# Problem Set 4 R08244002 Shin-Rong, Tsai

- **Monte Carlo integration in 10 dimensions**

  **1. Results**

  First generate data to *data.txt* from my previous homework with a slightly little change (*data_generator.c*), then run the code *estimate_error.c*.

  The first two results used autocorrelation to estimate error. The others used binning method with various $n_b$ (numbers per block). Then I use MATLAB to plot $\delta \mathrm{A} - n_b$ figure, in order to see when did they reach saturations.

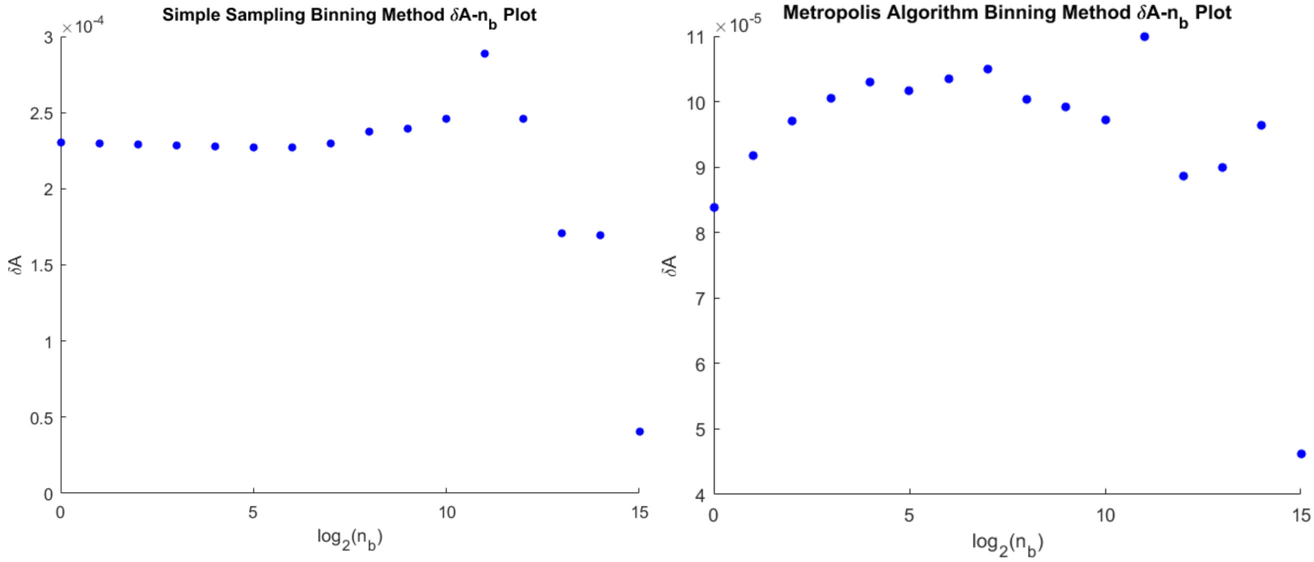
```
cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_4
$ gcc -o a.out estimate_error.c

cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_4
$ ./a.out
____Simple Sampling (autocorrelation)____
t = 1, tau_int = 5.00000e-01, sqrt(2 * tau_int) = 1.00000e+00
<A> = 2.43263e-01 +- 2.30453e-04

____Metropolis Algorithm (autocorrelation)____
t = 10, tau_int = 7.80682e-01, sqrt(2 * tau_int) = 1.24955e+00
<A> = 2.42959e-01 +- 1.04744e-04

____Simple Sampling (binning method)____
  m      nb        1/nb        deltaA
65536      1  1.00000e+00  2.30453e-04
32768      2  5.00000e-01  2.30567e-04
16384      4  2.50000e-01  2.28642e-04
 8192      8  1.25000e-01  2.27343e-04
 4096     16  6.25000e-02  2.29170e-04
 2048     32  3.12500e-02  2.26488e-04
 1024     64  1.56250e-02  2.26051e-04
  512    128  7.81250e-03  2.29855e-04
  256    256  3.90625e-03  2.35865e-04
  128    512  1.95312e-03  2.39445e-04
   64   1024  9.76562e-04  2.48062e-04
   32   2048  4.88281e-04  2.88772e-04
   16   4096  2.44141e-04  2.47649e-04
    8   8192  1.22070e-04  1.70263e-04
    4  16384  6.10352e-05  1.67341e-04
    2  32768  3.05176e-05  4.29003e-05

____Metropolis Algorithm (binning method)____
  m      nb        1/nb        deltaA
65536      1  1.00000e+00  8.38254e-05
32768      2  5.00000e-01  9.18363e-05
16384      4  2.50000e-01  9.71998e-05
 8192      8  1.25000e-01  1.00399e-04
 4096     16  6.25000e-02  1.02789e-04
 2048     32  3.12500e-02  1.01676e-04
 1024     64  1.56250e-02  1.03882e-04
  512    128  7.81250e-03  1.04501e-04
  256    256  3.90625e-03  1.00172e-04
  128    512  1.95312e-03  1.00013e-04
   64   1024  9.76562e-04  9.75213e-05
   32   2048  4.88281e-04  1.09538e-04
   16   4096  2.44141e-04  8.84801e-05
    8   8192  1.22070e-04  9.17238e-05
    4  16384  6.10352e-05  9.86423e-05
    2  32768  3.05176e-05  4.25723e-05
```




