Markov Chain Monte Carlo 法

— 簡単な例 —

茅根裕司

2005/12/08

MCMC 法の感じを掴むために、簡単な例を試してみる（荒井修論参照）。

1 Bayesian の考え方

始めに MCMC 法の根幹を成すベイズ理論について簡単に説明する。

1.1 主観確率

ベイズ推量の世界では、全ての確率は主観的な確率 (subjective probability) だとされている。普通、確率というのは対象の不確かさを示す量とされるが、ベイズ推量では不確かさはその対象を観測する人の知識にあると考える。

1.2 事前確率

観測者は研究対象である確率変数 X に関して、事前確率 $\Pr(X)$ を持っている。**事前確率 (prior) とは、観測者が観測以前に持っている主観確率を確率分布関数で表現したもの**である。

1.3 事後確率

観測者は観測を行う。観測した結果を D とする。観測結果 D によって、確率変数 X に関する主観確率が変化する。**観測結果 D のもとで確率変数 X に関する主観確率を $\Pr(X|D)$ と書き、事後確率 (posterior)** と呼ぶ。**この事後確率分布を求めるのがベイズ推量の目的**である。

1.4 尤度

確率変数 X の元では、観測結果 D が確率 $\Pr(D|X)$ に従って生成されるはずであることが分かっていたとする。$\Pr(D|X)$ を D が観測された時の X の**尤度 (likelihood)** と呼び、これを尤度関数 $\Pr(D|X)$ で定義された確率的生成モデルと呼ぶことがある。X の値によって D の出方が変わるという意味で、X を $\Pr(D|X)$ のパラメータと呼ぶことがある。

1.5 ベイズの定理

事前確率 $\Pr(X)$、事後確率 $\Pr(X|D)$、尤度 $\Pr(D|X)$ の間には以下の関係が成り立つ。これをベイズの定理という。分母の値を特に $\Pr(D)$ と書き証拠 (evidence) と言う。

$$\Pr(X|D) = \frac{\Pr(D|X)\Pr(X)}{\sum_{X\ \text{の全ての場合}} \Pr(D|X)\Pr(X)} \quad \text{or} \quad \Pr(X|D)dX = \frac{\Pr(D|X)\Pr(X)dX}{\int \Pr(D|X)\Pr(X)\,dX} \tag{1}$$

2 Markov chain Monte Carlo

2.1 モンテカルロ積分

モンテカルロ積分は事後確率分布 $\Pr(X|D)$ からサンプル $\{X_t;\ t = 1, 2, \ldots, n\}$ を取り出すことで期待値 $E[f(X)]$ を

$$E[f(X)] \simeq \frac{1}{n}\sum_{i=1}^{n} f(X_i) \tag{2}$$

と近似知的に評価する。従って $f(X_i)$ **の全体平均 (population mean) はサンプル平均で評価される。サンプル** $\{X_t\}$ **が独立のとき、大数の法則 (low of large numbers) により、サンプル数** n **を大きくするほど近似の精度は良くなることが保証される。**ここで n は解析する人が大きさを決めることが出来る数でありデータ数では無い。

一般に、事後確率分布 $\Pr(X|D)$ から独立にサンプル点 $\{X_t\}$ を取り出すことはほとんど不可能である。しかしながら $\{X_t\}$ は必ずしも独立である必要は無い。$\{X_t\}$ は適当な特徴を持った $\Pr(X|D)$ の台 (support) からサンプルを取り出すという過程によって生成される。これを行う一つの方法がマルコフ連鎖を用いるものである。

2.2 マルコフ連鎖

時刻 t（ステップ）$t \geq 0$ に於いて、連鎖の現在の状態 X_t だけに依存する分布 $\Pr(X_{t+1}|X_t)$ から次の状態 X_{t+1} をサンプルして、ランダム変数の列 $\{X_0, X_1, X_2, \ldots\}$ を生成することを考える。つまり、X_t を決めると次の状態 X_{t+1} はそれより過去の連鎖 $\{X_0, X_1, \ldots, X_{t-1}\}$ には依存しないとする。この列（連鎖）をマルコフ連鎖という。ここでは連鎖は時間的に一様で t に依らないとする。

時間的に一様なマルコフ連鎖は単一の定常分布 (stationary distribution) に収束することが示される（証明略）。この定常分布を $\pi(X)$ と書く。**定常分布** $\pi(X)$ **が正確に事後確率分布** $\Pr(X|D)$ **になるような操作を行うのが MCMC 法の目的**である。

t が増加するにつれてサンプル点は定常分布からサンプルされるようになり、初期条件の情報は失われる。つまり**定常分布は初期状態には依らない**ことになる。

2.3 Output analysis

“burn-in” と呼ばれる十分長い m 回の反復 (iterations) を行った後では、サンプル点 $\{X_t;\ t = m+1, m+2, \ldots, n\}$ は近似的に定常分布 $\pi(X)$ からのサンプルであると見なせる。ここでようやくマルコフ連鎖出力を用いて、成分 X に対する興味のある値を計算することが出来る。つまり、期待値 $E[f(X)]$ を推定することが出来る。収束前のサンプル (burn-in samples) はこの期待値の計算には使わず、推定値は

$$\bar{f} = \frac{1}{n-m}\sum_{t=m+1}^{n} f(X_t) \tag{3}$$

となる。これをエルゴード平均 (ergodic average) と呼ぶ。これを用いれば平均、標準偏差、相関、信頼区間などは全てモンテカルロ出力 $\{X_t;\ t = m+1, m+2, \ldots, n\}$ のサンプル平均、標準偏差、相関、信頼区間として求めることが出来る。例えば、X の marginal mean、marginal variance は以下の通りである。

$$\bar{X} = \frac{1}{n-m}\sum_{t=m+1}^{n} X_t \tag{4}$$

$$\sigma_X^2 = \frac{1}{n-m-1}\sum_{t=m+1}^{n} \left(X_t - \bar{X}\right)^2 \tag{5}$$

2.4 Metropolis-Hastings アルゴリズム

期待値 $E[f(X)]$ を求めるには、**定常分布が正確に事後確率分布 $\Pr(X|D)$ になるようにマルコフ連鎖を組み立てる方法を見つける必要**がある。この一つが Metropolis-Hastings アルゴリズムである。

Metropolis-Hastings アルゴリズムでは、まず時刻 t から $t+1$ の次の状態 X_{t+1} を決める為に、提案分布 (proposal distribution) から候補点 Y をサンプルする。ここで提案分布は現在の状態 X_t に依存してもしなくても良い。候補の点 Y はある確率 $\alpha(X_t, Y)$ で採択され、$X_{t+1} = Y$ となる。Y が採択されなかった場合 $X_{t+1} = X_t$ となる。確率 $\alpha(X_t, Y)$ は、

$$\alpha(X_t, Y) \equiv \min\left(1, \frac{\Pr(Y|D)q(X|Y)}{\Pr(X|D)q(Y|X)}\right) \tag{6}$$

で定義される。

従って Metropolis-Hastings アルゴリズムは

1. 初期状態 X_0 を決める。
2. 提案分布 $q(Y|X_t)$ から候補点 Y をサンプルする。
3. 疑似乱数 $U[0:+1]$ から一つの U をサンプルする。
4. - $U \leq \alpha(X_t, Y)$ なら $X_{t+1} = Y$ とする。
   - $U > \alpha(X_t, Y)$ なら $X_{t+1} = X_t$ とする。
5. t を一つ増やし、2,3,4 を n 回繰り返す。

ということをすればよい。**驚くべきことは提案分布はどんな形であってもいい**のである（証明略）。

しかし提案分布を選ぶとき、そのスケールを慎重に選ぶ必要がある。$Y - X_t$ が小さいような提案分布では一般に採択率が高くなるが、それにも関わらずゆっくりにしか収束しない。一方 $Y - X_t$ が大きくするような大胆な提案分布では、定常分布の中心部から裾の方へ動く様な（確率の低い）提案が多くなり、採択率は低くなる。そのような連鎖はあまり動かず、収束が遅くなる。理想的には提案分布はこれらの極端な二つの場合を避けるように選ぶべきである。

2.5 連鎖の数

今まで連鎖を一つだけ走らせることを考えてきたが、**複数の連鎖を走らせても構わない**。様々な文献で進めているやり方には矛盾があって、例えば多くの短い連鎖、幾つかの長い連鎖、非常に長い一つの連鎖を勧めているものがある。しかし $\pi(X)$ から独立したサンプルを得ることを期待して、たくさんの短い連鎖走らせることは、独立なサンプルを必要とする様な特別な理由がなければ、見当はずれであるということで意見は一致しつつある。**確かに独立なサンプルを得ることはエルゴード平均を求める際の条件になっていない**。幾つかの長い連鎖を実行するという学派と、一つの非常に長い連鎖を実行することを勧める学派との間の議論が続いている。

2.6 初期状態（初期値）

先に述べたようにマルコフ連鎖はどんな初期条件から始めても、定常分布に収束することが予想される。従ってすぐに収束するような連鎖では、極端な初期条件から始めても構わないが、収束が遅い場合には長い “burn-in” 時間を割けるために、慎重に初期条件を選ぶ必要がある。しかし**多くの場合は初期条件を選ぶのにあまり注意を払う必要な無い**。

3 実行

3.1 設定

MCMC 法でパラメータの推定を行うために、期待値 $\mu = 1.2$、分散 $\sigma^2 = 1.4$ の正規分布に、平均零、標準偏差 0.01 の正規乱数を加えたデータを用意した（図 1）。データの作成には Mathematica で

```
Needs["Statistics‘ContinuousDistributions‘"]
RandomNormal[ mu_, sigma_] := Random[NormalDistribution[mu, sigma]]
```

の様にした。

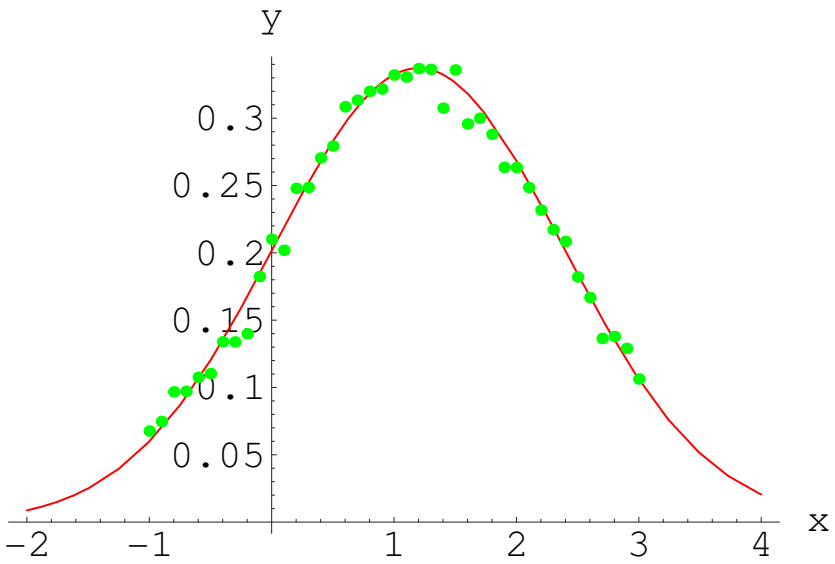

図 1　期待値 $\mu = 1.2$、分散 $\sigma^2 = 1.4$ の正規分布。

正規分布は

$$N(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \tag{7}$$

で書ける。データ点は $x = -1 \sim +3$ で刻み幅 0.1、総データ数 41 点である。データ点の座標を $x_i; i = 1, 2, \ldots, 41$ で表している。実際のデータ（乱数含む）を y_i とすると、

$$y_i = N(x_i, 1.2, 1.4) + (\text{平均零、標準偏差 } 0.01 \text{ に従う正規乱数})_i \tag{8}$$

と書ける。データ点 y_i をベクトルの形 $\mathbf{D}$ で表すと、$\mathbf{D} = (y_1, y_2, \ldots, y_{41})$ である。このデータ y_i からもとの分布 $N(x, \mu, \sigma^2)$ を推定するのが、今回の目的である。データ点は荒井修論と比較できるように同じ点を採用した。ただ誤差は乱数を用いているため必ずしも同じものではない。

