MCMC 产生任意 $P(x)$ 以及 MC 积分

龚明（计算物理）

CAS Key Laboratory of Quantum Information, USTC

2022 年 12 月 2 日

MC 和积分

$$I = \int f(x)P(x)dx = \frac{1}{N}\sum_i f(x_i) \quad P(x) = \frac{1}{N}\sum_{i=1}^{N}\delta(x - x_i).$$

和 Euler、辛普森求和的主要差别：在 Euler 求和中, x_i 取等间隔；但在 MC 积分中, x_i 为某个分布 $P(x)$ 的随机数。

优点：误差由大数定理确定，和维度无关，$\delta I \sim 1/\sqrt{N}$；在 d 维 Euler 求和中，间隔为 $\delta x \sim 1/N^{1/d}$，误差为

$$\delta I \sim h^4 \sim N^{-4/d}.$$

在高维积分中，它的收敛很慢。反之，要取得很高的精度，其离散点个数 $\propto N_1^d$ 发散, 其中 N_1 为每个方向的离散点个数。

MC 积分适合处理高维积分，在费曼图等计算中发挥了重要作用。

问题的提出

$$I = \int f(x)P(x)dx = \frac{1}{N}\sum_i f(x_i).$$

这个求和的基础:

1. 产生一组随机数 x_i ($i = 1 \sim N$);
2. 这组随机数满足 $P(x)$ 分布（不需要满足独立同分布）;
3. 为了满足收敛性要求，这个分布需要收敛（满足细致平衡）;

Metropolis 采用马尔科夫链 (Markovian chain) 抽样方法（1953 年），实现了上面的数组。

定义:

$$P(x^{i+1}|x^i x^{i-1}\cdots x^1) = P(x^{i+1}|x^i).$$

Metropolis 算法

$$X^{t+1} = \begin{cases} X^{\text{new}} & \text{with probability } r \\ X^{t} & \text{with probability } (1-r) \end{cases}$$

其中

$$r = \frac{P(X^{\text{new}})}{P(X^{t})}.$$

如果 $r \geq 1, X^{t+1} = X^{\text{new}}$ 是完全接受。

Markovian Chain

$$X^1 \to X^2 \to X^3 \to X^4 \to X^5 \cdots \to X^k \to \cdots .$$

如果 k 足够大，X^i 的分布会趋于稳定，满足细致平衡条件

$$\pi P = \pi, \quad \sum_i \pi_i P_{ij} = \pi_j.$$

Detailed balance（细致平衡）

考虑动力学方程

$$\frac{dP_n}{dt} = \sum_m P_m W_{m\to n} - P_n W_{n\to m}.$$

长时间达到平衡，所以

$$\frac{dP_n}{dt} = 0.$$

这个方程存在一个特殊的解，即细致平衡（充分不必要）

$$P_m W_{m\to n} = P_n W_{n\to m}.$$

在物理中，

$$P(\mathbf{x}) \propto \exp(-\beta H(\mathbf{x})).$$

平衡和细致平衡区别

没有细致平衡，也可以平衡；比如 $A \to B \to C \to D \to A$ 的循环（类似喷泉），也可以达到平衡；但是非细致平衡。

```fortran
program main
      implicit none
      integer, parameter :: NC = 100000000
      integer, parameter :: Nd = 2000
      double precision   :: dis(-Nd:Nd), rho
      double precision   :: X0, X1
      double precision   :: ds, x, E0, E1, dE, PI, ratio
      double precision   :: PDF, P1, xtrial, ya, rr
      integer            :: i, j, Nt, k

      PI = 4.0d0 * atan(1.0d0)
      ds = PI/dble(Nd)
      open(11, file='ding.dat', status='unknown')

      dis = 0.0d0

      Nt = 1
      X0 = 0.4
```

```
! Nt是指针
do i = 2, NC
xtrial = 2.0d0 *(rand() - 0.5d0) * PI
ratio = PDF(xtrial)/PDF(X0)

! 如果ratio > 1, 则100%接受, 所以可以合并在一起.
if(rand() .le. ratio) then
    X0 = xtrial
    Nt = Nt+1
else
    ! 以1-ratio的概率接受老的结果.
    X0 = X0
    Nt = Nt+1
endif

! X0 是更新后的数据, 统计其分布.
j = int(X0/ds)
dis(j) = dis(j) + 1.0d0
end do
```

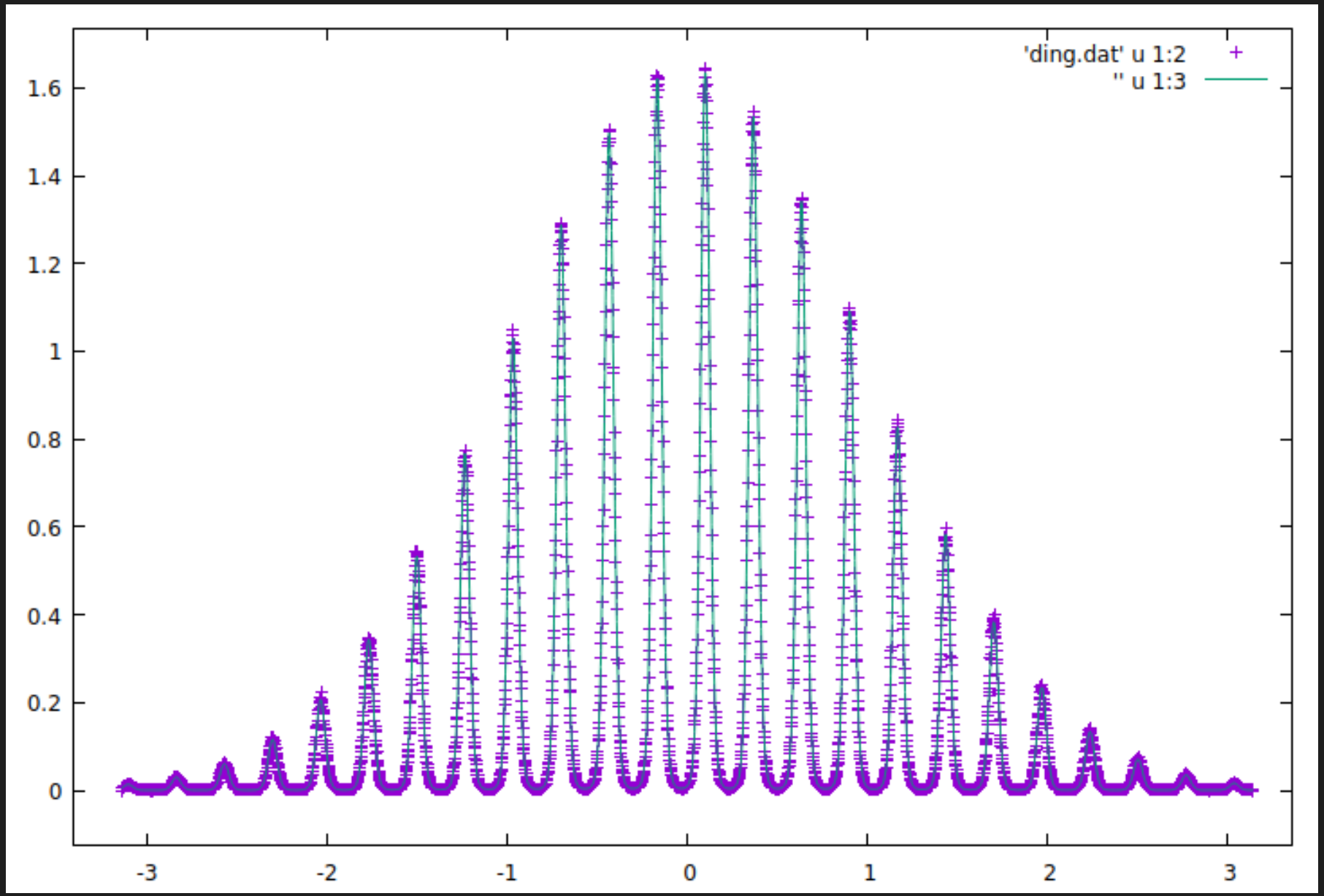
```
        ! X0 是更新后的数据, 统计其分布.
        j = int(X0/ds)
        dis(j) = dis(j) + 1.0d0
        end do

        write(*, *) " NC, Nt = ", NC, Nt

        Nt = int(Nt - dis(0)/2.0d0)
        dis(0) = dis(0)/2.0d0

        rho = 0.0d0
        do i= -Nd, Nd
        x = i * ds
        P1 = PDF(x)
        write(11, *) i * ds, dis(i)/Nt/ds, P1, dis(i)/Nt/ds - P1
        rho = rho + dis(i)/Nt
        end do

        write(*, *) " rho (should be 1.0d0) = ", rho
        close(11)
end program
```

```
double precision function PDF(x)
    implicit none
    double precision :: x
    double precision, parameter :: Z = 12.217257554294116
    PDF = exp(-0.5 * x*x - 3 *sin(23.5 *x + 2.3))/Z
end function
```