**2. Discussion**

From the results of calculating autocorrelation, we can see that:

| | Simple Sampling | Metropolis Algorithm |
|---|---|---|
| $t$ | 1 | 10 |
| $\sqrt{2 \cdot \tau_{int}}$ | 1 | 1.24955 |
| $\delta A$ | $2.30453 \times 10^{-4}$ | $1.04744 \times 10^{-4}$ |

$$\langle A \rangle = \frac{1}{N}\sum_{i=1}^{N} A_i \pm \sqrt{\frac{\langle A^2 \rangle - \langle A \rangle^2}{N-1}} \cdot \sqrt{2 \cdot \tau_{int}}$$

In Simple Sampling, since the random number is purely from random number generator, it has no correlation in the successive measurements. So "$t$" stops at 1 and $\sqrt{2 \cdot \tau_{int}}$ is 1, which means the estimated error through autocorrelation is the same as the original one.

In Metropolis Algorithm, "$t$" stops at 10 and $\sqrt{2 \cdot \tau_{int}}$ is not 1, which means there might be some correlation in their successive data series. It is reasonable that the following new random number depends on the previous one, it is not purely random. So eventually we have to multiply this potential error back. Despite of the fact that its $\sqrt{2 \cdot \tau_{int}}$ is bigger than Simple Sampling's, after multiplying their own original data error, Metropolis Algorithm's $\delta A$ is still smaller than Simple Sampling's. I think it is because that Metropolis Algorithm is weighted, the original error of data itself is small.

From the results of binning method, we can estimate their errors by deciding when they have reached saturation. And their results are close to the one estimated by autocorrelation.

Final results:

| | Simple Sampling | Metropolis Algorithm |
|---|---|---|
| Autocorrelation $\delta A$ | $2.30453 \times 10^{-4}$ | $1.04744 \times 10^{-4}$ |
| Binning Method $\delta A$ | $2.88772 \times 10^{-4}$ | $1.09538 \times 10^{-4}$ |
| $\langle A \rangle$ | $2.43263 \times 10^{-1}$ | $2.42959 \times 10^{-1}$ |

- **The Jackknife Method**

  To show that using Jackknife Method to estimate the error of the mean will get the same result as the original one, is equivalent to show that $(\sigma_{\bar{y}}^{JK})^2 = \sigma_{\bar{A}}^2$.

$$(\sigma_{\bar{y}}^{JK})^2 = \frac{N-1}{N}\sum_{s=1}^{N}(y(A_s^{JK}) - \bar{y}^{JK})^2$$

$$\sigma_{\bar{A}}^2 = \frac{\overline{A^2} - \bar{A}^2}{N-1} = \frac{1}{N-1}\left[\frac{1}{N}\left(\sum_{i=1}^{N} A_i^2\right) - \bar{A}^2\right] = \frac{1}{N-1}\left[\frac{1}{N}\sum_{i=1}^{N}(A_i - \bar{A})^2\right]$$

Where $A_s^{JK} = \frac{1}{N-1}\sum_{i\neq s}^{N} A_i$ and $\bar{y}^{JK} = \frac{1}{N}\sum_{s=1}^{N} y(A_s^{JK})$.

*<pf>:*

$$y(A_s^{JK}) - \bar{y}^{JK} = \frac{N\cdot\bar{y}^{JK} - y(A_i)}{N-1} - \frac{1}{N}\sum_{s=1}^{N} y(A_s^{JK})$$

$$= \frac{1}{N-1}\left[N\cdot\bar{y}^{JK} - y(A_i) - \frac{N-1}{N}\sum_{s=1}^{N}\frac{N\cdot\bar{y}^{JK} - y(A_i)}{N-1}\right]$$

$$= \frac{1}{N-1}\left[N\cdot\bar{y}^{JK} - y(A_i) - N\cdot\bar{y}^{JK} + \frac{1}{N}\sum_{s=1}^{N} y(A_i)\right]$$

$$= \frac{1}{N-1}[y(\bar{A}) - y(A_i)] = \frac{1}{N-1}[\bar{A} - A_i]$$

Bring this result back to the first equation, we get

$$(\sigma_{\bar{y}}^{JK})^2 = \frac{N-1}{N}\sum_{s=1}^{N}(y(A_s^{JK}) - \bar{y}^{JK})^2 = \frac{1}{N\cdot(N-1)}\sum_{i=1}^{N}(A_i - \bar{A})^2 = \sigma_{\bar{A}}^2$$

