Monte Carlo Method

A TUESDAY

9:50-12:15

三教1200

清華大學天文系

Department of Astronomy, Tsinghua University

Monte-Carlo Method

1. **What is MC method?**

   To use randomness to solve deterministic problems!

2. **Key technique:**

   Random Pseudo Number Generator

3. **Main applications:**

   3.1 Numerical integration (basic)

   3.2 Generating draws with a given probability distribution

   3.3 Advanced integration and Bayesian inference (MCMC)

Key concept: Repeated Random Sampling

Core technic: Random Number Generator

HRNG (Hardware Random Number Generator) - an entropy source generating randomness by physical processes: rolling dice, coin flip, roulette wheel, radio active decay, thermal noise, clock drift, mouse movement, astronomical observations of background noise, or stock market fluctuations. These classical methods are:

- ★ *Truly random!*
- ★ *Unpredictable!*
- ★ Not reproducible: making program not able to debug!
- ★ Very slow (a few bit/sec)
- ★ Not uniformly random (physical phenomena)!

Key concept: Repeated Random Sampling

Core technic: Random Number Generator

Alternatively, **QRNG (Quantum Random Number Generator)** exploiting the property of uncertainty principle of quantum mechanics, measuring quantum fluctuation in vacuum, through quantum entanglement (self-testing via violation to the Bell-inequality). Used in for cryptography security, Monte Carlo simulations etc.

★ *Truly random!*

★ *Unpredictable!*

★ Up to 10Gbps!

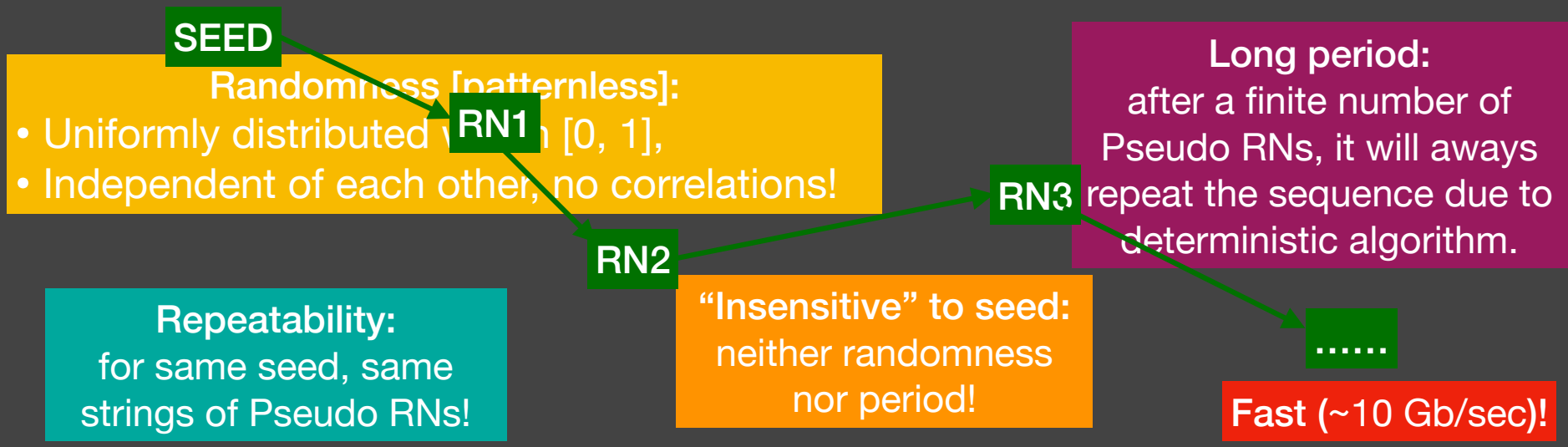
★ Not reproducible: making program not able to debug!

* A review on development of QRNG: https://arxiv.org/pdf/2203.00261.pdf
* QRNG, self-testing problem: https://www.nature.com/articles/npjqi201621

Key concept: Repeated Random Sampling

Core technic: Random Number Generator

SRNG (Software Random Number Generator) - using *deterministic algorithm* to realize ***Pseudo*** randomness (this lecture!)



Randomness [patternless]:
- Uniformly distributed within [0, 1],
- Independent of each other, no correlations!

Long period:
after a finite number of Pseudo RNs, it will aways repeat the sequence due to deterministic algorithm.

Repeatability:
for same seed, same strings of Pseudo RNs!

"Insensitive" to seed:
neither randomness nor period!

Fast (~10 Gb/sec)!

Portable:
executable in the same way on different computer architectures!