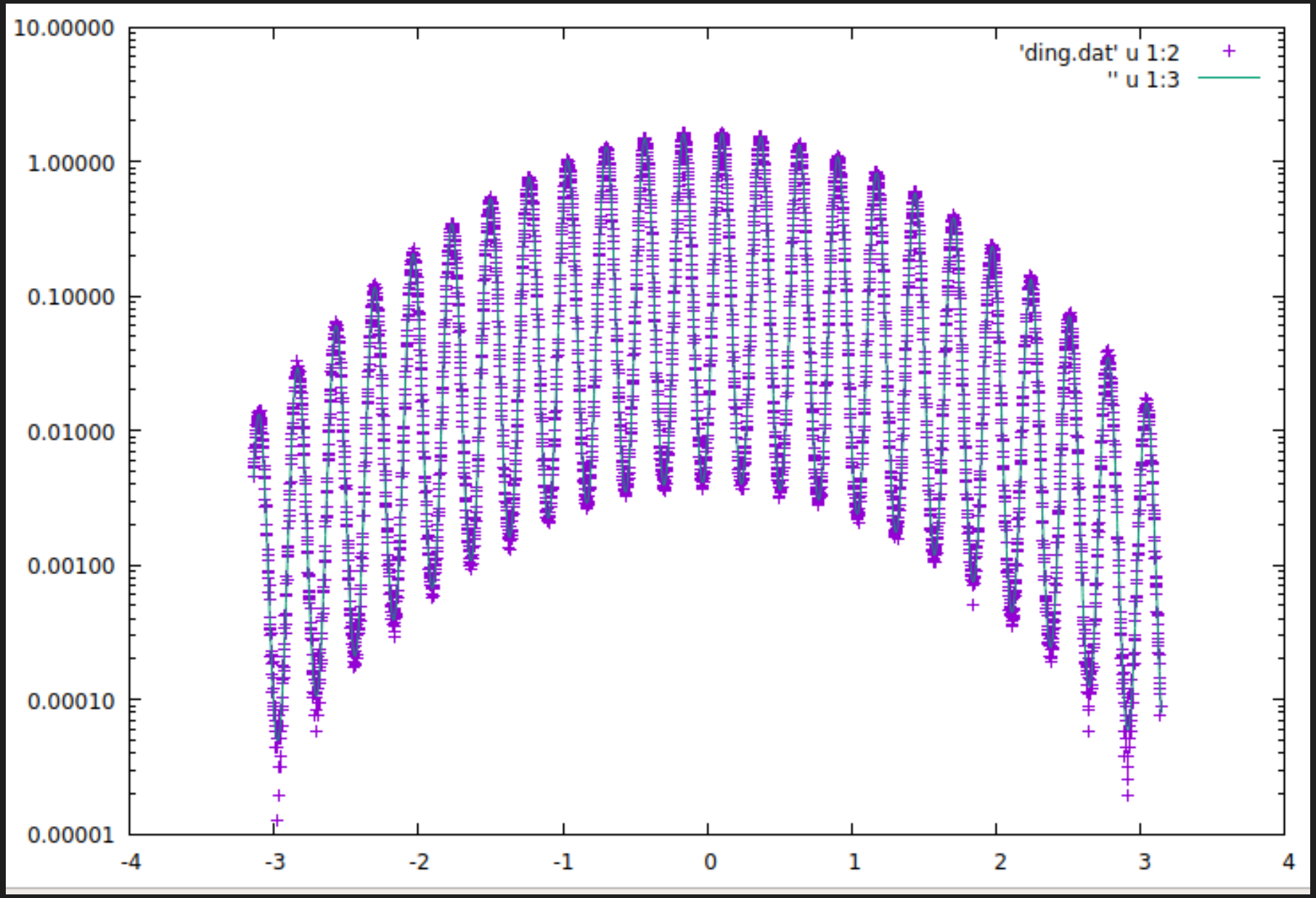
目标函数分布

$$P(x) = \frac{1}{Z}\exp\left(-0.5x^2 - 3\sin(23.5x + 2.3)\right), \quad x \in [-\pi, \pi].$$

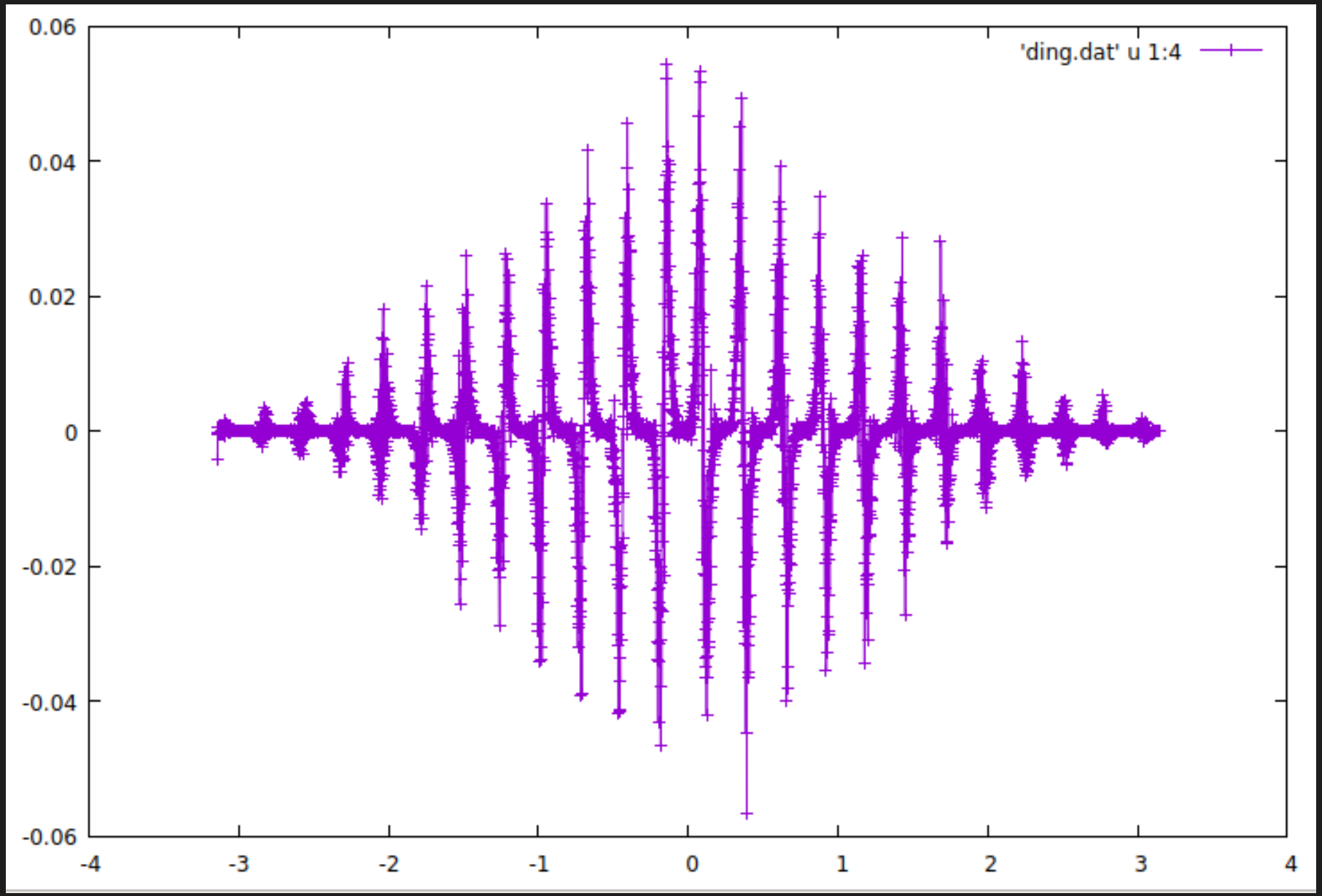
分布函数



分布函数

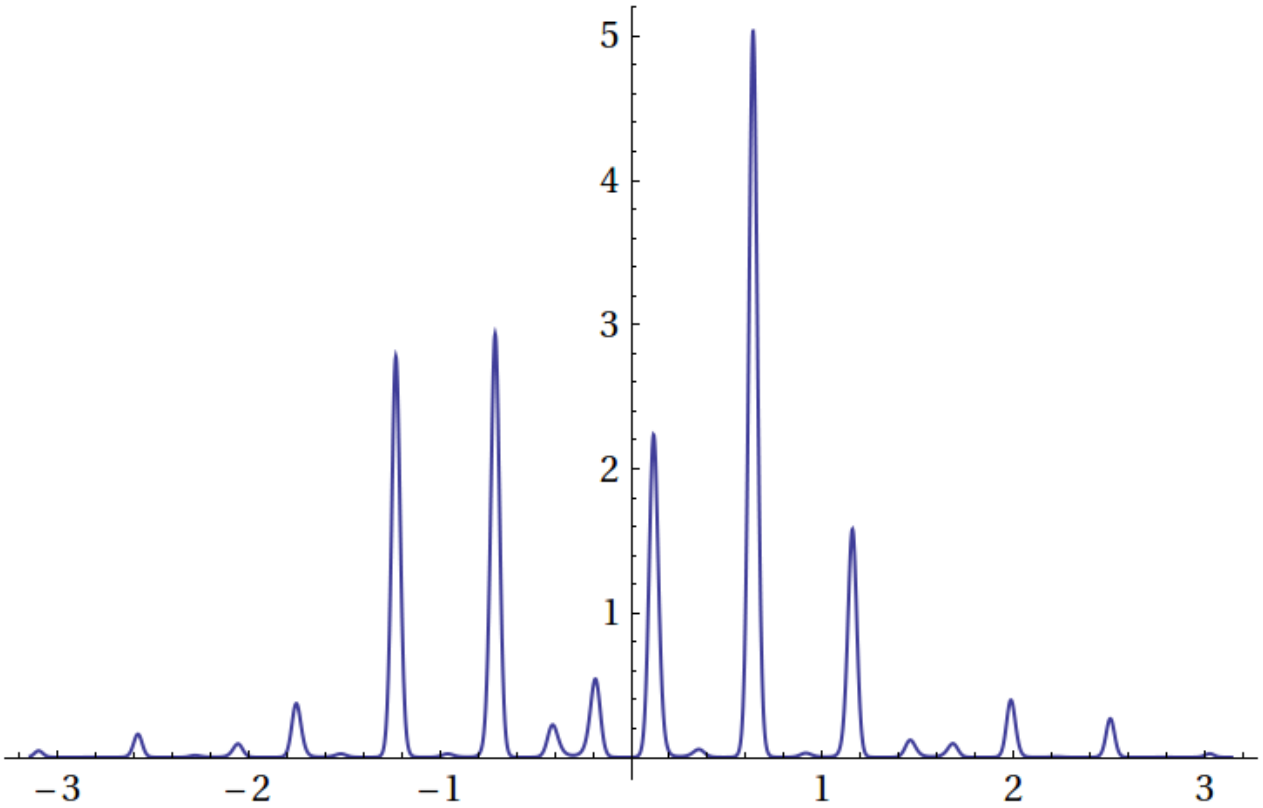

'ding.dat' u 1:2
'' u 1:3
10.00000
1.00000
0.10000
0.01000
0.00100
0.00010
0.00001
-4
-3
-2
-1
0
1
2
3
4