3.2 提案分布及び採択確率

今求めたいものは期待値 μ、分散 σ^2 である。MCMC 法で求められる値を X_t で書くと、今の場合 $X_t = (\mu_t, \sigma_t^2)$ と書ける。今回は採択の基準となる確率 U、提案分布 $q(Y|X)$ として、一様乱数 $U[0:+1], U[-1:+1]$ を使用した。$U[0:+1], U[-1:+1]$ は以下の関数によって作成される。

- $U[0:+1]$

```
double GetRandom(){
  return (double)rand()/(RAND_MAX+1.) ;
}
```

- $U[-1:+1]$

```
double GetRandom(){
  return 2.0*(double)rand()/(RAND_MAX+1.) -1.0 ;
}
```

より「良い」乱数を作成する為に乱数の “種” を

```
srand(time(NULL)) ; // 乱数の発生系列を変更する。
```

で作成している。

今の場合は提案分布は現在の状態 X_t に全く依存せず

$$q(Y|X) = q(X|Y) \tag{9}$$

である。具体的には

$$X_{t+1} = X_t + 0.03 \times U[-1:+1] \tag{10}$$

つまり、

$$\left(\mu_{t+1},\, \sigma^2_{t+1}\right) = \left(\mu_t + 0.03 \times U[-1:+1],\, \sigma^2_t + 0.03 \times U[-1:+1]\right) \tag{11}$$

である。

誤差が平均零、標準偏差 0.01 に従うので、$\mathbf{D}$ の確率つまり尤度関数 $\mathcal{L}(X) = \Pr(\mathbf{D}|X)$ は

$$\mathcal{L}(X_t) = \Pr(\mathbf{D}|X_t) = \frac{1}{(2\pi \cdot 0.01^2)^{41/2}} \exp\left[-\frac{1}{2}\sum_{i=1}^{41} \frac{\left\{y_i - N(x_i, \mu_t, \sigma^2_t)\right\}^2}{0.01^2}\right] \tag{12}$$

である。また $\Pr(X_t)$ は事前に何の情報もないことから、一様密度を仮定して

$$\mathrm{P}(X_t) = \mathrm{Cosnt.} \tag{13}$$

とする。Eq.(12),(1),(13) より、

$$\Pr(X_t|\mathbf{D}) \propto \exp\left[-\frac{1}{2}\sum_{i=1}^{41} \frac{\left\{y_i - N(x_i, \mu_t, \sigma^2_t)\right\}^2}{0.01^2}\right] \tag{14}$$

となる。よて Eq.(9),(14) より採択確率 $\alpha(X, Y)$ は以下のように書ける。

$$\begin{aligned}\alpha(X_t, Y) &= \min\left(1, \frac{\Pr(Y|D)q(X|Y)}{\Pr(X|D)q(Y|X)}\right) \\ &= \min\left(1, \exp\left[\frac{1}{2}\sum_{i=1}^{41} \frac{\left\{y_i - N(x_i, \mu_t, \sigma^2_t)\right\}^2 - \left\{y_i - N(x_i, \mu_{t+1}, \sigma^2_{t+1})\right\}^2}{0.01^2}\right]\right)\end{aligned} \tag{15}$$

この exponential の中は、最小二乗法で計算していることになる。

4 結果

壱拾万試行のモンテカルロ出力の収束の様子を載せる（図 2)。これから自然と定常状態に達することが分かる。よって壱拾万試行後に定常分布（目的とする事後確率分布）に成ったと仮定する。この時の採択率は約 **46** ~ **47**% であった。

今回行った作業は以下の通りである。

1. **一回のモンテカルロ試行を用いた場合。**以下で拾ステップ毎にデータをサンプリングしているのは、データ毎の相関が少なくなるようにする為である。拾ステップ経てば、前回出力された情報と完全に独立であると仮定している。
   - 定常分布から拾ステップ間隔を置いて 500 データをサンプリングした分布。図,3,4,5。
   - 定常分布から拾ステップ間隔を置いて 1000 データをサンプリングした分布。図,6,7,8。
   - 定常分布から拾ステップ間隔を置いて 10000 データをサンプリングした分布。図,9,10,11。
2. **複数回のモンテカルロ試行を用いた場合。**このとき初期条件は動的に変化させている。モンテカルロ試行一回当たりは、壱拾万試行後の五百データをサンプルとしている。
   - 五百データを平均し、壱千回モンテカルロ試行を行う。(一回のモンテカルロ試行当たりのデータは壱個。全データ 1×1000 個。)。図 12,13,14。
   - 五百データ、壱千回モンテカルロ試行を行う。(一回のモンテカルロ試行当たりのデータは五百個。全データ 500×1000 個。)。図 15,16,17。

以上より五百データを平均した場合以外は、誤差の範囲内（1σ の範囲）でデータの値と一致していることが分かる。つまり誤差の範囲内で事後確率分布 $\Pr(X|\mathbf{D})$ を推定することができた。五百データを平均した場合は、各試行毎に同じ様な定常分布から平均を求めているので、当然平均の値は同じ様な値になり（定常分布は初期条件によらないので)、分散も極めて小さくなる。これはモンテカルロ出力を正しく扱っておらず、これで得られた分布は正しい事後確率分布を与えない。

5 結論

今回の場合、結果よりモンテカルロ試行は複数回行う必要はなく、一回のモンテカルロ試行の、十分独立と思える定常分布から適当に（出来れば数は多い方が良い）データをサンプリングし、これから分布を求めるのが一番効率が良いと考えられる。複数回のモンテカルロ試行を用いた場合も、確かに正しい分布を与えるが、計算時間がかかる上に、定常分布が初期値に依存しないと言うモンテカルロ連鎖の利点を生かしていない点で、無駄な計算をしていることになる。ただこれが必要ない作業か？、と考えた場合良く分からない。

6 Program source

6.1 定常分布に収束する様子をプロットするプログラム

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<math.h>
4  #include<time.h>
5  #define PI 3.141592654
6
7  double GetRandom(void) ; // -1 から 1 までの疑似乱数を生成する関数。
8  double GetRandom01(void) ; // -1 から 1 までの疑似乱数を生成する関数。
```

```
double Normal( double, double, double) ;

int main(void){

  srand(time(NULL)) ; // 乱数の発生系列を変更する。

  int i, n;
  double x_i[41] ; // 変数
  double y_i[41] ; // 実際のデータ。エラー含む。

  double U ;
  double mu_t, mu_tmp ; // 平均
  double sigma_t2, sigma2_tmp ; // 分散

  double yNXt, yNY ;

  double alpha ; // 採択確率
  int N ; // 採択数

  /// 入力データの為のファイルの open 及びもとデータの配列への代入 //////////////////////////
  FILE *fiptr ;
  fiptr = fopen("Random.dat","r+") ;
  if( fiptr==NULL ) printf("failed at fopen\n") ;
  else{
    for( i=0; i<41; i++) fscanf( fiptr, "%lf%lf", &x_i[i], &y_i[i]) ;
  }
  fclose(fiptr) ;
  /// 出力ファイルの open ////////////////////////////////////////////////////////////////
  FILE *foptr ;
  foptr = fopen("MCMC_plot3.dat","w+") ;
  if( foptr==NULL ) printf("failed at fopen\n") ;
  ////////////////////////////////////////////////////////////////////////

  // 初期値 //////////////////////////
  mu_t = 2.0 ;
  sigma_t2 = 2.1 ;
  fprintf( foptr, "%lf   %lf\n", mu_t,sigma_t2) ;
  N = 0 ;

  for(n=0; n<10000;n++){
    // 提案分布による値の変更 /////////
    U = GetRandom() ;
    mu_tmp = mu_t + 0.03*U ;
    U = GetRandom() ;
    sigma2_tmp = sigma_t2 + 0.03*U ;

    // 尤度関数 ///////////////////////
    yNXt = 0. ;
    for(i=0;i<41;i++) yNXt = yNXt + pow( y_i[i]-Normal(x_i[i],mu_t,sigma_t2), 2 ) ;
    //
    yNY = 0. ;
    for(i=0;i<41;i++) yNY = yNY + pow( y_i[i]-Normal(x_i[i],mu_tmp,sigma2_tmp), 2 ) ;

    // 採択確率
    alpha = exp(  ( yNXt-yNY )/(2.*pow(0.01,2)) ) ;
    if( alpha >=1.0 ) alpha = 1.0 ;

    // 値の更新
    U = GetRandom01() ;
    if( U<=alpha ){
      mu_t = mu_tmp ;
      sigma_t2 = sigma2_tmp ;
      N = N + 1 ;
    }
    fprintf( foptr, "%lf   %lf\n", mu_t,sigma_t2) ;

  }
  fclose(foptr) ;

  printf("採択率 %lf\n", ((double)N)/((double)n) ) ;

  return 0 ;
}

double GetRandom(){
  return 2.0*(double)rand()/(RAND_MAX+1.) -1.0 ;
}

double GetRandom01(){
  return (double)rand()/(RAND_MAX+1.) ;
}

double Normal( double x, double mu, double sigma2){
  return 1./sqrt(2.*PI*sigma2)*exp( -pow((x-mu),2.)/(2.*sigma2) ) ;
}
```

6.2 定常分布に達した後のデータを出力するプログラム

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#define PI 3.141592654
#define statinary 100000

double GetRandom(void) ; // -1 から 1 までの疑似乱数を生成する関数。
double GetRandom01(void) ; // 0 から 1 までの疑似乱数を生成する関数。
double Normal( double, double, double) ; // 正規分布

int main(void){

  int i, n, j;
  double x_i[41] ; // 変数
  double y_i[41] ; // 実際のデータ。エラー含む。