"Sensitive" to seed (as chaos phenomena are sensitive to initial condition):
Sequences by slightly different seeds diverge very quickly to avoid of correlation in multi-parameter cases!

Key concept: Repeated Random Sampling

Core technic: Random Number Generator

CSPRNG (Cryptography Secure *Pseudo* Random Number Generator) - directly related to everyday life: computer operating system, encrypted message transmission, banking and finance etc. It requests unpredictability and no way of gaining knowledge of future as well as past! (HRNG is often used in combination to generate initial SEED for CSPRNG)

Examples of RNG algorithm

- **Linear congruential generator (LCG)**

$$X_{i+1} = (aX_i + b) \bmod m \quad (a, b, m \text{ are all carefully chosen integers!})$$

Map this to $[0, 1)$ by $U_{i+1} = \frac{X_{i+1}}{m}$, then $\{U_i\}$ gives a random number series in $[0,1)$

2,147,483,647: the 8th Mersenne Prime!

★ Ansi-C: a = 1103515245, b = 12345, $m = 2^{31}$, Period $\sim m \sim$ 2E9

★ RAND/MINSTD (matlab/C/C++): a=16807, b=0, $m = 2^{31} - 1$, Period $\sim m \sim$ 2E9

★ RANF (CRAY): a = 44,485,709,377,909, b=0, $m = 2^{48} - 1$, Period $\sim m \sim$ 3E14

★ drand48 (LINUM/UNIX): a=25214903917, b=11, $m = 2^{48}$, Period $\sim m \sim$ 3E14

★ RANDU (Fortran): a=65539, b=0, $m = 2^{31}$, Period $\sim m/4 \sim$5E8, *very bad! Fail many Random test!*

★ NAG (Fortran): $a = 13^{13}$, b=0, $m = 2^{59}$, Period $\sim m/4 \sim$5E17, *lower-ranked random bits are "sub-random") —> do not use them!*

either power of 2, or Mersenne *numbers* have simple bitwise operations for modulo

Examples of RNG algorithm

- **Combined LCG:**

★ e.g., ran2 in *Numerical Recipes,* $\text{Period} \sim 2.3 \times 10^{18}$

$$X_{i+1} = (40014 X_i) \bmod 2147483563$$
$$Y_{i+1} = (40692 Y_i) \bmod 2147483399$$
$$Z_{i+1} = (X_{i+1} + Y_{i+1}) \bmod 2147483563$$

Map this to $[0, 1)$ by $U_{i+1} = \dfrac{Z_{i+1}}{2147483563}$

- **Permuted Congruential Generator (PCG, 2014+)**

★ Applying an output permutation function (bitwise rotation/shift) to improve randomness of modulo-2^n LCG. Excellent quality (pass all RANDOMNESS tests!), arbitrary long period, reproducible (but not easy to predict!), multiple stream feature, very fast, small storage!

Examples of RNG algorithm

- **Lagged Fibonacci Generators (LFG)**

Fibonacci sequence: $F_n = F_{n-1} + F_{n-2}$

$$S_n = S_{n-j} \clubsuit S_{n-k} \pmod{m} \quad [0 < j < k, m = 2^{p(e.g., =32 \text{ or } 64)}]$$

$\clubsuit$ is a binary operation: $+, -, \times, \text{XOR}$

Generators of this kind must remember the last k words of states!

XOR => Generalized feedback shift register (GFSR)

- **PANLUX:** Period of 10^{171} (Portable Fortran 77 implementation)
- *Mersenne Twister MT19937*: Period of $2^{19937} - 1 \sim 10^{5982}$ (GFSR with additional bit shift)

The 24th Mersenne Prime!

*MT is the default PRNG for Microsoft Excel, GNU, IDL, Matlab, *Python (NumPy)*, R, Julia etc

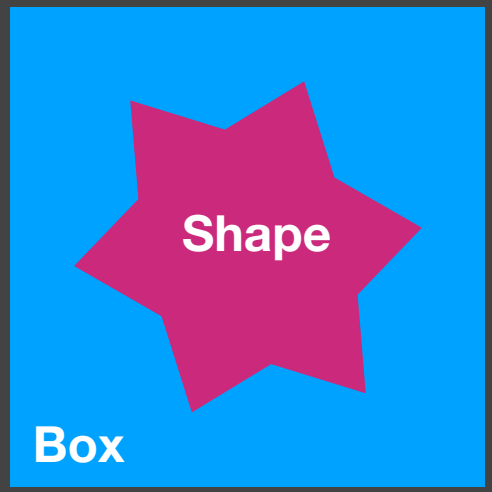
Applications I — Numerical integration

Monte-Carlo Integration

Prelude: ask for integration I of area within the shape

$$\frac{A_{\text{shape}}}{A_{\text{box}}} = P(\text{a random dart hits the shape})$$

$$= \frac{\#\ \text{hits shape}}{\#\ \text{in box}}$$



$$I = A_{\text{shape}} = A_{\text{box}} \frac{\#\ \text{hits shape}}{\#\ \text{in box}}$$

with known A_{box}

Calculate π….