误差: 数值结果和解析结果的差

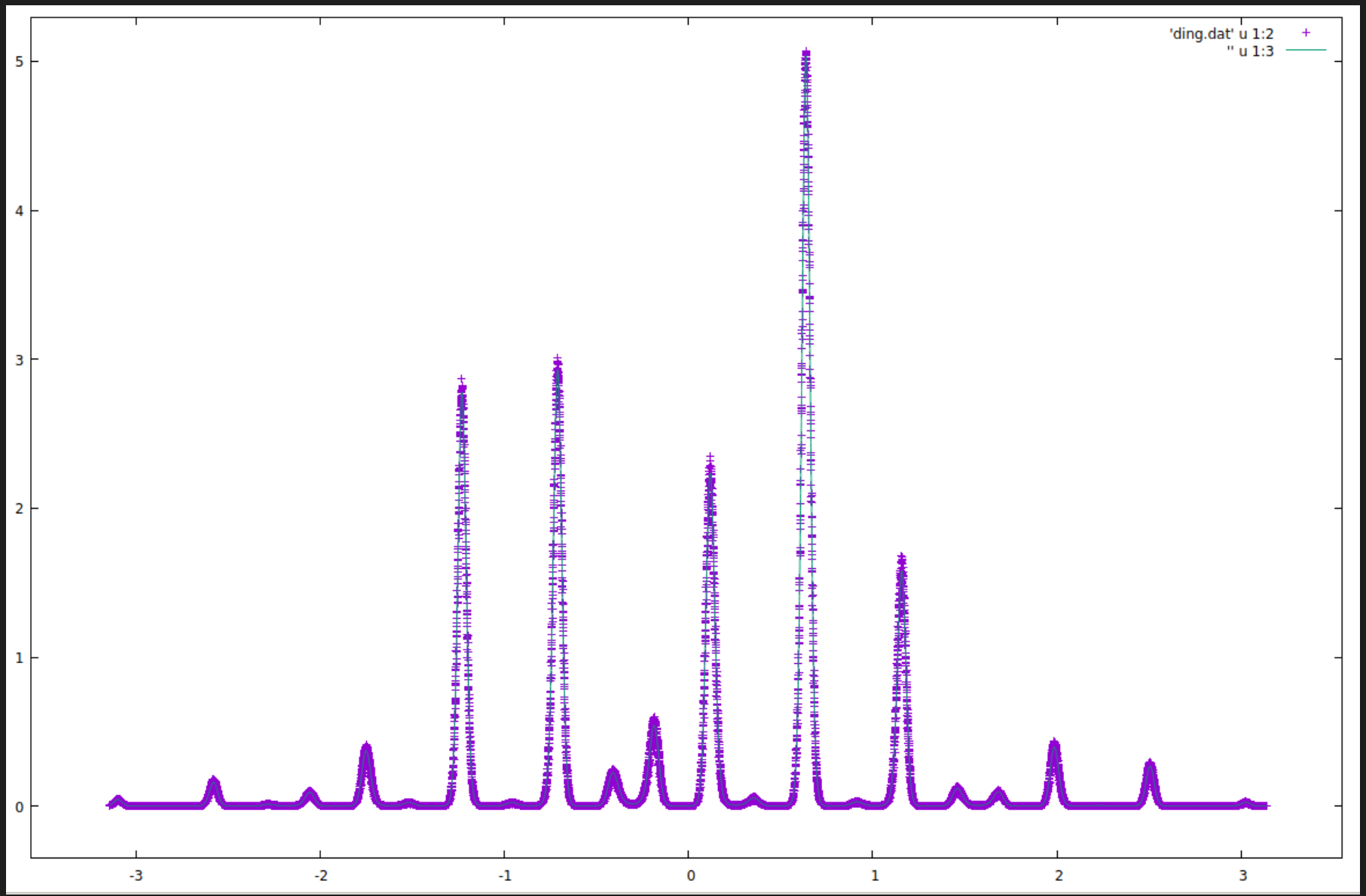


另外一个分布函数

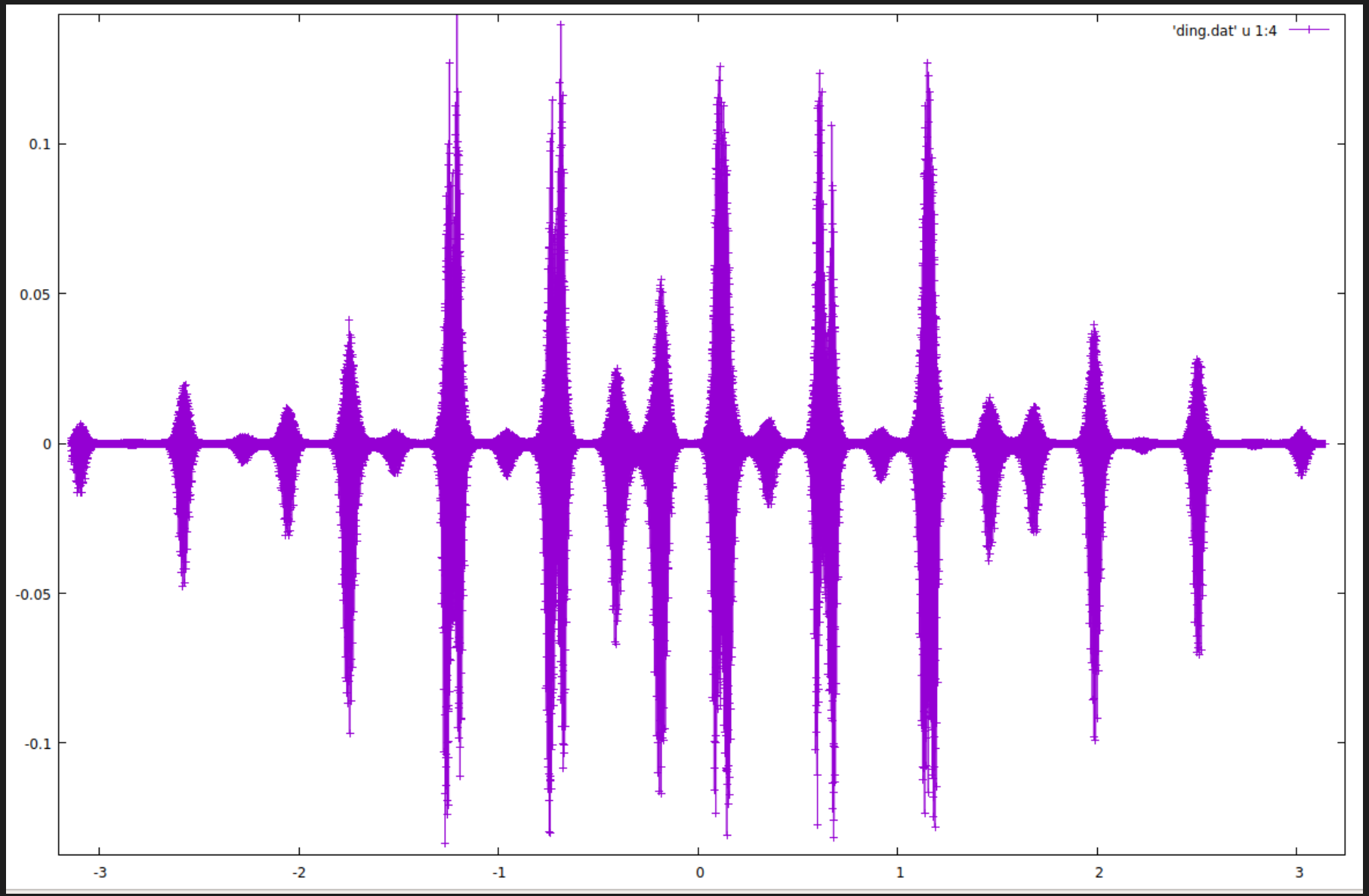
```
Z = 43.401934166947 32;
Plot[Exp[-0.5 * x * x - 3 * Sin[23.5 * x + 2.3] - 2.6 * Sin[13.5 * x + 2.3]] / Z,
 {x, -Pi, Pi}, PlotRange → All]
```