```

```
   double U ; // U[-1:+1]
   double mu_t, mu_tmp ; // 平均
   double sigma_t2, sigma2_tmp ; // 分散

   double yNXt, yNY ; //尤度関数を計算するための値
   double alpha ; // 採択確率
   int N ; // 採択数

   double mu_t_sum,mu_t_h ; // 事後確率分布の期待値の平均値
   double sigma_t2_sum,sigma_t2_h ; // 事後確率分布の分散の平均値

   /// 入力データの為のファイルの open 及びもとデータの配列への代入 ////////////////////////
   FILE *fiptr ;
   fiptr = fopen("Random.dat","r+") ;
   if( fiptr==NULL ) printf("failed at fopen\n") ;
   else{
     for( i=0; i<41; i++) fscanf( fiptr, "%lf%lf", &x_i[i], &y_i[i]) ;
   }
   fclose(fiptr) ;
   /// 出力ファイルの open ////////////////////////////////////////////////////////////
   FILE *foptr2 ;
   foptr2 = fopen("MCMC_shikou_burn-in100000.dat","w+") ;
   if( foptr2==NULL ) printf("failed at fopen\n") ;
   ////////////////////////////////////////////////////////////////////////////////////

   // MCMC 法の実行 試行 4671 回 ////////////////////////////////////////////////////////
   srand(time(NULL)) ; // 乱数の発生系列を変更する。
   // 初期値 ////////////////////////
   U = GetRandom() ;
   mu_t = 2.0+U ;
   sigma_t2 = 2.1+U ;
   N = 0 ;
   mu_t_sum = 0. ;
   sigma_t2_sum = 0. ;
   j=0 ;

   for(n=0; n<100000+statinary*10;n++){
     // 提案分布による値の変更 /////////
     U = GetRandom() ;
     mu_tmp = mu_t + 0.03*U ;
     U = GetRandom() ;
     sigma2_tmp = sigma_t2 + 0.03*U ;

     // 尤度関数 ///////////////////////
     yNXt = 0. ;
     for(i=0;i<41;i++) yNXt = yNXt + pow( y_i[i]-Normal(x_i[i],mu_t,sigma_t2), 2 ) ;
     //
     yNY = 0. ;
     for(i=0;i<41;i++) yNY = yNY + pow( y_i[i]-Normal(x_i[i],mu_tmp,sigma2_tmp), 2 ) ;

     // 採択確率
     alpha = exp(  ( yNXt-yNY )/(2.*pow(0.01,2)) ) ;
     if( alpha >=1.0 ) alpha = 1.0 ;

     // 値の更新
     U = GetRandom01() ;
     if( U<=alpha ){
       mu_t = mu_tmp ;
       sigma_t2 = sigma2_tmp ;
       N = N + 1 ;
     }

     if(n>=100000){
       mu_t_sum = mu_t_sum + mu_t ;
       sigma_t2_sum = sigma_t2_sum + sigma_t2 ;
       j++ ;
       if( j%10==0) fprintf( foptr2, "%lf   %lf     %d\n", mu_t, sigma_t2, j/10) ;
     }
   }

   mu_t_h = mu_t_sum/j ;
   sigma_t2_h = sigma_t2_sum/j ;

   printf("mu= %lf\n", mu_t_h) ;
   printf("sigma^2= %lf\n", sigma_t2_h) ;
   printf("rate= %lf\n", ((double)N)/((double)n) ) ;

   fclose(foptr2) ;
   ////////////////////////////////////////////////////////////////////////////////////////////////

   return 0 ;
}

double GetRandom(){
   return 2.0*(double)rand()/(RAND_MAX+1.) -1.0 ;
}
double GetRandom01(){
   return (double)rand()/(RAND_MAX+1.) ;
}
double Normal( double x, double mu, double sigma2){
   return 1./sqrt(2.*PI*sigma2)*exp( -pow((x-mu),2.)/(2.*sigma2) ) ;
}
```

6.3 千回モンテカルロ試行を実行し、全てのデータ及び試行毎の平均を出力するプログラム

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#define PI 3.141592654
#define shikou 1000

```

```
double GetRandom(void) ; // -1 から 1 までの疑似乱数を生成する関数。
double GetRandom01(void) ; // 0 から 1 までの疑似乱数を生成する関数。
double Normal( double, double, double) ; // 正規分布

int main(void){

  int i, n, j, k;
  double x_i[41] ; // 変数
  double y_i[41] ; // 実際のデータ。エラー含む。

  double U ; // U[-1:+1]
  double mu_t, mu_tmp ; // 平均
  double sigma_t2, sigma2_tmp ; // 分散

  double yNXt, yNY ; //尤度関数を計算するための値
  double alpha ; // 採択確率
  int N ; // 採択数

  double mu_t_sum,mu_t_h ; // 事後確率分布の期待値の平均値
  double sigma_t2_sum,sigma_t2_h ; // 事後確率分布の分散の平均値

  /// 入力データの為のファイルの open 及びもとデータの配列への代入 //////////////////////////
  FILE *fiptr ;
  fiptr = fopen("Random.dat","r+") ;
  if( fiptr==NULL ) printf("failed at fopen\n") ;
  else{
    for( i=0; i<41; i++) fscanf( fiptr, "%lf%lf", &x_i[i], &y_i[i]) ;
  }
  fclose(fiptr) ;
  /// 出力ファイルの open ////////////////////////////////////////////////////////////////
  FILE *foptr1 ;
  foptr1 = fopen("MCMC_shikou_small.dat","w+") ;
  if( foptr1==NULL ) printf("failed at fopen\n") ;
  FILE *foptr2 ;
  foptr2 = fopen("MCMC_shikou_small_all.dat","w+") ;
  if( foptr2==NULL ) printf("failed at fopen\n") ;
  ////////////////////////////////////////////////////////////////////////////////////////

  // MCMC 法の実行 試行 4671 回 ////////////////////////////////////////////////////////
  for(k=0; k<shikou; k++){
    srand(time(NULL)) ; // 乱数の発生系列を変更する。
    // 初期値 ////////////////////////
    U = GetRandom() ;
    mu_t = 2.0+U ;
    sigma_t2 = 2.1+U ;
    N = 0 ;
    mu_t_sum = 0. ;
    sigma_t2_sum = 0. ;
    j=0 ;

    for(n=0; n<100500;n++){
      // 提案分布による値の変更 /////////
      U = GetRandom() ;
      mu_tmp = mu_t + 0.03*U ;
      U = GetRandom() ;
      sigma2_tmp = sigma_t2 + 0.03*U ;

      // 尤度関数 ///////////////////////
      yNXt = 0. ;
      for(i=0;i<41;i++) yNXt = yNXt + pow( y_i[i]-Normal(x_i[i],mu_t,sigma_t2), 2 ) ;
      //
      yNY = 0. ;
      for(i=0;i<41;i++) yNY = yNY + pow( y_i[i]-Normal(x_i[i],mu_tmp,sigma2_tmp), 2 ) ;

      // 採択確率
      alpha = exp(  ( yNXt-yNY )/(2.*pow(0.01,2)) ) ;
      if( alpha >=1.0 ) alpha = 1.0 ;

      // 値の更新
      U = GetRandom01() ;
      if( U<=alpha ){
        mu_t = mu_tmp ;
        sigma_t2 = sigma2_tmp ;
        N = N + 1 ;
      }

      if(n>=100000){
        mu_t_sum = mu_t_sum + mu_t ;
        sigma_t2_sum = sigma_t2_sum + sigma_t2 ;
        j++ ;
        fprintf( foptr2, "%lf   %lf     %d   %d\n", mu_t, sigma_t2, j, k+1) ;
      }
    }

    mu_t_h = mu_t_sum/j ;
    sigma_t2_h = sigma_t2_sum/j ;

    printf("%d/%d\n", k+1, shikou) ;
    printf("mu= %lf\n", mu_t_h) ;
    printf("sigma^2= %lf\n", sigma_t2_h) ;
    printf("rate= %lf\n", ((double)N)/((double)n) ) ;
    fprintf( foptr1, "%lf   %lf   %lf      %d\n", mu_t_h, sigma_t2_h, ((double)N)/((double)n), k+1) ;
  }
  fclose(foptr1) ;
  fclose(foptr2) ;
  ////////////////////////////////////////////////////////////////////////////////////////////////////

  return 0 ;
}

double GetRandom(){
  return 2.0*(double)rand()/(RAND_MAX+1.) -1.0 ;
}
double GetRandom01(){
  return (double)rand()/(RAND_MAX+1.) ;
}
double Normal( double x, double mu, double sigma2){
  return 1./sqrt(2.*PI*sigma2)*exp( -pow((x-mu),2.)/(2.*sigma2) ) ;
}
```

6.4 モンテカルロ出力を解析するプログラム

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

#define shikou 1000

int main(void){

  int i, j, N ;

  double mu[shikou] ; // MCMC で得られた期待値データ。
  double sigma2[shikou] ; // MCMC で得られた分散データ。
  double saitaku ; // MCMC で得られた採択率データ。
  double iteration ; // MCMC で得られた試行数データ。

  double sum_mu, sum_sigma2 ;
  double mu_h, sigma2_h ;

  double sum_mu_s, sum_sigma2_s ;
  double mu_s_h, sigma2_s_h ;

  /// 入力データの為のファイルの open 及びもとデータの配列への代入 //////////////////////////
  FILE *fiptr ;
  fiptr = fopen("MCMC_shikou_small.dat","r+") ;
  if( fiptr==NULL ) printf("failed at fopen\n") ;
  else{
    for( i=0; i<shikou; i++) fscanf( fiptr, "%lf%lf%lf%lf", &mu[i], &sigma2[i], &saitaku, &iteration) ;
  }
  fclose(fiptr) ;
  /// 出力ファイルの open ////////////////////////////////////////////////////////////////
  FILE *foptr1 ;
  foptr1 = fopen("MCMC_shikou_mu_dist_small.dat","w+") ;
  if( foptr1==NULL ) printf("failed at fopen\n") ;
  FILE *foptr2 ;
  foptr2 = fopen("MCMC_shikou_sigma2_dist_small.dat","w+") ;
  if( foptr2==NULL ) printf("failed at fopen\n") ;
  ////////////////////////////////////////////////////////////////////////////////////////

  sum_mu =0. ;
  sum_sigma2 = 0. ;
  for(i=0;i<shikou;i++){
    sum_mu = sum_mu + mu[i] ;
    sum_sigma2 = sum_sigma2 + sigma2[i] ;
  }
  mu_h = sum_mu/shikou ;
  sigma2_h = sum_sigma2/shikou ;

  sum_mu_s =0. ;
  sum_sigma2_s = 0. ;
  for(i=0;i<shikou;i++){
    sum_mu_s = sum_mu_s + pow( mu[i] -mu_h, 2.) ;
    sum_sigma2_s = sum_sigma2_s + pow( sigma2[i] -sigma2_h, 2.) ;
  }
  mu_s_h = sum_mu_s/shikou ;
  sigma2_s_h = sum_sigma2_s/shikou ;

  printf("<mu>=%lf\n", mu_h) ;
  printf("<sigma^2>=%lf\n", sigma2_h) ;
  printf("sigma_mu^2=%lf\n", mu_s_h) ;
  printf("sigma_sigma^2=%lf\n", sigma2_s_h) ;

  for( j=0; 1.189+0.0001*(j+1)<=1.198; j++){
    N = 0 ;
    for(i=0;i<shikou;i++){
      if( mu[i] >= 1.189+0.0001*j && mu[i] < 1.189+0.0001*(j+1) ){
        N++ ;
      }
    }
    fprintf( foptr1,"%lf   %lf\n", (( 1.189+0.0001*j )+( 1.189+0.0001*(j+1) ))/2, ((double)N)/(shikou/10000.0) ) ;
  }

  for( j=0; 1.390+0.0002*(j+1)<=1.425; j++){
    N = 0 ;
    for(i=0;i<shikou;i++){
      if( sigma2[i] >= 1.390+0.0002*j && sigma2[i] < 1.390+0.0002*(j+1) ){
        N++ ;
      }
    }
    fprintf( foptr2,"%lf   %lf\n", (( 1.390+0.0002*j )+( 1.390+0.0002*(j+1) ))/2, ((double)N)/(2.0*shikou/10000.0) ) ;
  }

  fclose(foptr1) ;
  fclose(foptr2) ;
  ////////////////////////////////////////////////////////////////////////////////////////////////

  return 0 ;
}
```