Crazy mathematicians!

Fully respected!

Indian Mathematician (1914)

Ramanujan's Formula for Pi

$$\frac{1}{\pi} = \frac{\sqrt{8}}{9801} \sum_{n=0}^{\infty} \frac{(4n)!}{(n!)^4} \times \frac{26390n + 1103}{396^{4n}}$$

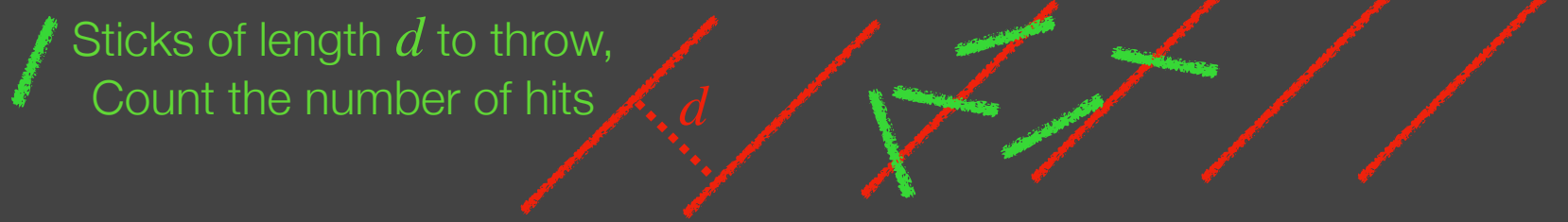
Gregory-Leibniz series (with 500,000 times to five decimal place):
π=(4/1)-(4/3)+(4/5)-(4/7)+(4/9)-(4/11)+(4/13)-(4/15)+…

Calculate $\pi = X \sin(180°/X)$ with a large X in degree.

Buffon's Needle Problem:

$$P\left(\frac{\#\ of\ \ hit}{\#\ of\ total\ \ throw}\right) = 2/\pi$$

Sticks of length d to throw,
Count the number of hits



Applications I — Numerical integration

Monte-Carlo Integration *Uniform sampling*

Consider integration $I = \int_{\nu_{\vec{x}}} f(\vec{x})\, \mathrm{d}^d x$, where the integration domain $\nu_{\vec{x}}$ is spanned by $\vec{x} = \{x_1, x_2, x_3 \dots\}$ in d dimensions and has a volume of $V_{\vec{x}} = \Delta_{x_1}\Delta_{x_2}\Delta_{x_3}\dots$

Now for simplicity let's define $f(\vec{x})$ on $\vec{x} \in [0,1)^d$. To compute the MC integral I, we do the following:

1. Generate N random vectors (with RNG) of $\vec{x}_i \in [0,1)^d$ with flat distribution
2. Compute $I_N = V_{\vec{x}}\langle f \rangle = V_{\vec{x}} \left[\frac{\Sigma_i^N f(\vec{x}_i)}{N} \right]$.
3. We expect summation in $I_N \to I$ for $N \to \infty$.

Q: Why so? How different is I_N from the true integral I for given N?

$\sigma_{I_N} = V_{\vec{x}}\sigma_{\langle f \rangle} = V_{\vec{x}} \frac{s_f}{\sqrt{N}}$ (standard error of the mean),

where the variance s_f^2 can be estimated as $s_f^2 = \langle f^2 \rangle - \langle f \rangle^2$. As $N \to \infty$, $\sigma_{I_N} \to 0$.

Applications I — Numerical integration

Monte-Carlo Integration

Lets compare MC integration with standard integration in d dimensions using grid-based sampling with same number of sampling points of N. Note in the latter case each dimension has $n = N^{1/d}$ regular spacing points.