■

- **Binning with the Jackknife**

  **1. Results**

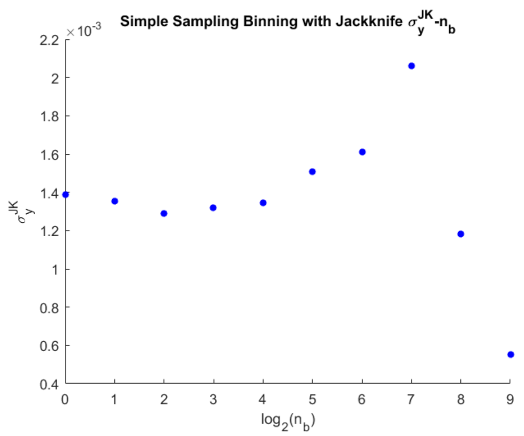
  Run the code *binning_jackknife.c*, and plot the result with MATLAB to see the saturation part.

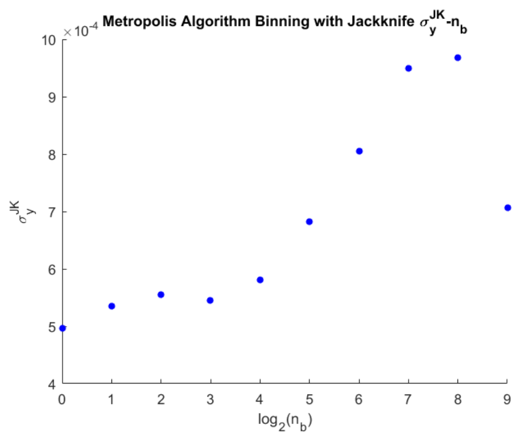
```
cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_4
$ gcc -o d.out binning_jackknife.c

cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_4
$ ./d.out
Best estimate of the secondary quantity:
Simple Sampling      : 7.84313e-01
Metropolis Algorithm : 7.85314e-01

____Simple Sampling (Binning with Jackknife)____
  m      nb       yjk_ave       errorJK
 1024      1  7.84313e-01  1.38947e-03
  512      2  7.84313e-01  1.35476e-03
  256      4  7.84313e-01  1.29095e-03
  128      8  7.84313e-01  1.31758e-03
   64     16  7.84313e-01  1.34615e-03
   32     32  7.84313e-01  1.50613e-03
   16     64  7.84313e-01  1.60866e-03
    8    128  7.84313e-01  2.05965e-03
    4    256  7.84313e-01  1.18297e-03
    2    512  7.84313e-01  5.52831e-04

____Metropolis Algorithm (Binning with Jackknife)____
  m      nb       yjk_ave       errorJK
 1024      1  7.85314e-01  4.96924e-04
  512      2  7.85314e-01  5.34764e-04
  256      4  7.85314e-01  5.55551e-04
  128      8  7.85314e-01  5.44581e-04
   64     16  7.85314e-01  5.81386e-04
   32     32  7.85314e-01  6.82351e-04
   16     64  7.85314e-01  8.04724e-04
    8    128  7.85314e-01  9.49124e-04
    4    256  7.85314e-01  9.67994e-04
    2    512  7.85314e-01  7.06769e-04
```




  **2. Discussion**

  From the course we know that the best estimate of the secondary quantity is $\bar{y} = y(\bar{A})$, which matches the results from $\bar{y}^{JK}$. It means that *y(A)* acts like a linear function, since $\bar{y} = y(\bar{A}) = y\left(\frac{1}{N}\sum_{s=1}^{N} A_s^{JK}\right) = \bar{y}^{JK} = \frac{1}{N}\sum_{s=1}^{N} y(A_s^{JK})$. Maybe it's because the difference is small, that segment is linear.

  Final result:

| | Simple Sampling | Metropolis Algorithm |
|---|---|---|
| Secondary Observable $f$(I) | $7.84313 \times 10^{-1} \pm 2.05965 \times 10^{-3}$ | $7.85314 \times 10^{-1} \pm 9.67994 \times 10^{-4}$ |