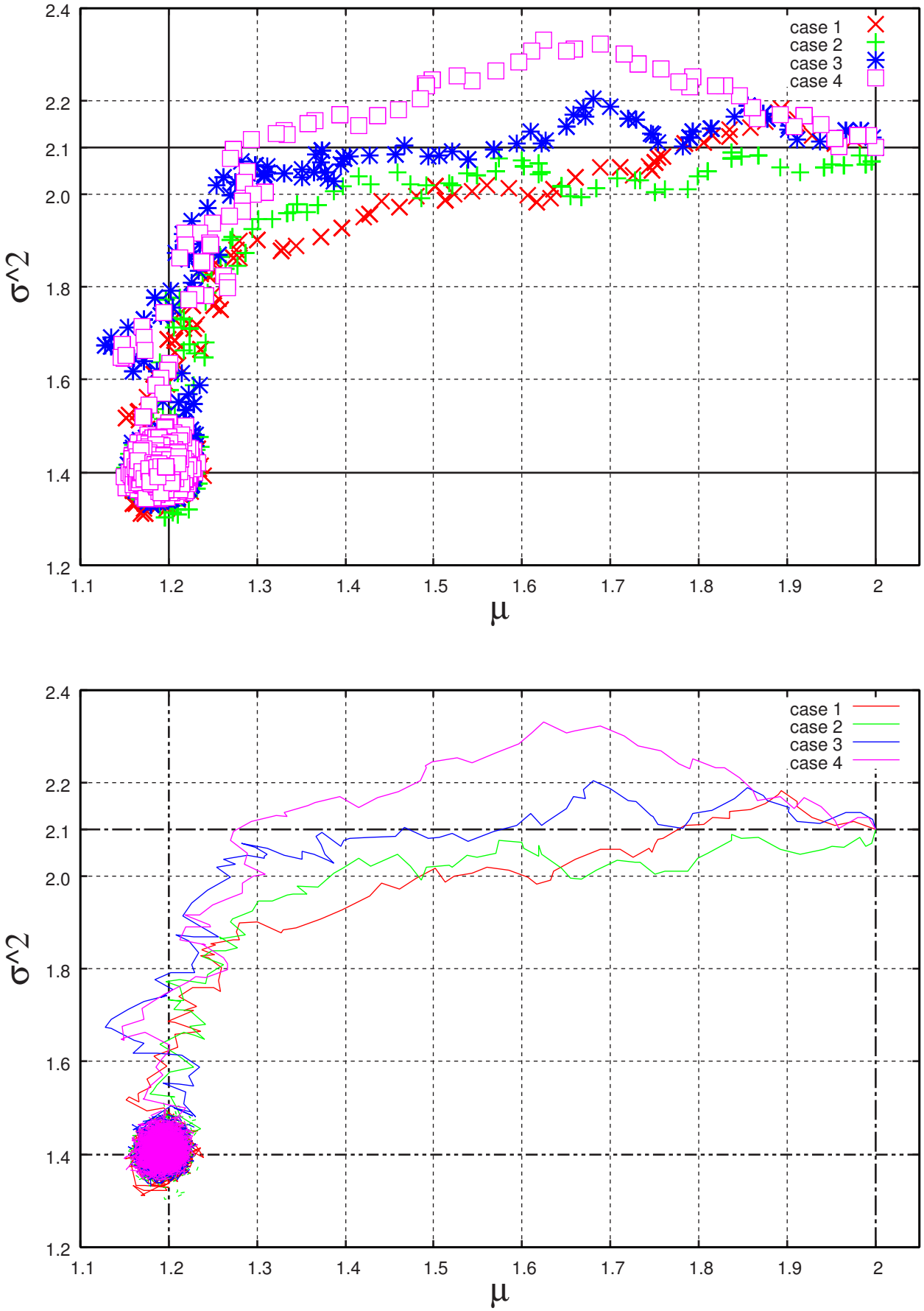

図 2　初期値 $X_0 = (2.0, 2.1)$ から出発した場合のモンテカルロ試行の例。

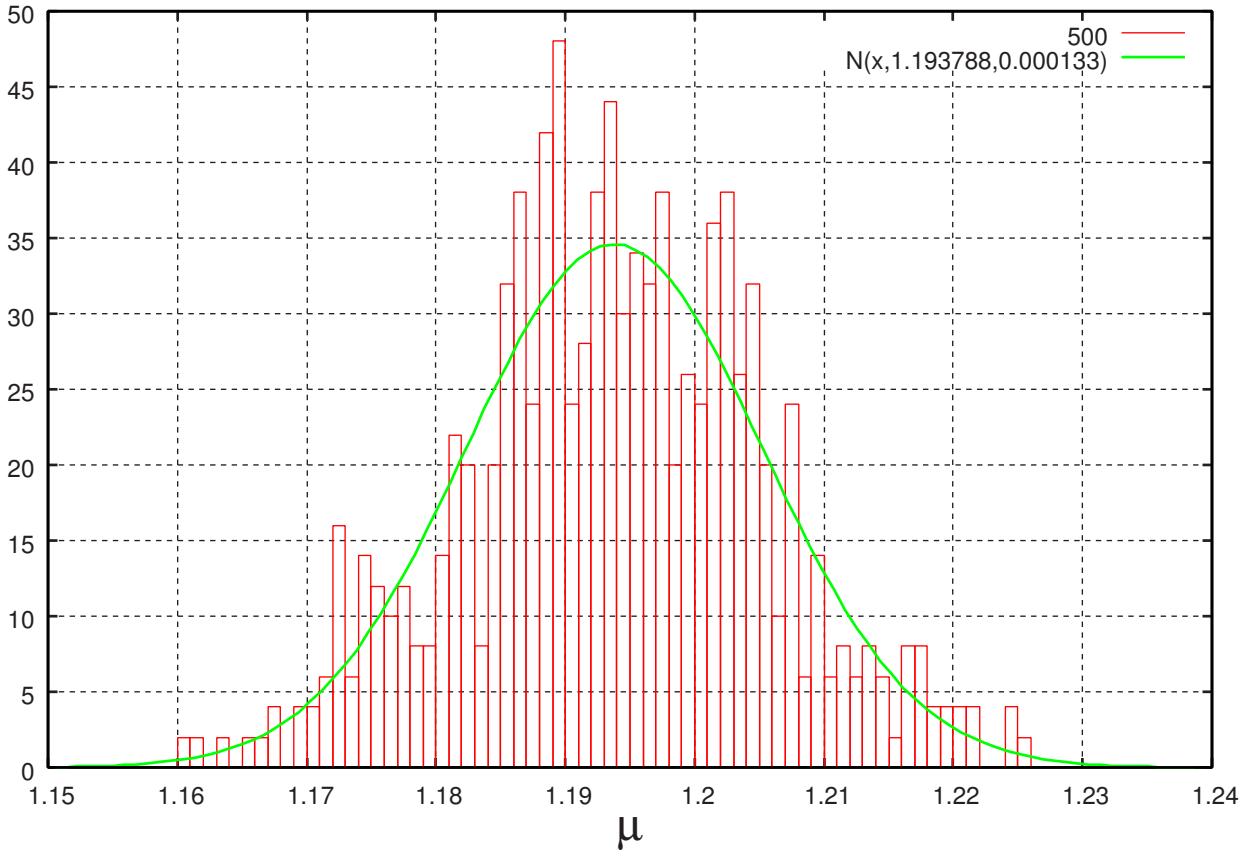

図 3　壱拾万試行後、拾ステップ毎に 500 データサンプリングした μ の分布。$N(x, 1.193788, 0.000133)$

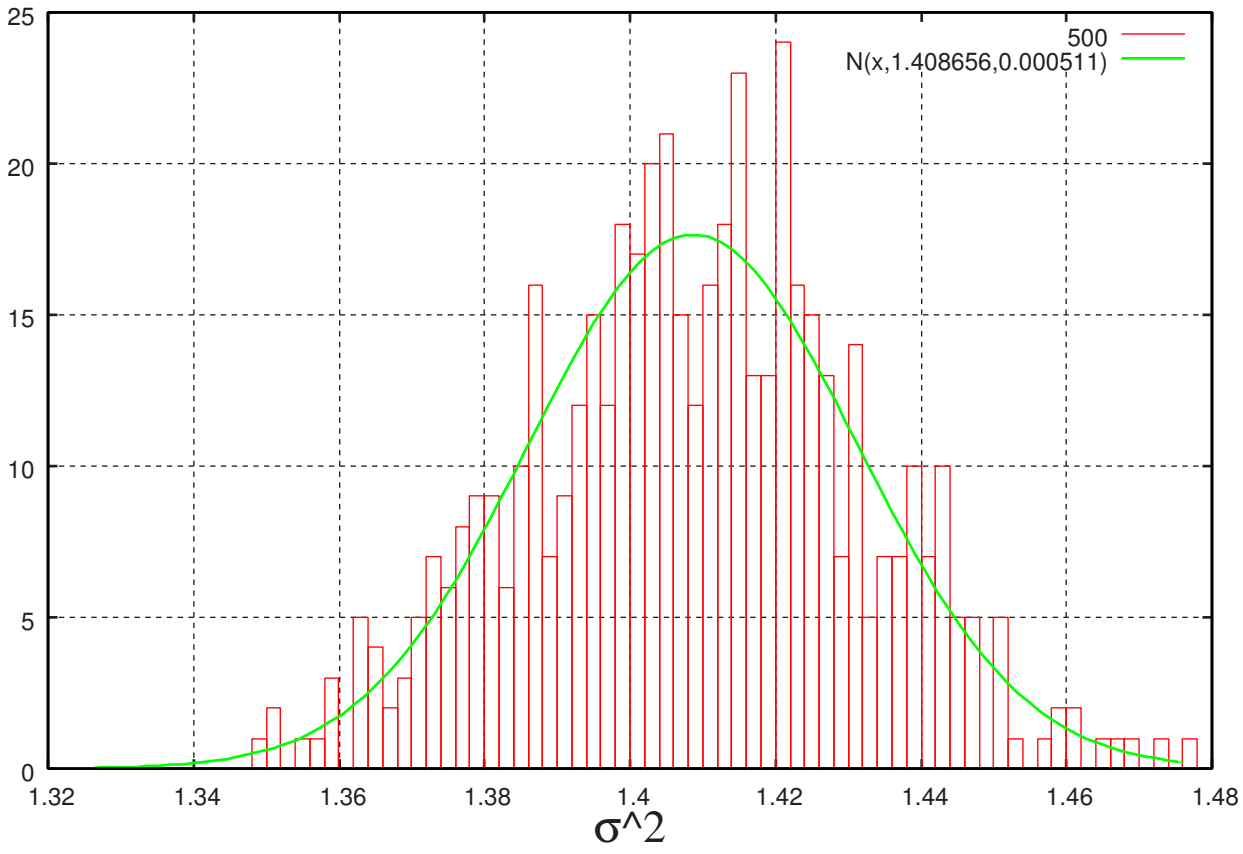

図 4　壱拾万試行後、拾ステップ毎に 500 データサンプリングした σ^2 の分布。$N(x, 1.408656, 0.000511)$

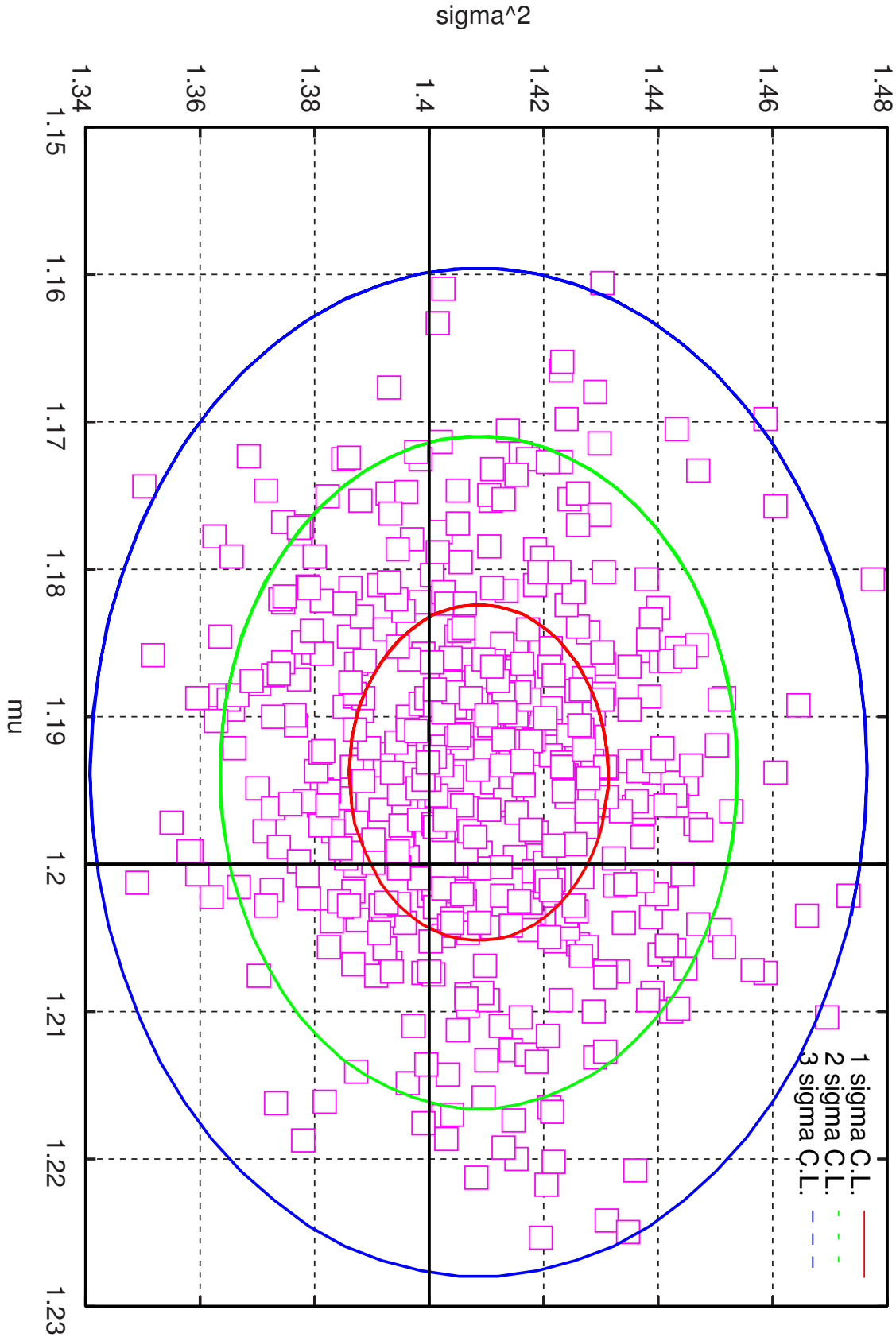