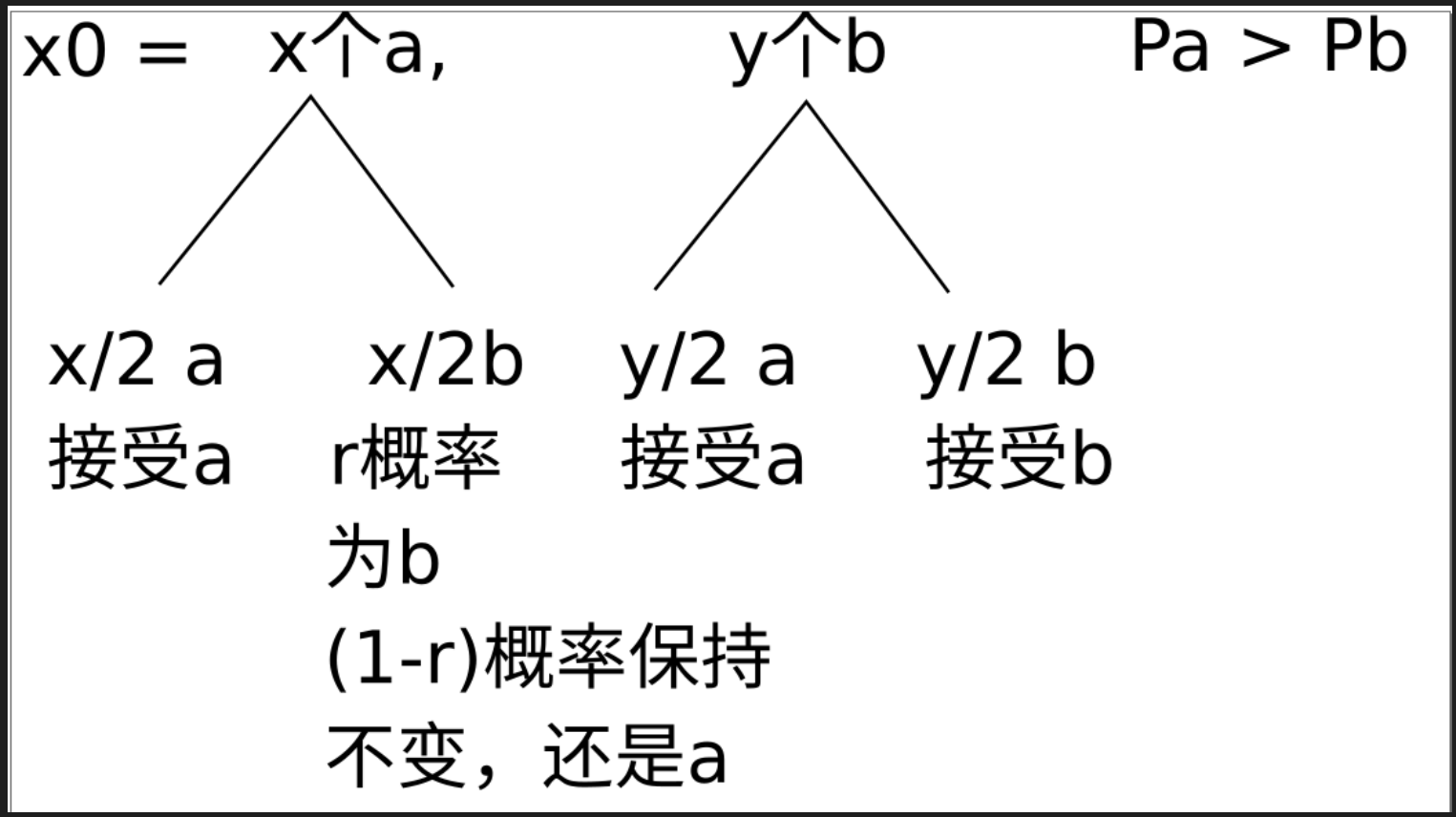
数值结果



误差



双值随机模型的细致平衡



$$N_x = x + \frac{x}{2} + (1-r)\frac{x}{2} + \frac{y}{2}, \quad N_y = y + \frac{y}{2} + x\frac{r}{2}.$$

这个概率不变，所以

$$\frac{N_y}{N_x} = \frac{y}{x} = r \to y = rx.$$

MC 数值积分（1d）

```
program main
      implicit none
      integer  :: i, NC, k
      double precision :: x, y, exactValue

      exactValue = 3.2934122860965243
      do k=1, 20
      NC = 5000000 * k
      y = 0.0d0
      do i=1, NC
      x =rand()
      y = y + exp(-x*x - 2.5 * sin(8.2 * x + 3.5))
      end do
      y = y / NC
      write(*, *) NC, y, y - exactValue

      end do
end program
```

简单起见，考虑一个简单的模型在 [0, 1] 范围内的数值积分，我们采用均匀函数做积分。

MC 数值积分：输出结果

5000000	3.292475651598824 3	-9.365569583046351 5E-004
10000000	3.294473729500526 2	1.061520943397287 2E-003
15000000	3.294432159411430 7	1.019950854301843 2E-003
20000000	3.292636165984008 2	-7.760425731206943 6E-004
25000000	3.292719927559102 8	-6.922809980260957 7E-004
30000000	3.292975003883156 7	-4.372046739722357 2E-004
35000000	3.293656215746069 8	2.440071889409090 0E-004
40000000	3.293371666682964 9	-4.054187416402399 4E-005
45000000	3.293500818212944 4	8.860965581547475 3E-005
50000000	3.293583850547516 8	1.716419903878829 7E-004
55000000	3.293418659140993 7	6.450583864836545 4E-006
60000000	3.293641961373257 0	2.297528161281370 3E-004
65000000	3.293819313271909 4	4.071047147804485 2E-004
70000000	3.294044850633400 6	6.326420762716722 4E-004
75000000	3.293328374299020 9	-8.383425810798428 3E-005
80000000	3.293198022462776 8	-2.141860943520868 0E-004
85000000	3.292562977401341 4	-8.492311557874998 6E-004
90000000	3.293563262473170 1	1.510539160411994 4E-004
95000000	3.293141763370512 0	-2.704451866168966 5E-004
100000000	3.293859562588621 8	4.735403149287478 4E-004

MC 数值积分（2d）

```
NIntegrate[Exp[-x1*x1 - x2*x2 - 2.5*Sin[8.2*x1 + 2.5*x2 + 3.5]],
 {x1, 0, 1}, {x2, 0, 1}]


2.05326
```

```
program main
      implicit none
      integer  :: i, NC, k
      double precision :: x1, x2, y, exactValue

      exactValue = 2.0532571251259997
      do k=1, 20
      NC = 5000000 * k
      y = 0.0d0
      do i=1, NC
      x1 =rand()
      x2 =rand()
      y = y + exp(-x1*x1 - x2*x2 - 2.5 * sin(8.2 * x1 + 2.5 * x2 + 3.5))
      end do
      y = y / NC
      write(*, *) NC, y, y - exactValue

      end do
end program
```

MC 数值积分：输出结果

5000000	2.053916042586727 6	6.588156427578262 3E-004
10000000	2.053761844918000 2	5.046179740304523 9E-004
15000000	2.052028389763485 0	-1.228837180484720 9E-003
20000000	2.052314429168134 3	-9.427977758353911 5E-004
25000000	2.052066499871394 0	-1.190727072575725 7E-003
30000000	2.052980405563923 6	-2.768213800461438 7E-004
35000000	2.054300109203449 4	1.042882259479682 7E-003
40000000	2.053407782482140 8	1.505555381711154 4E-004
45000000	2.053884718753594 9	6.274918096251802 5E-004
50000000	2.052889865707360 0	-3.673612366097422 9E-004
55000000	2.053537855283249 9	2.806283392802200 2E-004
60000000	2.053006069208113 8	-2.511577358559335 9E-004
65000000	2.053225631567400 5	-3.159537656927113 9E-005
70000000	2.053432002495266 1	1.747755512964133 1E-004
75000000	2.053198899775929 0	-5.832716804077264 8E-005
80000000	2.053366589378717 0	1.093624347472399 4E-004
85000000	2.053438536833530 8	1.813098895611098 3E-004
90000000	2.053465416570769 5	2.081896267998040 2E-004
95000000	2.053264244859160 9	7.017951911986537 E-006
100000000	2.053330451079832 0	7.322413586230425 2E-005

MC 数值积分（5d）

```
program main
      implicit none
      integer  :: i, NC, k
      double precision :: x1, x2, x3, x4, x5, z1, z2, y
      double precision :: exactValue, fc

!     exactValue = 2.0532571251259997
      do k=1, 20
      NC = 5000000 * k
      y = 0.0d0
      do i=1, NC
      x1 =rand()
      x2 =rand()
      x3 =rand()
      x4 =rand()
      x5 =rand()
      z1 = x1*x1 + x2*x2 + x3*x3 + x4*x4 + x5*x5
      z2 = 8.2*x1 + 2.5 *x2 + 3.5 + 2.5 *x3 *x5 + 3.8 *x4
      fc = exp(-z1 - 2.5 * sin(z2))
      y = y + fc
      end do
      y = y / NC
      write(*, *) NC, y
      end do
end program
```

```
  5000000  0.67860948226564166
 10000000  0.67742626316973287
 15000000  0.67798798940447780
 20000000  0.67820055248730726
 25000000  0.67807490833263273
 30000000  0.67798914219545636
 35000000  0.67759941227700249
 40000000  0.67797775285925144
 45000000  0.67804110469780288
 50000000  0.67812413744218381
 55000000  0.67825987468634252
 60000000  0.67809202467232188
 65000000  0.67807602696560376
 70000000  0.67787588451565672
 75000000  0.67792679786073951
 80000000  0.67814669059593469
 85000000  0.67795048534480384
 90000000  0.67820270255044190
 95000000  0.67802358679034946
100000000  0.67804839115221516
```

对于高维积分，MC 方法可能比常见的数值方法更加精确。当然，不是说别的方法完全失效—但要非常小心。

Mathematica 5d 数值积分

```
NIntegrate[
 Exp[-x1 * x1 - x2 * x2 - x3 * x3 - x4 * x4 - x5 * x5 -
   2.5 * Sin[8.2 * x1 + 2.5 * x2 + 3.5 + 2.5 * x3 * x5 + 3.8 * x4]],
 {x1, 0, 1}, {x2, 0, 1}, {x3, 0, 1}, {x4, 0, 1}, {x5, 0, 1}]
```

```
NIntegrate::eincr :
  The global error of the strategy GlobalAdaptive has increased more than 2000 times. The
      global error is expected to decrease monotonically after a number of integrand
      evaluations. Suspect one of the following: the working precision is insufficient
      for the specified precision goal; the integrand is highly oscillatory or it is not a
      (piecewise) smooth function; or the true value of the integral is 0. Increasing
      the value of the GlobalAdaptive option MaxErrorIncreases might lead to a
      convergent numerical integration. NIntegrate obtained 0.6780027207036384`
      and 0.00006488683203321574` for the integral and error estimates. >>

0.678003
```

注意这些报错，一定要非常小心；希望大家慎重采用 Mathematica 处理高维积分。

Cuba –a library for multidim. integration

The Cuba library offers a choice of four independent routines for multidimensional numerical integration: Vegas, Suave, Divonne, and Cuhre.

