l=0;l<m;l++)
281.         {
282.             Markov_temp1[l] = Markov[0][l];
283.         }
284.
285.         //Markov_temp = Equal(m,Markov,Markov_temp);
286.         for (l=0;l<(D-1);l++)
287.         {
288.             for (s=0;s<m;s++)
289.             {
290.                 Markov_temp1[s] = Markov_temp1[s] * Markov[l+1][s];
291.             }
292.         }
```

```
293.         temp1 = sum(Markov_temp1,m);
294.         temp2 = 1.0 - exp((-1)*a);
295.         Intm[i] = (temp1*(pow(temp2,D)))/m;
296.         printf("i now is %ld \n",i);
297.         //fflush(stdout);
298.     }
299.     //获得蒙特卡洛积分值
300.     Intm_aver = (sum(Intm,L))/L;
301.
302.     for (i=0; i<D; i++)
303.     {
304.         free(*(Markov + i));
305.     }
306.     free(Markov_temp1);
307.
308.     return Intm_aver;
309. }
```