図 5　壱拾万試行後、拾ステップ毎に 500 データサンプリングした μ, σ^2 の分布

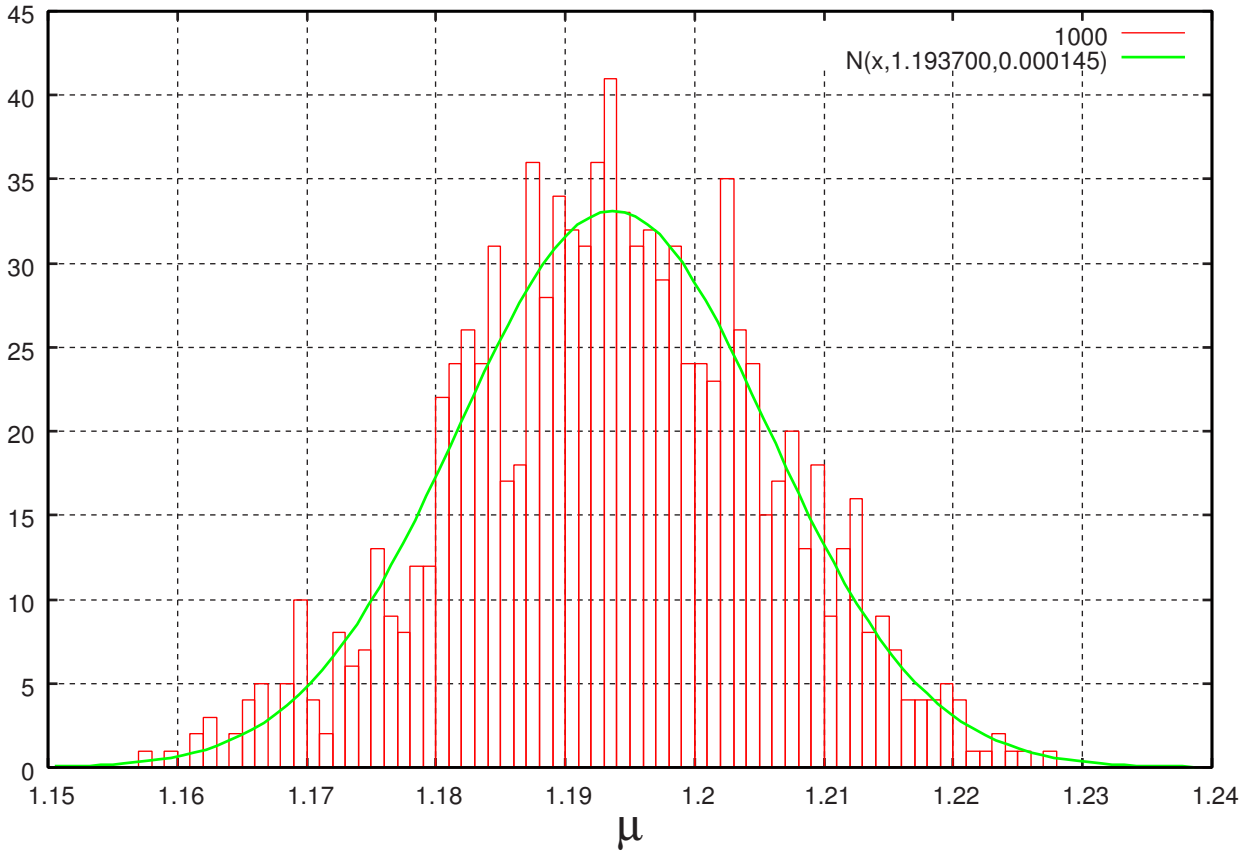

図 6　壱拾万試行後、拾ステップ毎に 1000 データサンプリングした μ の分布。$N(x, 1.193700, 0.000145)$

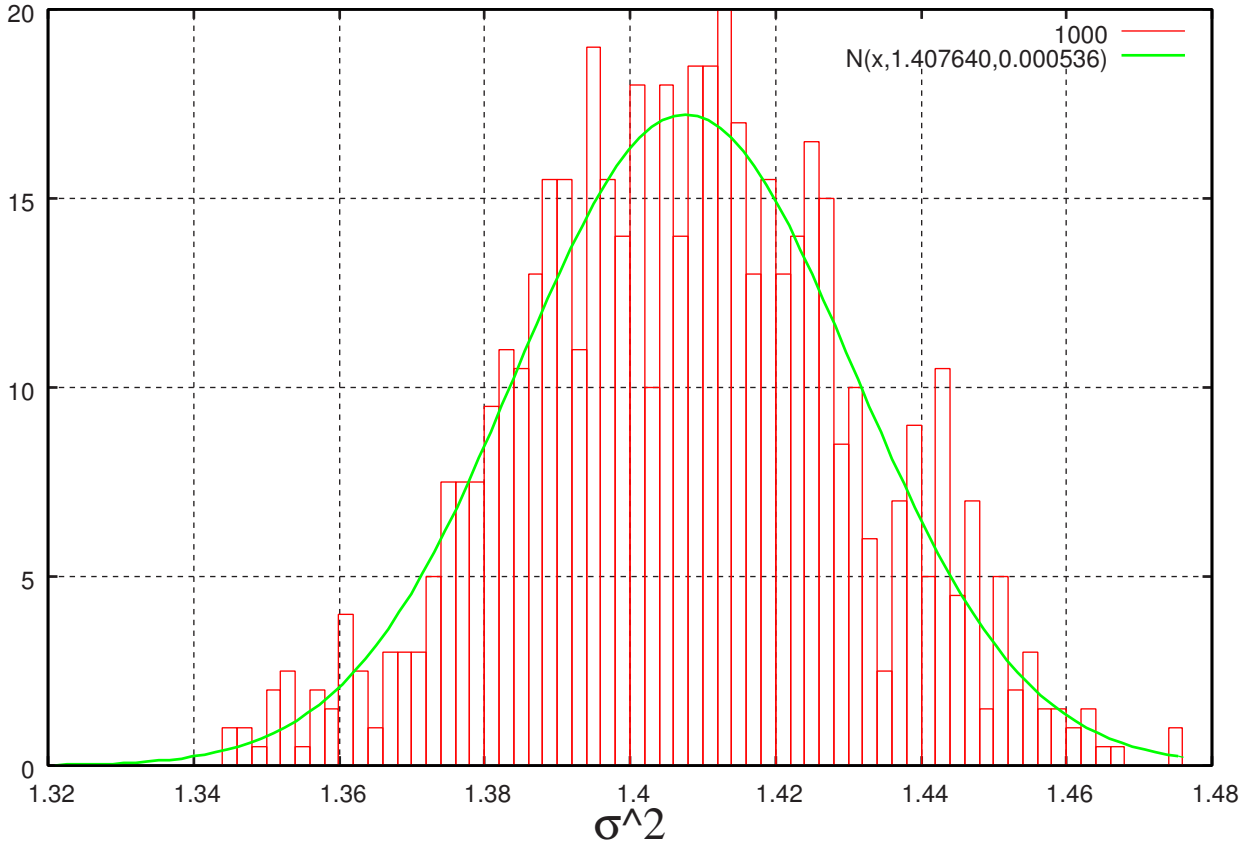

図 7　壱拾万試行後、拾ステップ毎に 1000 データサンプリングした σ^2 の分布。$N(x, 1.407640, 0.000536)$

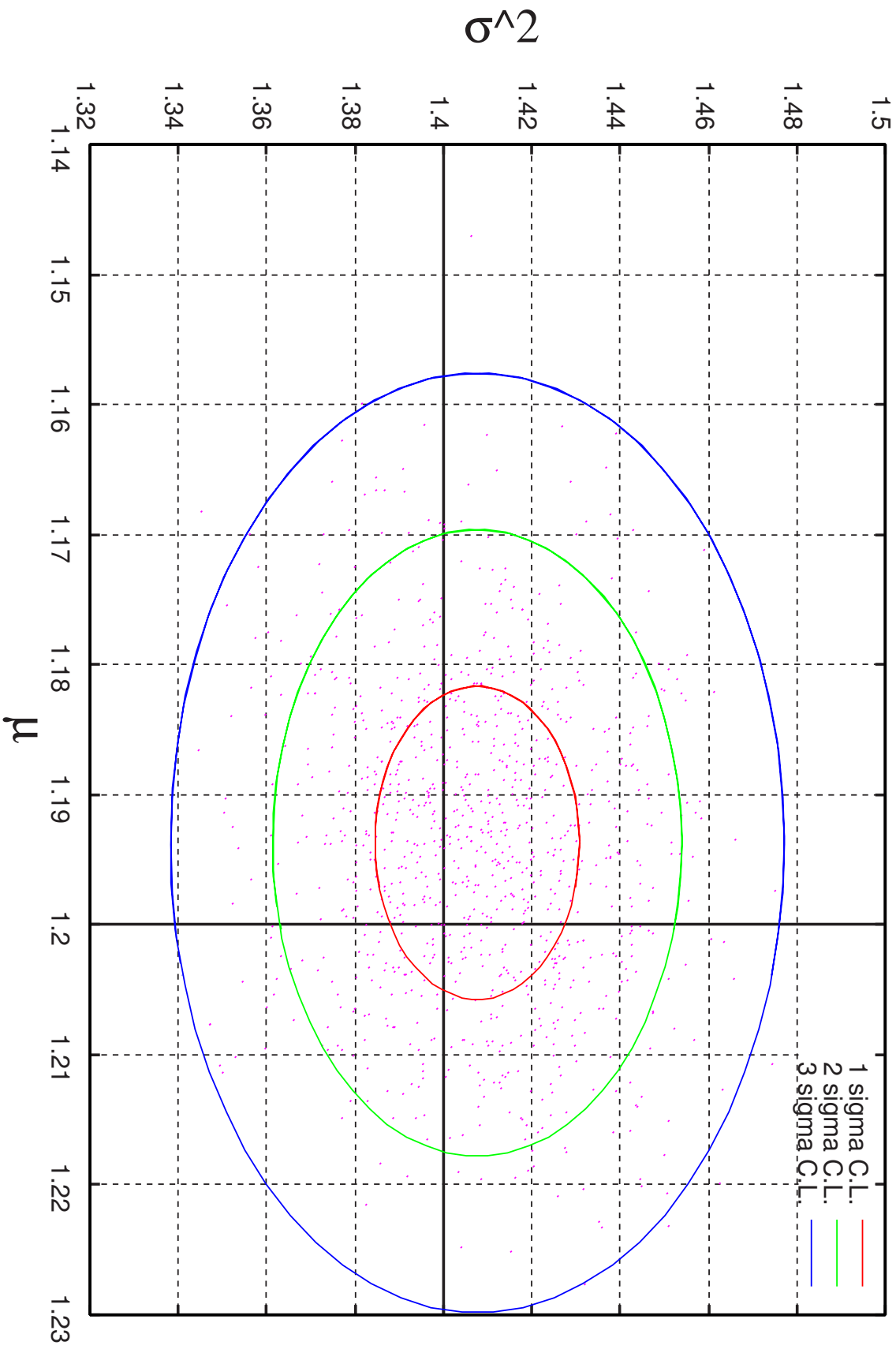