They work by very different methods, summarized in the following table:

Routine	Basic integration method	Algorithm type	Variance reduction
Vegas	Sobol quasi-random sample *or* Mersenne Twister pseudo-random sample *or* Ranlux pseudo-random sample	Monte Carlo Monte Carlo Monte Carlo	importance sampling
Suave	Sobol quasi-random sample *or* Mersenne Twister pseudo-random sample *or* Ranlux pseudo-random sample	Monte Carlo Monte Carlo Monte Carlo	globally adaptive subdivision + importance sampling
Divonne	Korobov quasi-random sample *or* Sobol quasi-random sample *or* Mersenne Twister pseudo-random sample *or* Ranlux pseudo-random sample *or* cubature rules	Monte Carlo Monte Carlo Monte Carlo Monte Carlo deterministic	stratified sampling, aided by methods from numerical optimization
Cuhre	cubature rules	deterministic	globally adaptive subdivision

All four have a C/C++, Fortran, and Mathematica interface and can integrate vector integrands. Their invocation is very similar, so it is eas
another for cross-checking. For further safeguarding, the output is supplemented by a χ^2-probability which quantifies the reliability of the err

The source code compiles with gcc, the GNU C compiler. The C functions can be called from Fortran directly, so there is no need for adapte
code with the library is straightforward and requires no extra tools.

In Fortran and C/C++ the Cuba library can (and usually does) automatically parallelize the sampling of the integrand.

总结

1. 操作简单，编程容易，易于实践；
2. MC 积分收敛速度较慢；
3. 可以很容易达到 $10^{-3} - 10^{-4}$ 的精度，但要取得更好的精度，需要采取一些数值技巧；
4. 如果要求精度不高，计算时间很快（$\sim$ 秒 - 分钟量级）；

本课程目标：

1. 会利用 MCMC 产生任意概率分布，理解其意义；
2. 会做简单的高维积分的计算；
3. 会利用 MC 方法研究多体模型的相变（主要是 Ising 模型）；

例子

$$I = \int_0^{\pi} \sin(x)dx = 2.$$

用离散法

$$I = \pi \sum_j w_j \sin(x_j) = \pi \sum_j \frac{1}{N} \sin(x_j), \quad x_j = jdx = \frac{\pi j}{N}.$$

我们用随机数做这个积分

$$I = \pi \sum_j \frac{1}{N} \sin(x_j), \quad x_j = \text{rand}(0, \pi).$$

比如数值结果

N	I	err = |I - 2.0|
1000000	2.001132768175370	1.1327681753703 44E-003
9000000	2.000037021570377 3	3.7021570377326896E-005
36000000	2.000065808924955 7	6.5808924955668147E-005
100000000	1.999946954950957 9	5.3045049042133030E-005

典型特征：N 要求很大，才能得到正确的结果。

利用 Mathematica 可以计算

$$I = \int_0^{\pi} dxdy \sin\left(x^2 + y^3 + \cos(xy)\right) = 0.6134018336782885.$$

利用随机数积分

$$I = \pi^2 \sum_i \frac{1}{N} \sin\left(x_i^2 + y_i^3 + \cos(x_i y_i)\right).$$

```
do i=1, N
x = rand() * pi
y = rand() * pi
z = z + pi * pi* sin(x*x + y*y*y + cos(x*y))
!z = z + pi * sin(x)
end do
z = z /dble(N)
write(*, *) N, z, abs(z - 0.6134018336782885)
```

N	I	err = |I - I0|
1000000	0.6027457065364477 7	1.065612365993284 8E-002
4000000	0.6100856531635082 7	3.316177032872347 2E-003
25000000	0.6136670548208795 8	2.652246244989608 9E-004
49000000	0.6148635236798022 4	1.461693483421622 2E-003
100000000	0.6137131214423670 0	3.112912459863892 4E-004

维度对计算复杂度、计算时间等影响不大，err $\sim 1/\sqrt{N}$。

推广到高维（解析结果检验）

$$I = \int_0^{\pi} dx_1 dx_2 dx_3 \cdots dx_8 \prod_{i=1}^{8} \sin(x_i) = 2^8 \simeq \pi^8 \sum_i \frac{1}{N} \prod_j \sin(x_j(i)).$$

```
do i=1, N
x1 = rand() * pi
x2 = rand() * pi
x3 = rand() * pi
x4 = rand() * pi
x5 = rand() * pi
x6 = rand() * pi
x7 = rand() * pi
x8 = rand() * pi
tmp = sin(x1) * sin(x2) * sin(x3) * sin(x4) * sin(x5) * sin(x6) * sin(x7) * sin(x8)
z = z + pi **8 * tmp
Np = Np +1
end do
z = z /dble(Np)
write(*, *) Np, z, abs(z - 2.0d0**8)
```

Area

N	I	err = |I - I0|
1000000	255.79912196852166	0.2008780314783393 8
4000000	256.21560699536900	0.2156069953690007 4
16000000	255.95219412687362	4.780587312637862 9E-002
36000000	255.92368260278087	7.631739721912822 5E-002
64000000	255.96753171906013	3.246828093986664 5E-002
100000000	256.00454944361570	4.549443615701420 6E-003

相当于每个方向做 10 个网格，精度达到 10^{-3}。维度越高，这个方法的优势越明显。

一般性理论

将上面的结果推广到任意维度、任意曲面 Ω:

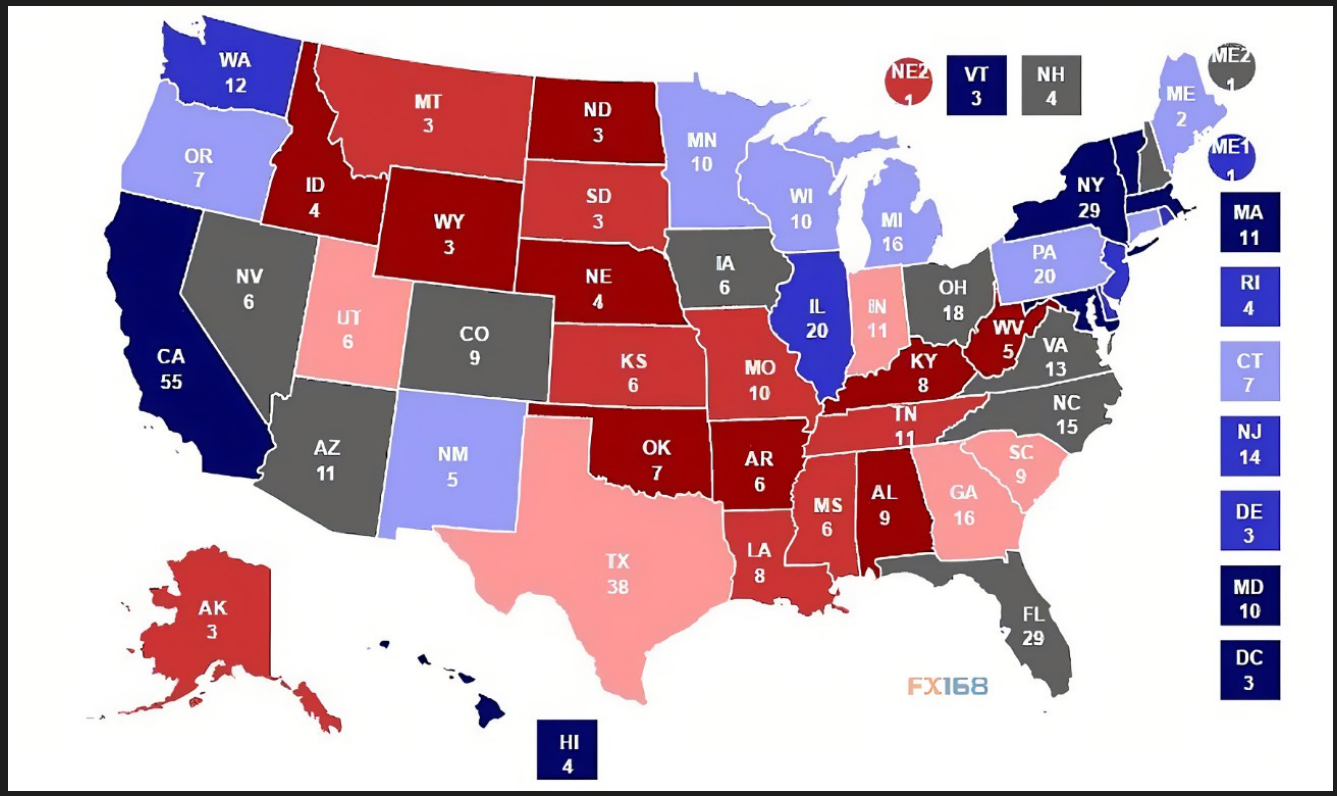
$$I = \int_{\mathbf{x}\in\Omega} dx f(x) \simeq \text{Area} \times \sum_{\mathbf{x}_i\in\Omega} \frac{1}{N} f(\mathbf{x}_i), \quad \mathbf{x}_i \text{ is uniform rand.}$$