図 8　壱拾万試行後、拾ステップ毎に 1000 データサンプリングした μ, σ^2 の分布

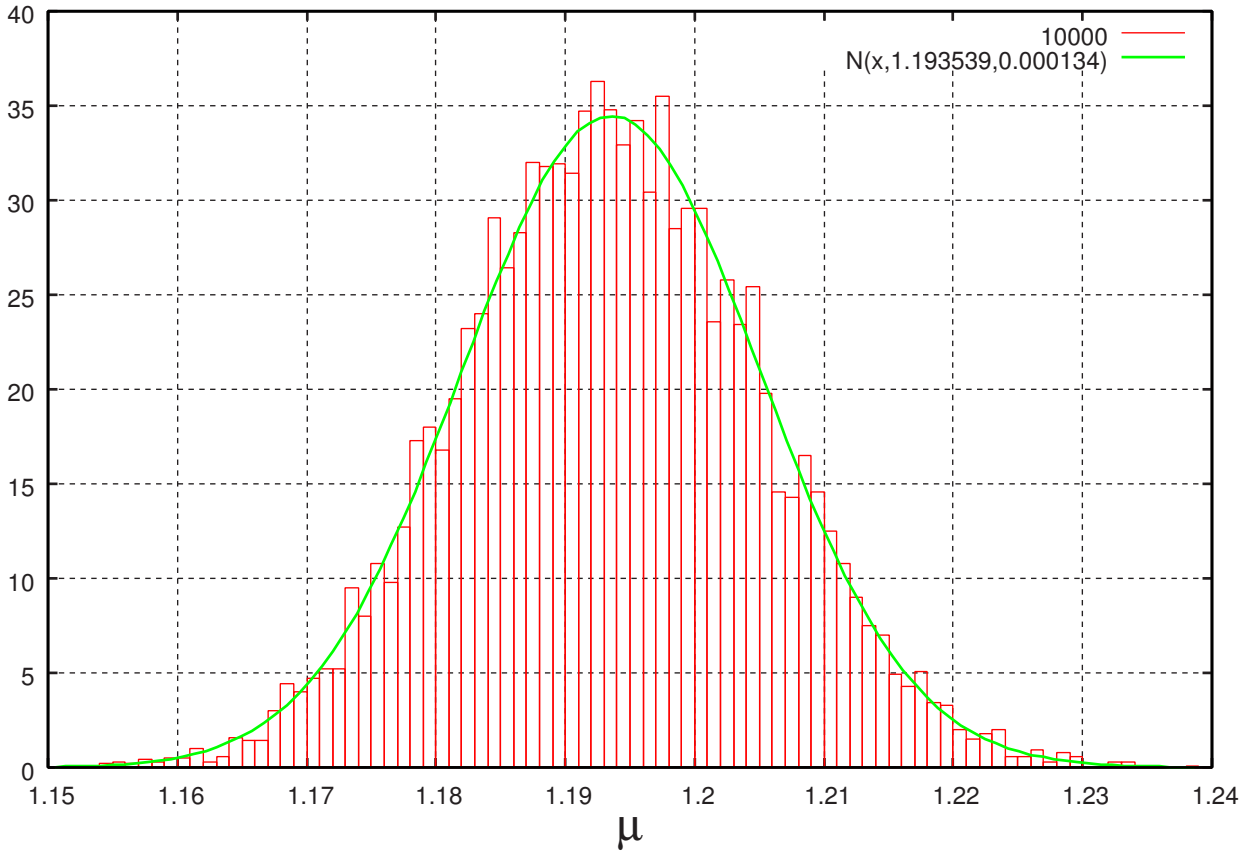

図 9　壱拾万試行後、拾ステップ毎に 10000 データサンプリングした μ の分布。$N(x, 1.193539, 0.000134)$

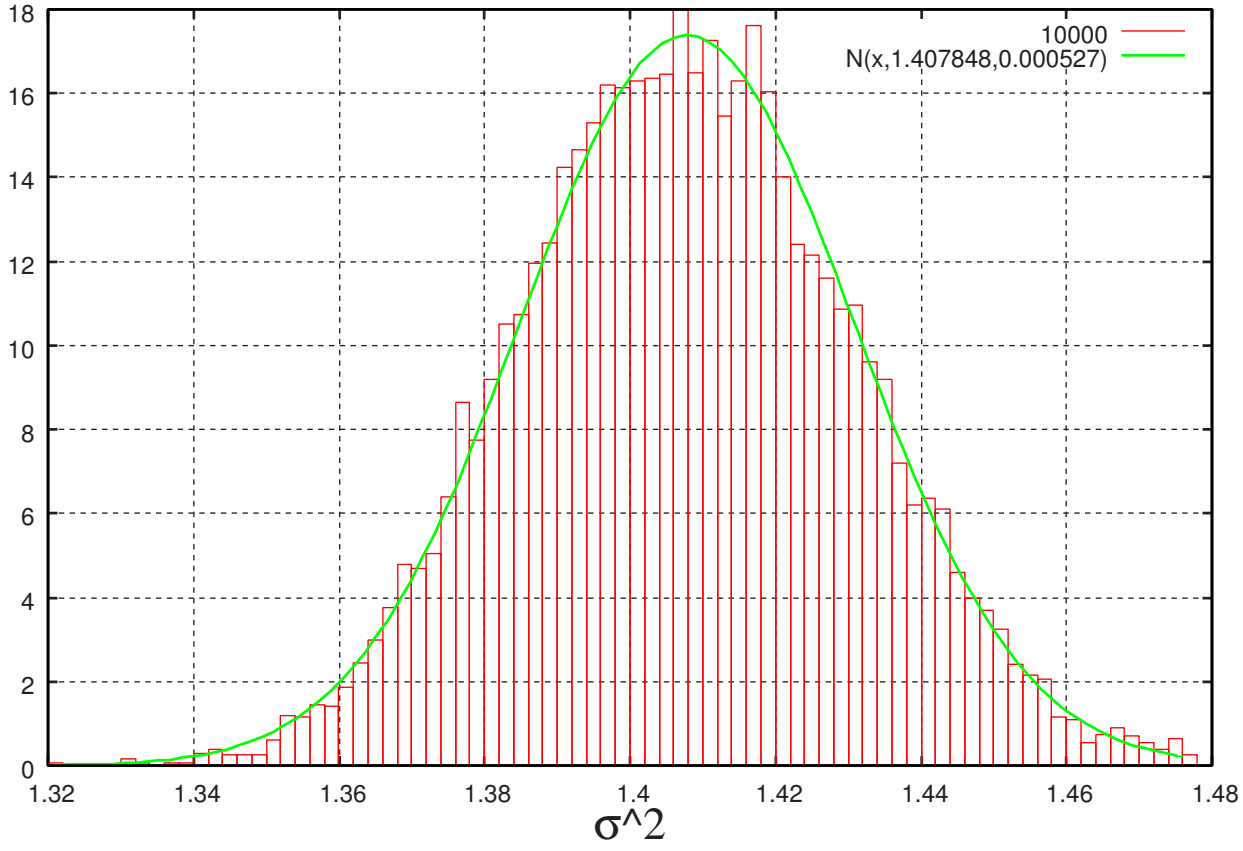

図 10　壱拾万試行後、拾ステップ毎に 10000 データサンプリングした σ^2 の分布。$N(x, 1.407848, 0.000527)$

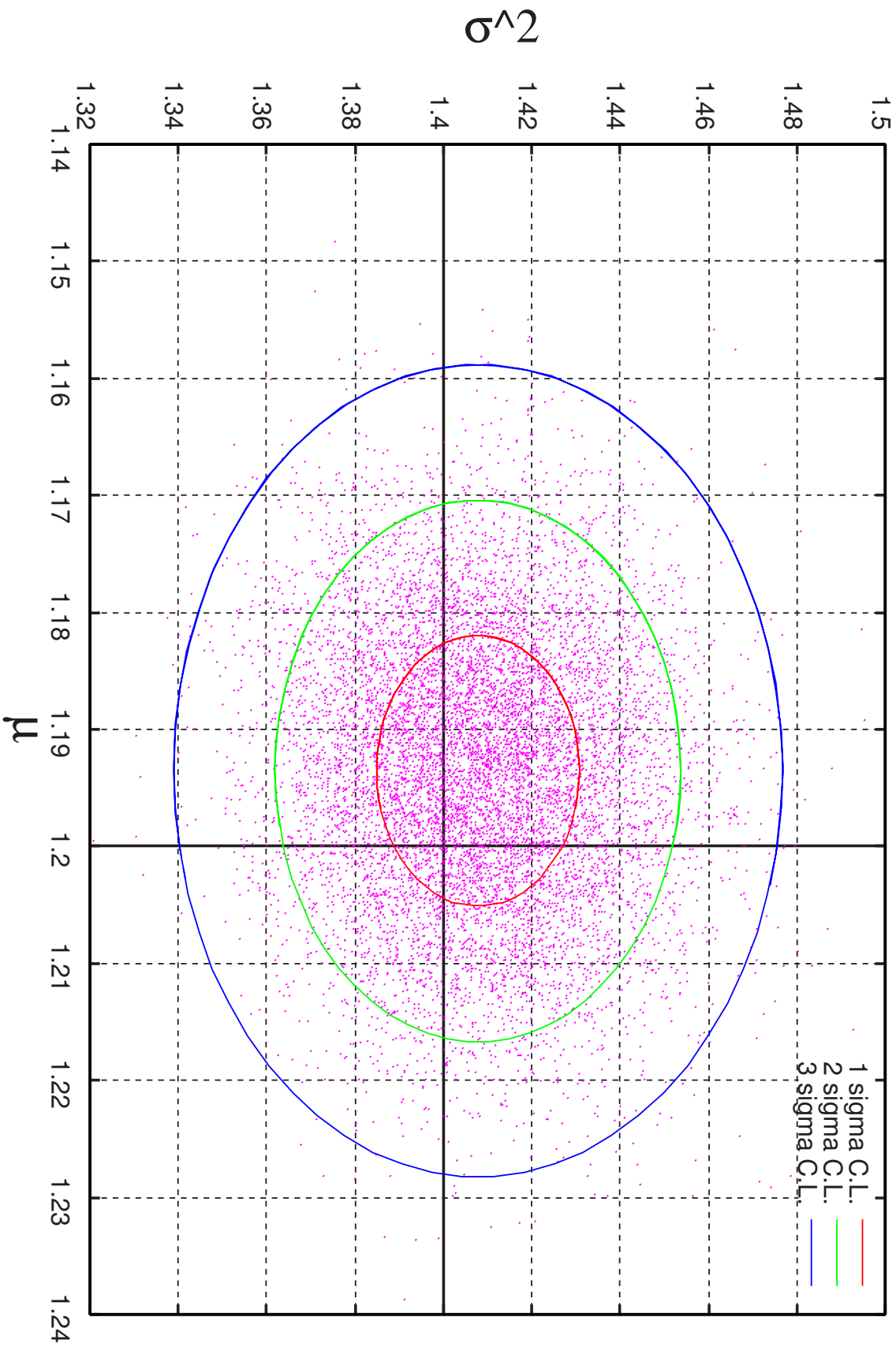

図 11　壱拾万試行後、拾ステップ毎に 10000 データサンプリングした μ, σ^2 の分布

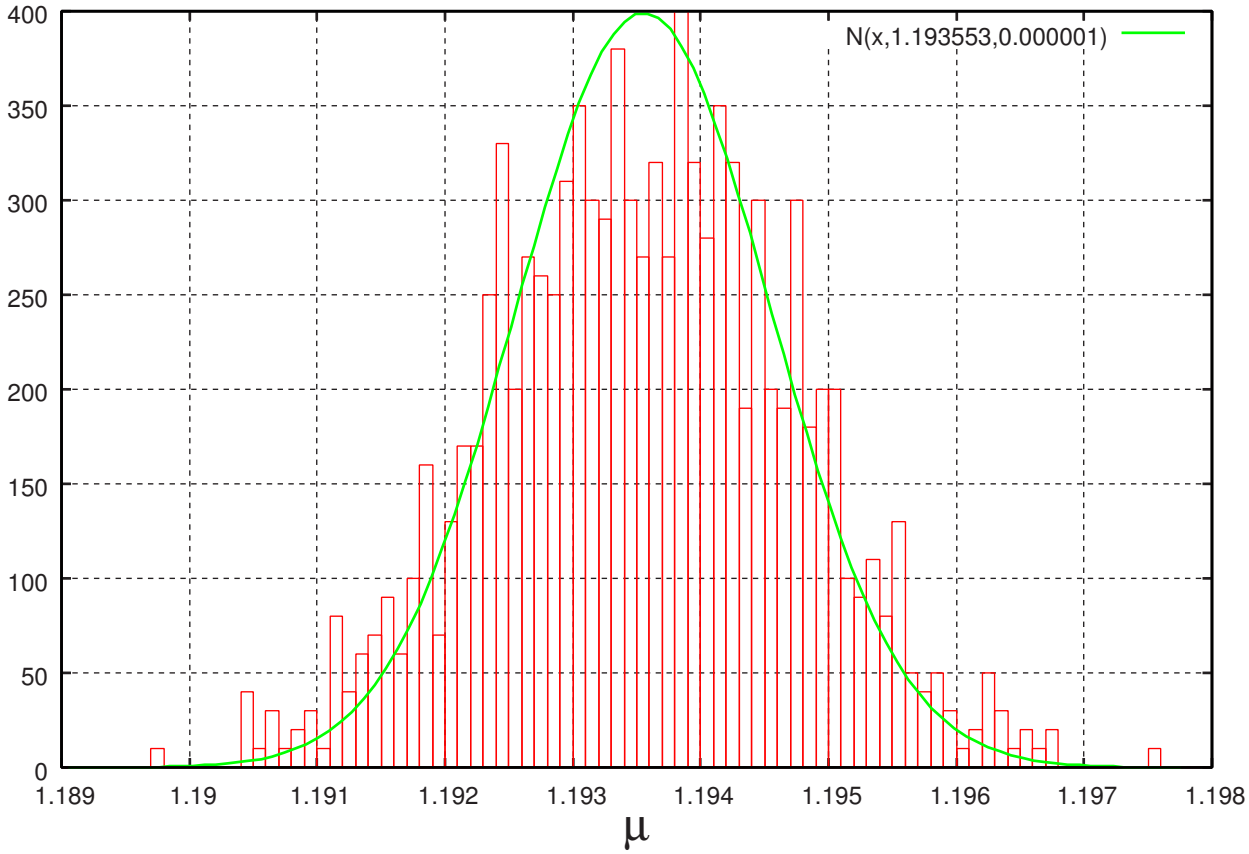

図 12　壱モンテカルロ試行の平均を 1000 回繰り返した場合の μ の分布。$N(x, 1.193553, 0.000001)$

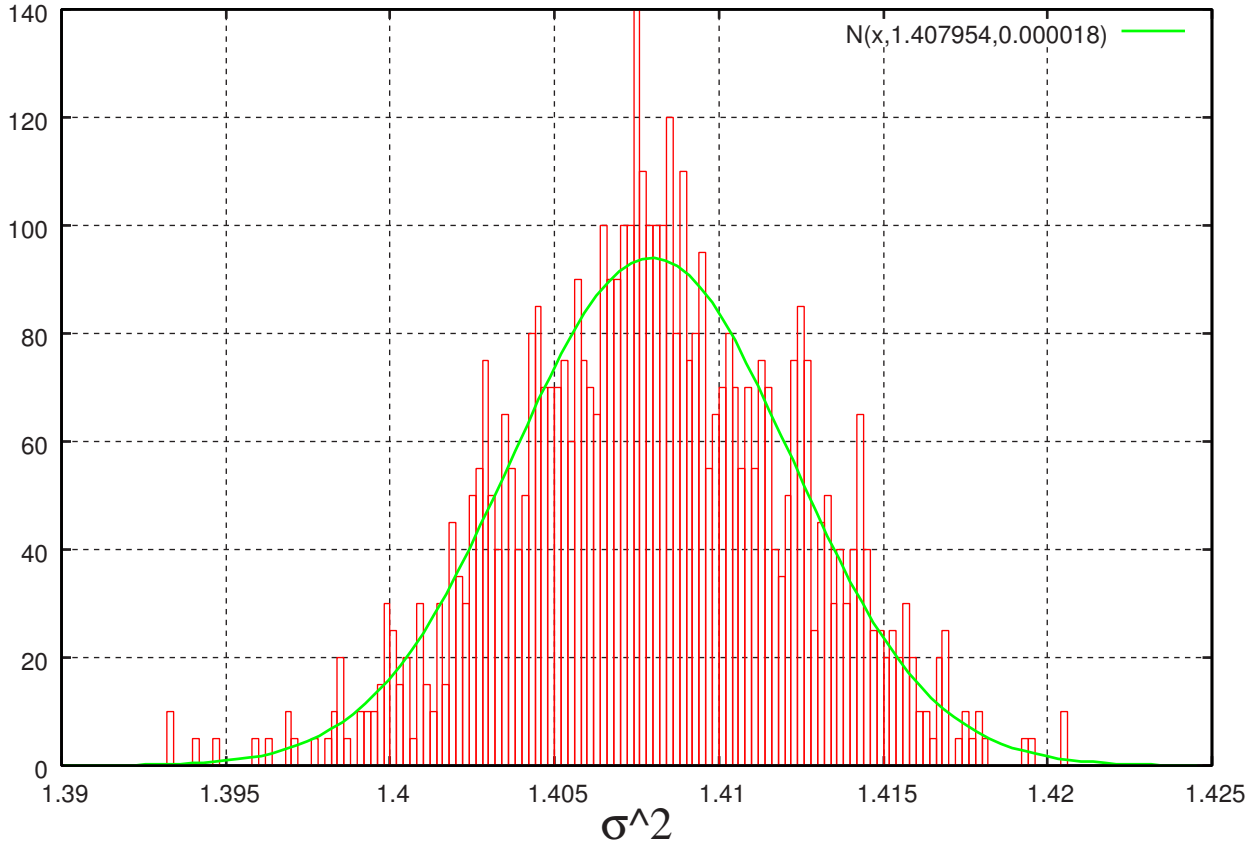

図 13　壱モンテカルロ試行の平均を 1000 回繰り返した場合の σ^2 の分布。$N(x, 1.407954, 0.000018)$

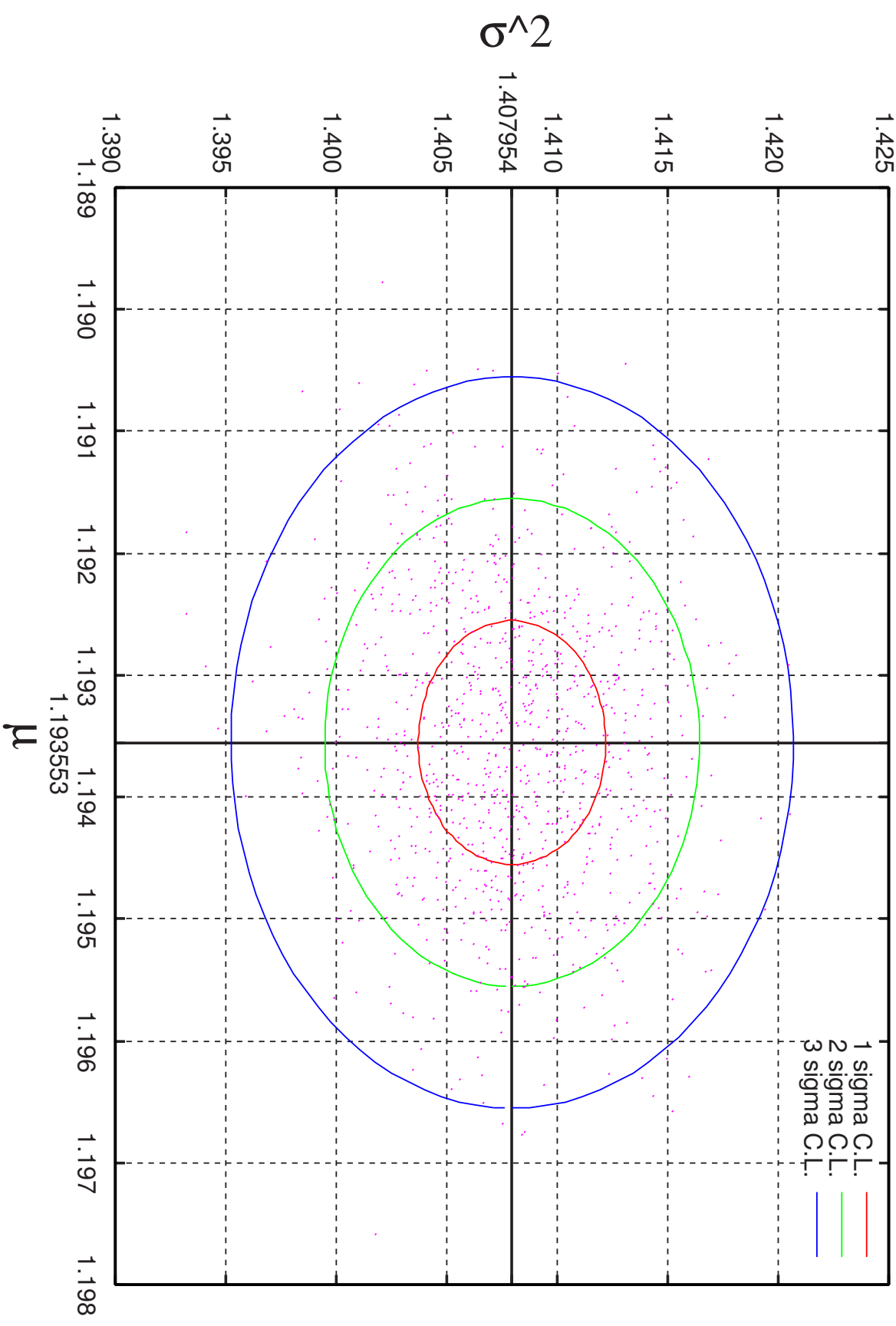

図 14　壱モンテカルロ試行の平均を 1000 回繰り返した場合の μ, σ^2 の分布。