取 $f(x,y) = \sin\left(x^2 + y^3 + \cos(xy)\right)$, $\Omega = \{x^2 + y^2 < \pi | x \geq 0, y \geq 0\}$.
Mathematica 得到 $I = 1.0529392707752274$。

```
do k=1, 10
N = 1000000 * k * k
z = 0.0d0
Np = 0
do i=1, N
x = rand() * pi
y = rand() * pi
if(x*x + y*y - 3 .lt. 0.0d0) then
z = z + pi * 3* 0.25 * sin(x*x + y*y*y + cos(x*y))
Np = Np +1
endif
end do
z = z /dble(Np)
write(*, *) Np, z, abs(z - 1.0529392707752274)
end do
```

Area

N	I	err = |I - I0|
237818	1.0504987950139868	2.440500754750960E-003
3816917	1.0527085566653043	2.307391034335193E-004
11701153	1.0529981557491306	5.885998039278916E-005
23869444	1.0529422608710950	2.965102357199711E-006



类似：选举中的民意调查，只需要调查少部分人，就可以了解大概情况。因为选举要求精度不高（可能允许 5% 左右的误差），则可以调查很少的人得到一个结论。

- 另：金融衍生产品定价和交易风险估计，也涉及很高的维度，必须采用蒙特卡洛方法。而传统的方法已经无法得到可靠结果（“维度灾难”）。

随机函数分布 $\rho(x)$

假设 x 不是均匀分布，而是其它分布，比如 $\rho(x)$

$$\rho(x) = \frac{1}{N}\int \delta(x - x_i), \quad \int_a^b \rho(x)dx = 1.$$

那么计算

$$I = \frac{1}{N}\sum_i \frac{g(x_i)}{\rho(x_i)}.$$

利用

$$\frac{1}{N}\sum_i h(x_i) = \int_a^b h(x)\rho(x)dx.$$

所以我们有

$$I = \frac{1}{N}\sum_i \frac{g(x_i)}{\rho(x_i)} = \int_a^b f(x)dx.$$

困难：需要从均匀分布找到任意分布 $\rho(x)$，很不方便。推广到高维更困难。

物理学中的蒙特卡洛算法

我们计算热力学量

$$Z = \int d\mathbf{x} d\mathbf{p} \exp(-\beta H(\mathbf{x}, \mathbf{p})),$$

我们简化为 $\mathbf{q} = (\mathbf{x}, \mathbf{p}) \in \Gamma$

$$Z = \int d\mathbf{q} \exp(-\beta H(\mathbf{q})).$$

为了保证平衡，涉及关键概念

- 细致平衡
- 重要抽样
- 数学概念：Markovian 过程

把物理问题转化为高阶积分问题。物理中会更加复杂，因为需要考虑量子效应（略过）。

以后介绍：这个方法可以推广到其它非指数形式：重要抽样。

考虑一个复杂的网络（收入、信息等）

$$P_i(t+dt) - P_i = \sum_j P_i W_{i\to j} - P_j W_{j\to i}.$$

平衡，代表某种稳定状态，出现条件

$$P_i(t+dt) = P_i(t).$$

比如某家家庭的收支平衡，表明它的支出和收入互相抵消。细致平衡（特解）

$$P_i W_{i\to j} = P_j W(j \to i).$$

物理中

$$P_i = \exp(-\beta H(\mathbf{q}_i))/Z \propto \exp(-\beta H(\mathbf{q}_i)).$$

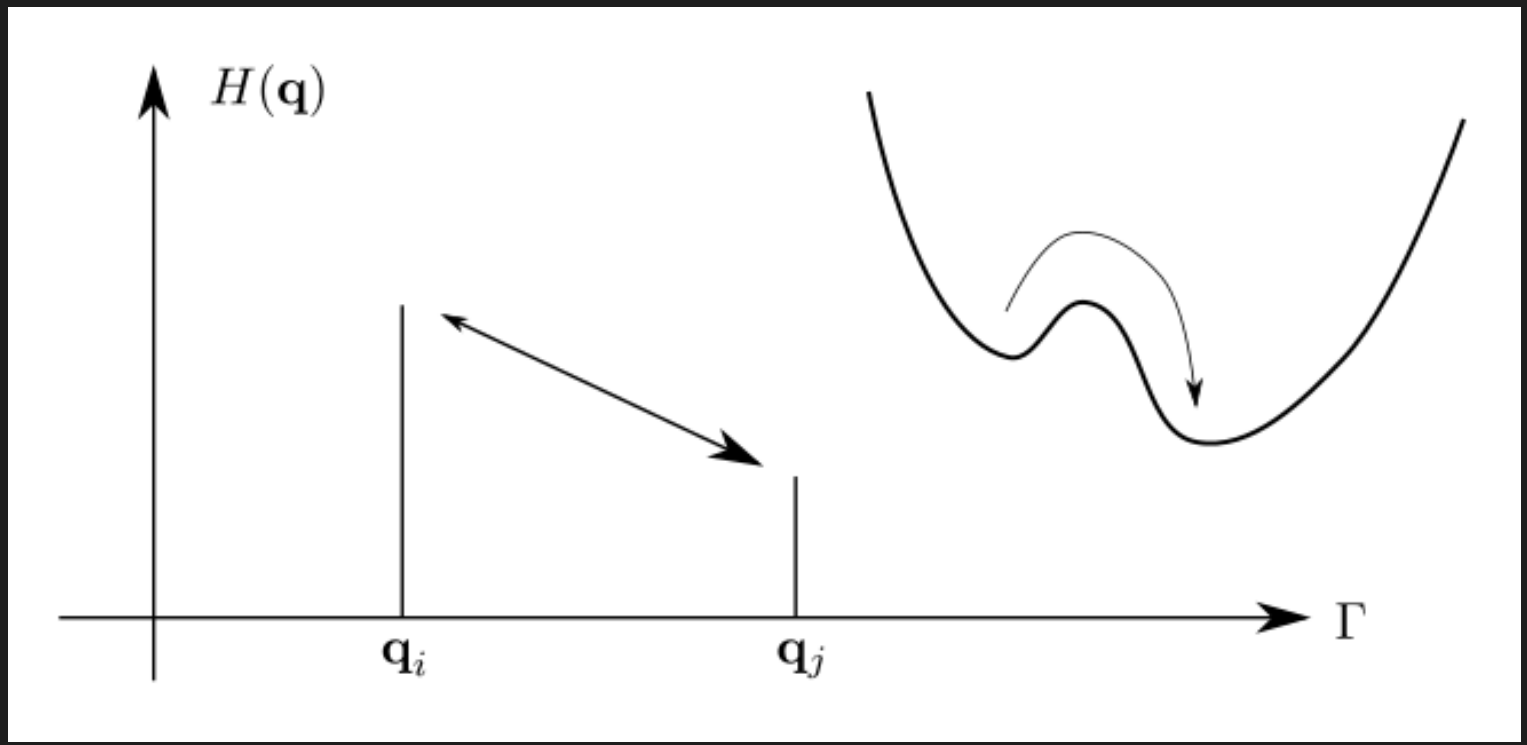
所以

$$\frac{W_{i\to j}}{W_{j\to i}} = \frac{P_j}{P_i} = \exp(\beta \delta E_{ij}).$$

其中

$$\delta E_{ij} = H(\mathbf{q}_i) - H(\mathbf{q}_j).$$

显而易见，能量会从高处“流向”低处；能量越低，占据几率越大。



Metropolis 抽样

- 对 $\mathbf{q}_i$，计算 $E_1 = H(\mathbf{q}_i)$
- 随机找一个新的 $\mathbf{q}'$，计算 $E_2 = H(\mathbf{q}')$
- 如果 $E_1 > E_2$, 则表明新构形能量更低，$W_{1\to 2} = 1$, $\mathbf{q}_{i+1} = \mathbf{q}'$
- 如果 $E_1 < E_2$, 表明新构型能量更高，以概率 $\eta = e^{-\beta(E_2 - E_1)}$ 接受新构型

$$\mathbf{q}_{i+1} = \begin{cases} \mathbf{q}' & \xi < \eta, \\ \mathbf{q}_i & \xi > \eta. \end{cases}$$

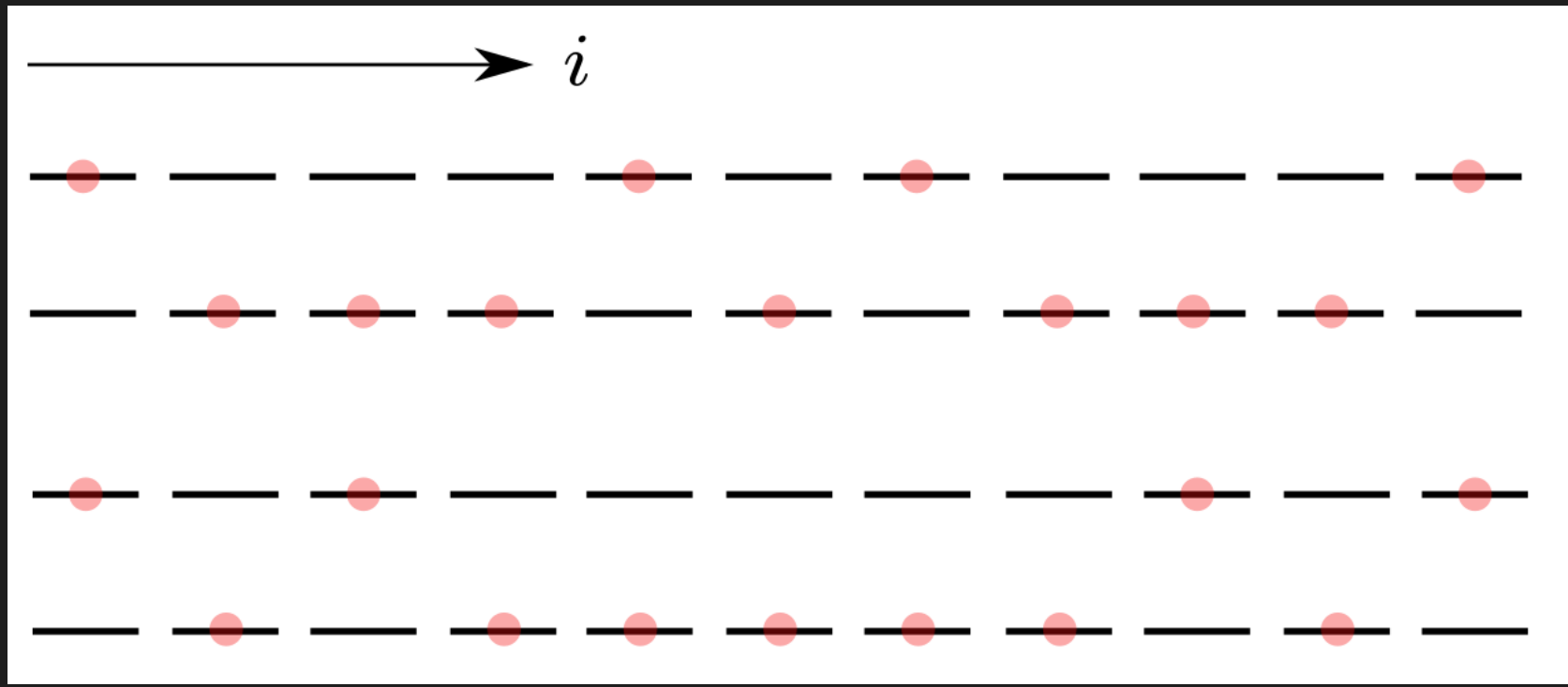
得到（马尔科夫链）序列

$$\mathbf{q}_1, \quad \mathbf{q}_2, \cdots, \quad \mathbf{q}_i, \quad \cdots.$$

配分函数（参考重要抽样）

$$\langle A \rangle = \frac{\int A \exp(-\beta H) d\mathbf{x}}{Z} = \frac{1}{A} \sum_i A_i.$$

一个简单的图像



考虑两个能级，如果

$$W_{i\to j} = \min(1, \exp(-\beta\delta E_{ij}), \quad \delta E_{ij} = E_i - E_j.$$

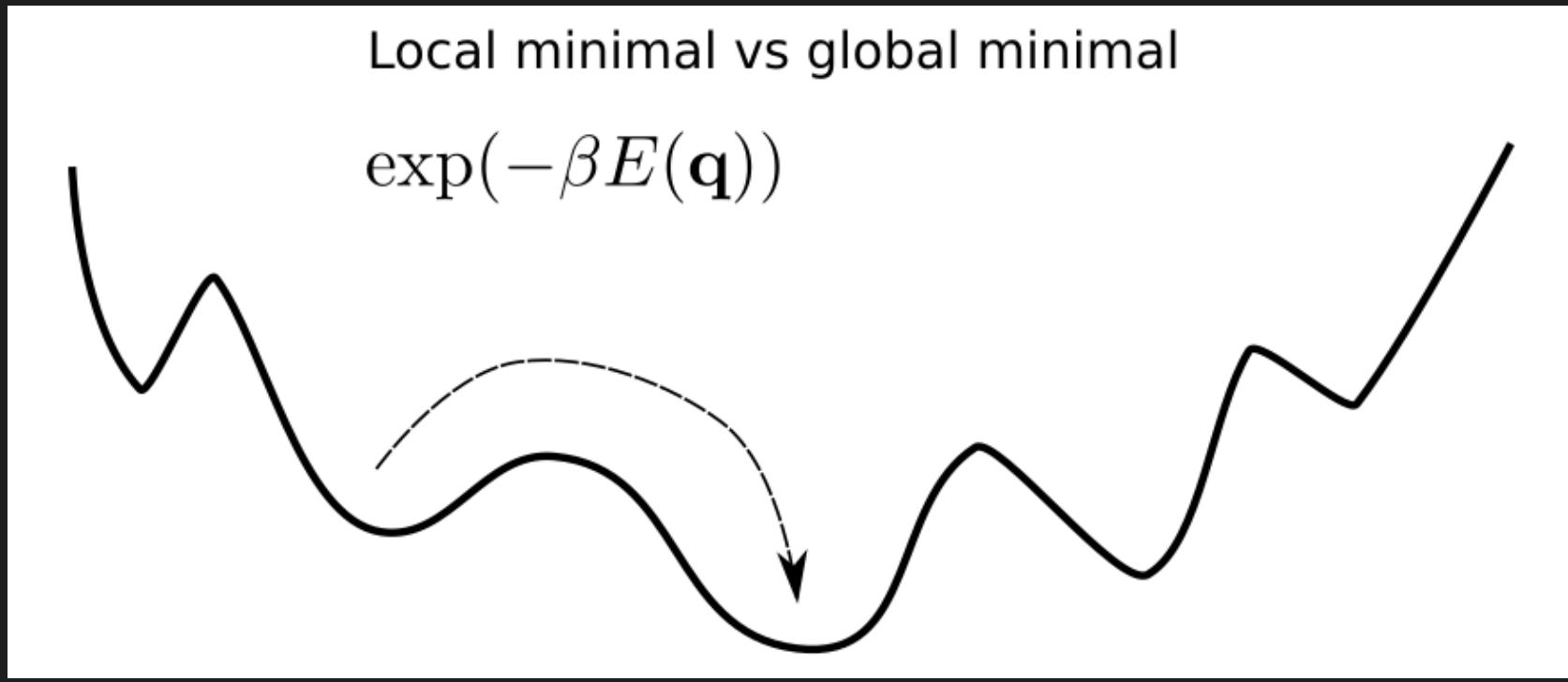
那么产生的序列

$$P_i \propto \exp(-\beta E_i), \quad i = 1, 2.$$

蒙特卡洛模拟和非线性优化问题

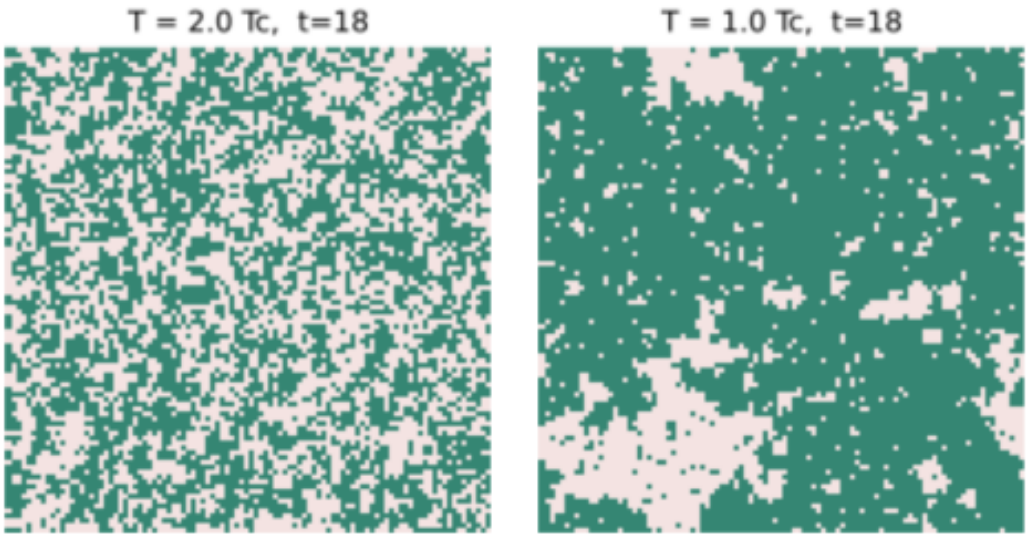
旅行推销员问题、（材料）结构优化问题，都涉及到非线性方程的最小值

$$E = E(q_1, q_2, \cdots, q_N) = E(\mathbf{q}).$$

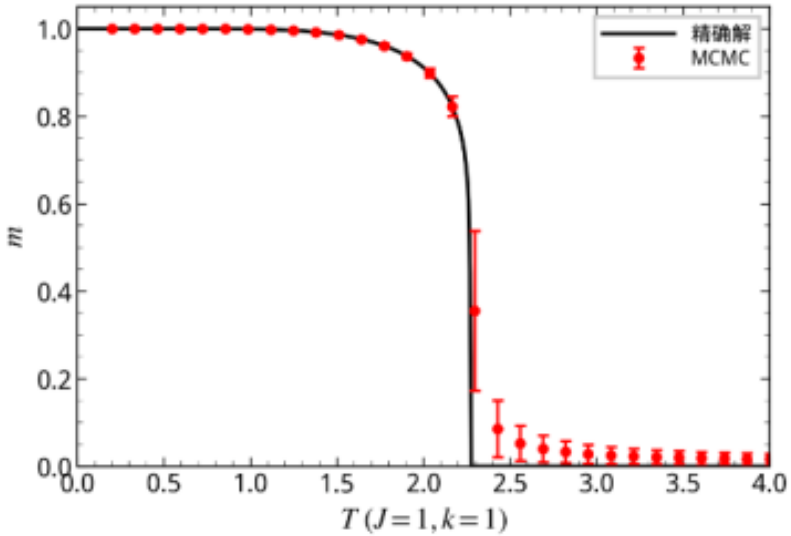


引入一个假设的温度 β 分布，可以跳出 local minimal, 进入 global minimal。

应用：Ising 模型的相变



二维伊辛模型 MCMC 模拟



二维伊辛模型单位格点磁矩随温度的变化

最经典的应用是研究 2D Ising 模型的自发磁化（尺寸可以很大，上面的例子，$N = 100 \times 100$，Hilbert 空间维度 $|\mathcal{K} = 2^{10^4}|$）