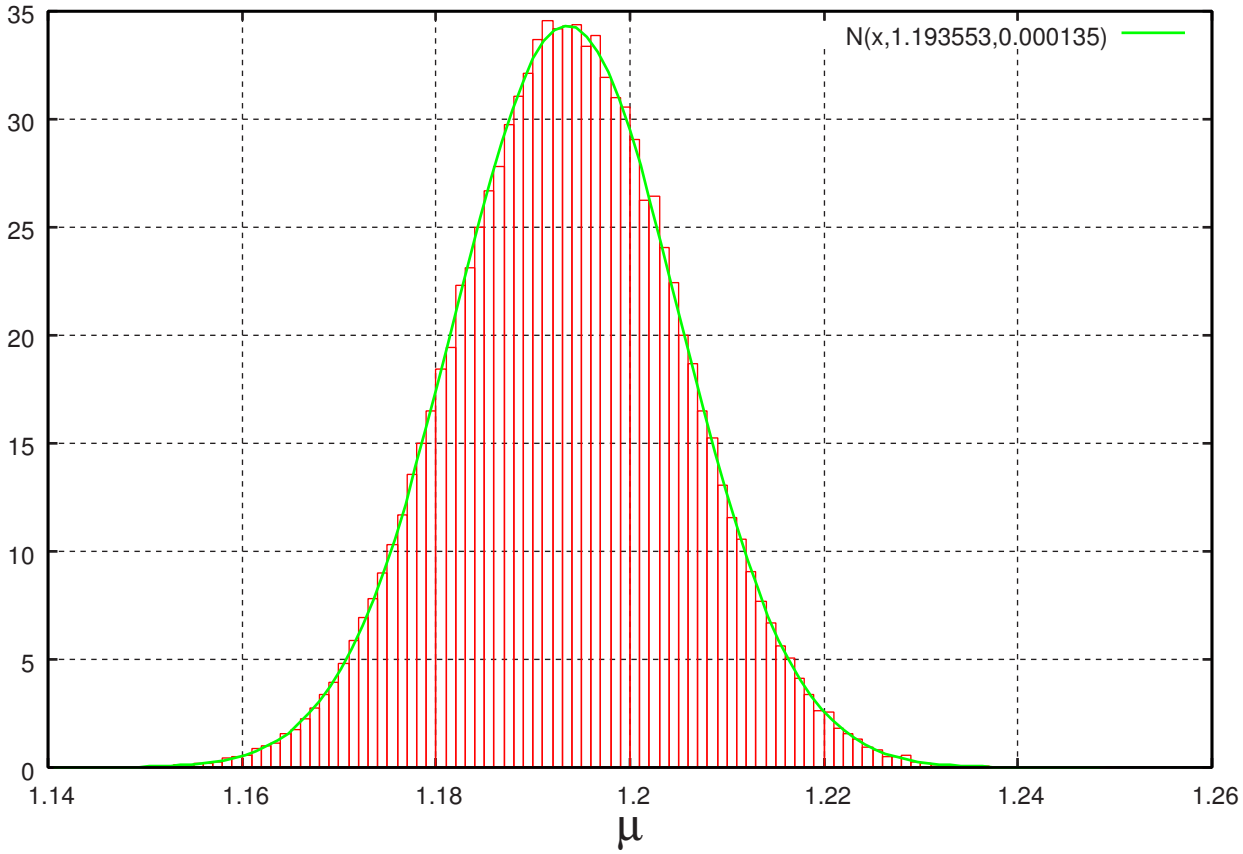

図 15　壱モンテカルロ試行当たり 500 個、これを 1000 回繰り返した場合の μ の分布。$N(x, 1.193553, 0.000135)$

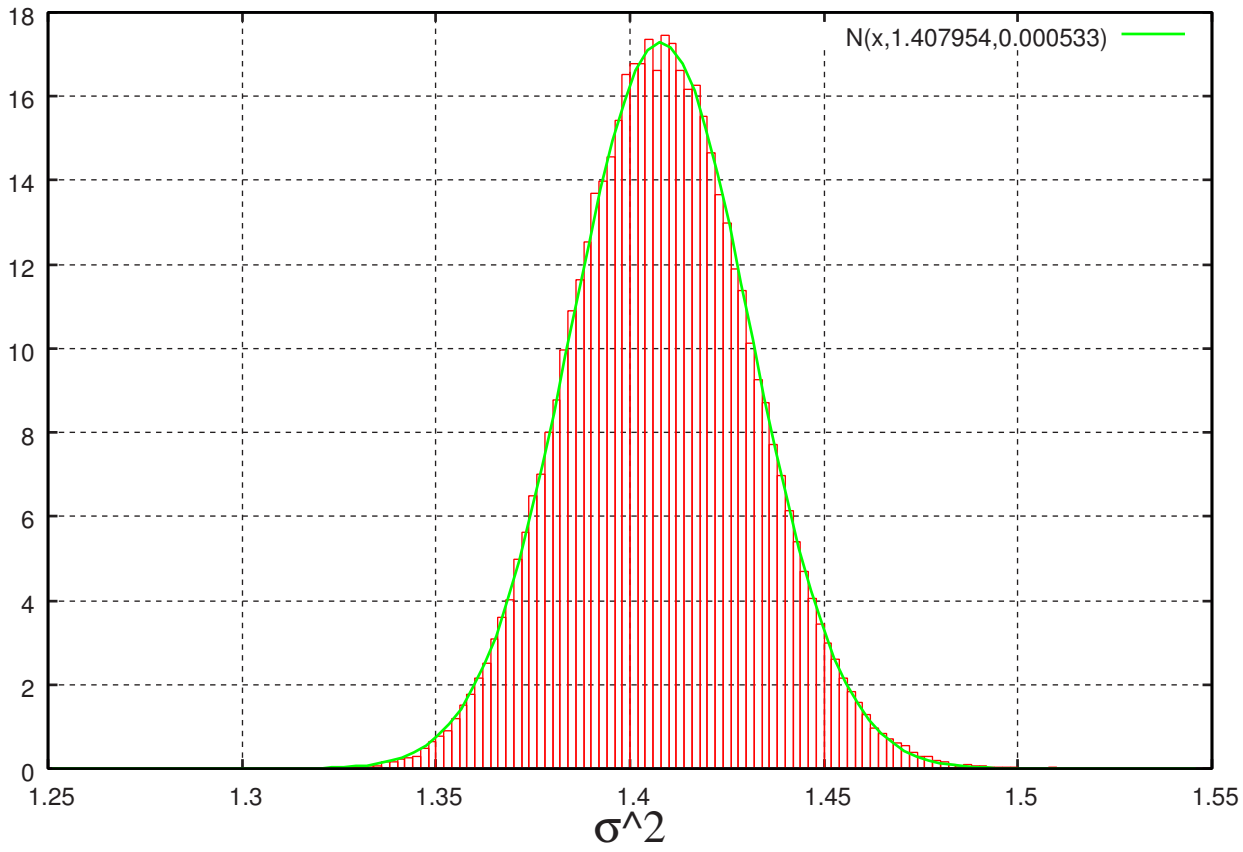

図 16　壱モンテカルロ試行当たり 500 個、これを 1000 回繰り返した場合の σ^2 の分布。$N(x, 1.407954, 0.000533)$

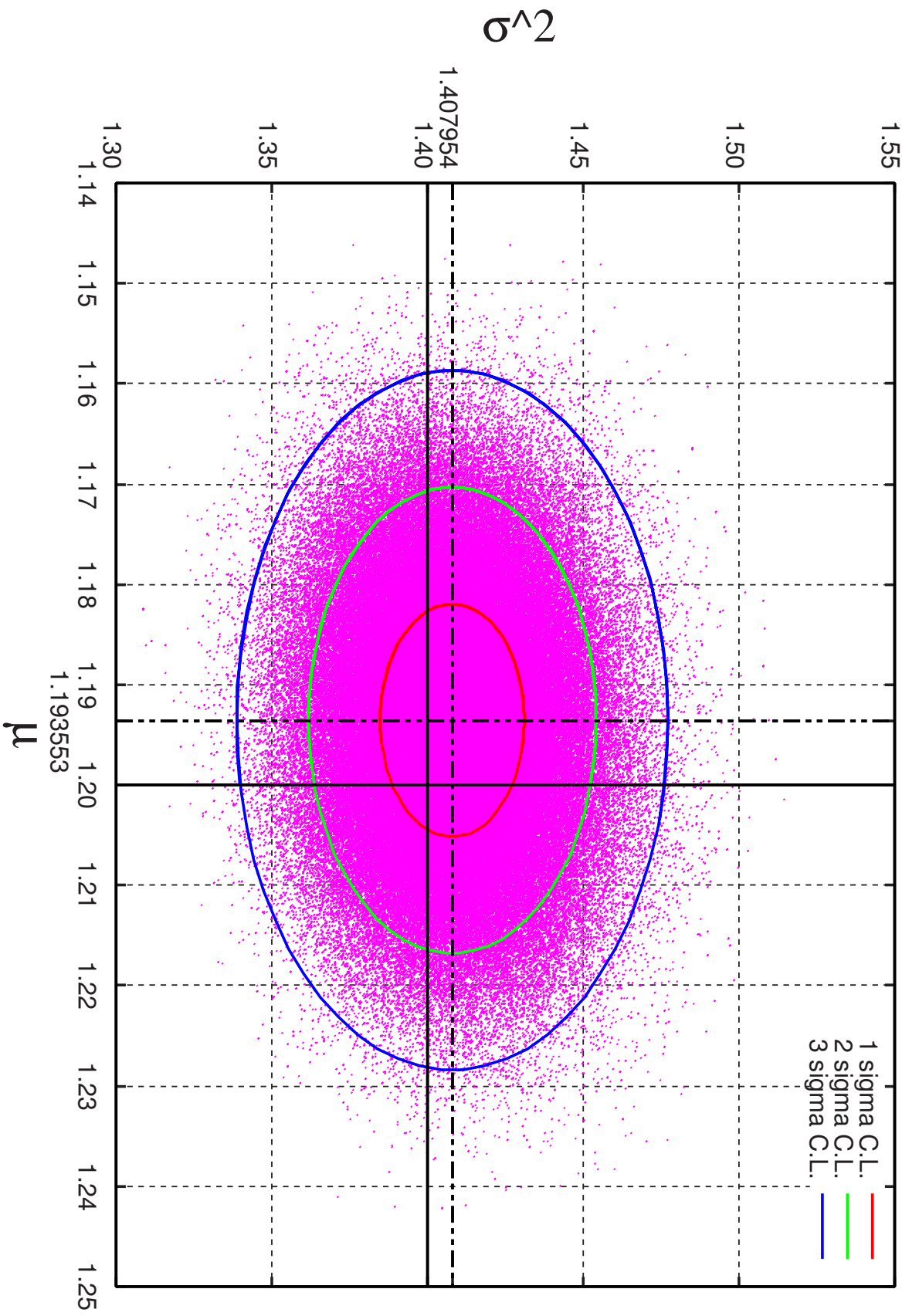

図 17　壱モンテカルロ試行当たり 500 個、これを 1000 回繰り返した場合の μ, σ^2 の分布。