$$H = -J \sum_{\langle i,j \rangle} s_i s_j, \quad s_i = \pm 1.$$

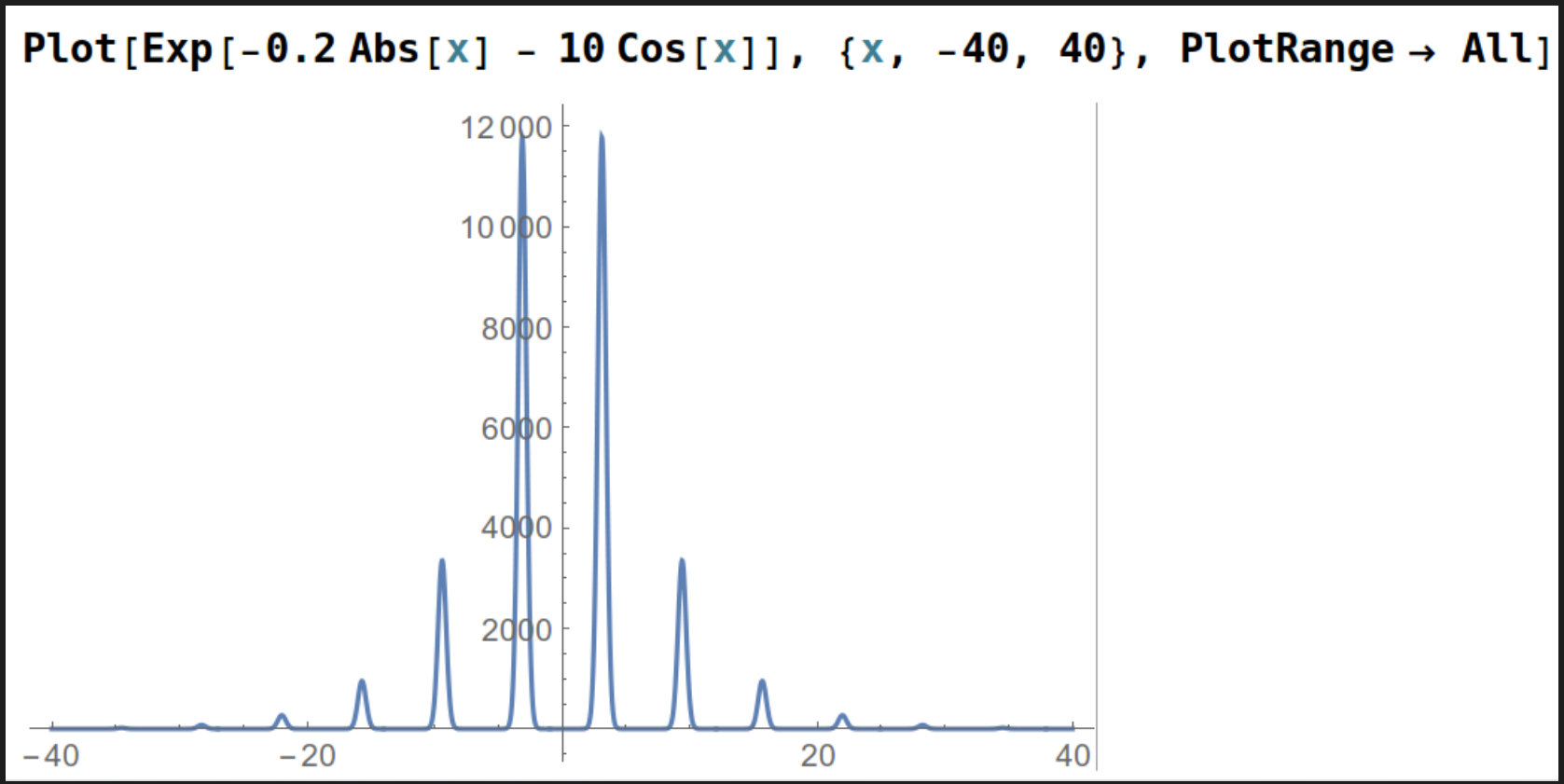
临界温度发生在

$$k_B T_c = \frac{2}{\ln(1+\sqrt{2})} J = 2.26919 J.$$

来自物理所物理学中的蒙特卡洛方法，夏晨.

困难

```
Plot[Exp[-0.2 Abs[x] - 10 Cos[x]], {x, -40, 40}, PlotRange → All]
```



许多函数，曲线很复杂，积分有主要贡献和次要贡献 — 次要贡献不能忽略。

- 舍弃次要贡献：引入很大误差
- 不舍弃次要贡献：消耗大量计算资源

重要抽样（Important sampling）

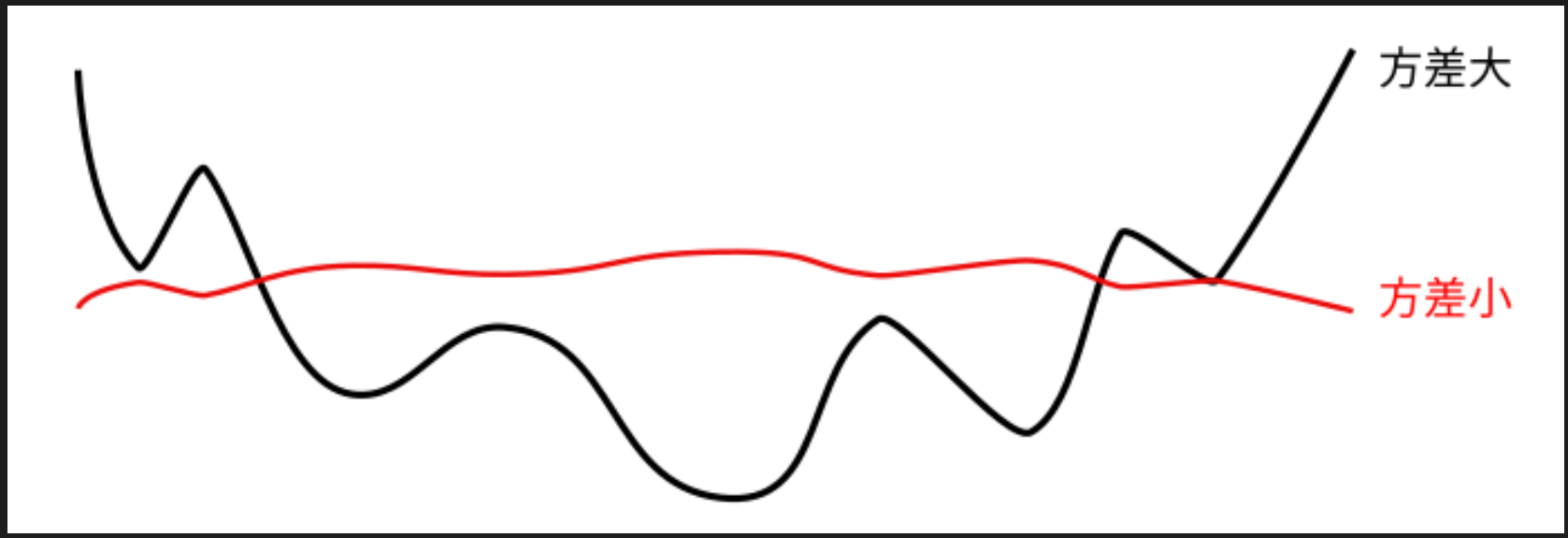
前面提到（非均匀抽样）

$$\text{Area} \times \frac{1}{N}\sum_i \frac{f(\mathbf{x}_i)}{\rho(\mathbf{x}_i)} = \int_{\mathbf{x}_i\in\Omega} f(\mathbf{x})d\mathbf{x}.$$

其误差/方差

$$\sigma_f = \frac{1}{\sqrt{N}}\sigma\left(\frac{f(\mathbf{x})}{\rho(\mathbf{x})}\right).$$

所以，函数 f/ρ 越平坦，方差越小；越不平坦，方差越大。



如果 $\rho \to f$（不可能，因为 f 可能出现振荡），则方差可以尽量减小。

利用 Metropolis 算法，保证细致平衡，可以实现

$$P_i \propto \exp(-\beta E_i).$$

我们可以稍加改造，实现任意概率分布，比如

$$P(\mathbf{x}) = \rho(\mathbf{x}) \geq 0.$$

只需要将 $E_i = -\beta^{-1} \ln \rho(\mathbf{x}_i)$ 即可。

假设有一个分布函数 $\rho(\mathbf{x})$，非常靠近 f，则可以显著降低计算误差。