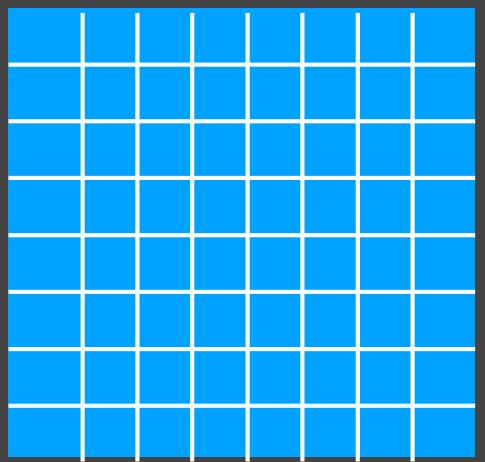
Error scales as:

$\sigma_I \propto \frac{1}{n} \propto N^{-1/d}$ (mid-point rule)

$\sigma_I \propto \frac{1}{n^2} \propto N^{-2/d}$ (trapezoidal rule)

$\sigma_I \propto \frac{1}{n^4} \propto N^{-4/d}$ (Composite Simpson's rule)

$\sigma_I \propto \propto N^{-1/2}$ (MC integration)

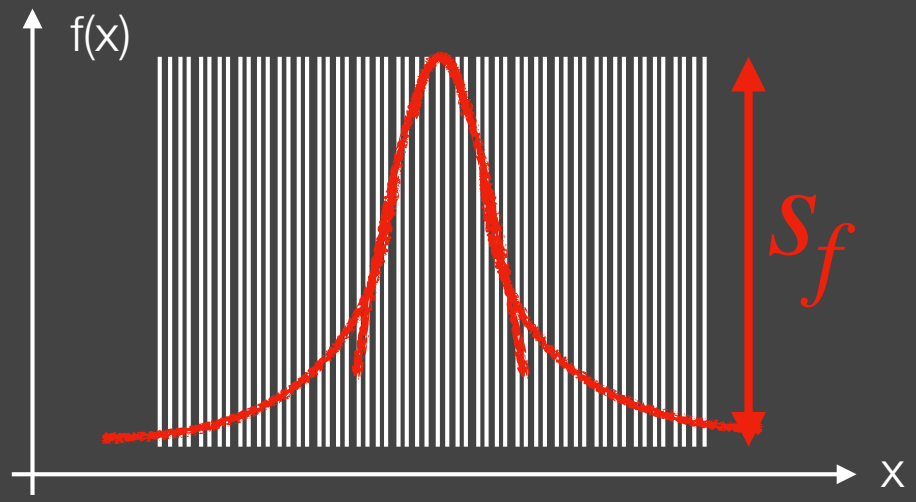


For small d, standard integration (SI) is better than MCI.
For high d, MCI is more efficient than SI using grid-based sampling!

Applications I — Numerical integration

Monte-Carlo Integration

But, above MCI scheme under uniform sampling is not always efficient!
Consider a Gaussian or an exponential function:



Suppose we have a function $f(x)$ (integrand) very spiky, under uniform sampling,

$s_f = \sqrt{\langle f^2 \rangle - \langle f \rangle^2}$ can be very large, so that $\sigma_{I_N} \propto s_f/\sqrt{N}$ is also large!

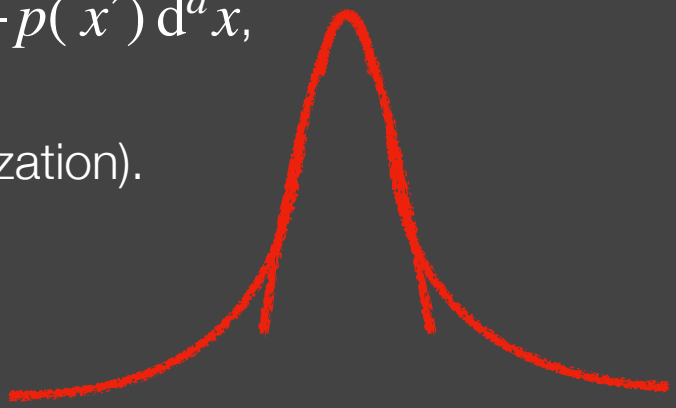
Therefore, uniform sampling is not efficient to obtain accurate results!

Applications I — Numerical integration

Monte-Carlo Integration

- Now consider $I = \int_{\nu_x} f(\vec{x})\,\mathrm{d}^d x = \int_{\nu_x} \frac{f(\vec{x})}{p(\vec{x})} p(\vec{x})\,\mathrm{d}^d x,$

  where $\int_{\nu_x} p(\vec{x})\,\mathrm{d}^d x = \int_{\nu_P} \mathrm{d}^d P = 1$ (normalization).



- Define $g(\vec{x}) \equiv \frac{f(\vec{x})}{p(\vec{x})}$, then rewrite:

$$I = \int_{\nu_x} g(\vec{x})\, p(\vec{x})\,\mathrm{d}^d x = \int_{\nu_P} g(\vec{x}(p))\,\mathrm{d}^d [P].$$

Uniform in P

$\vec{x} \sim p(\vec{x})$

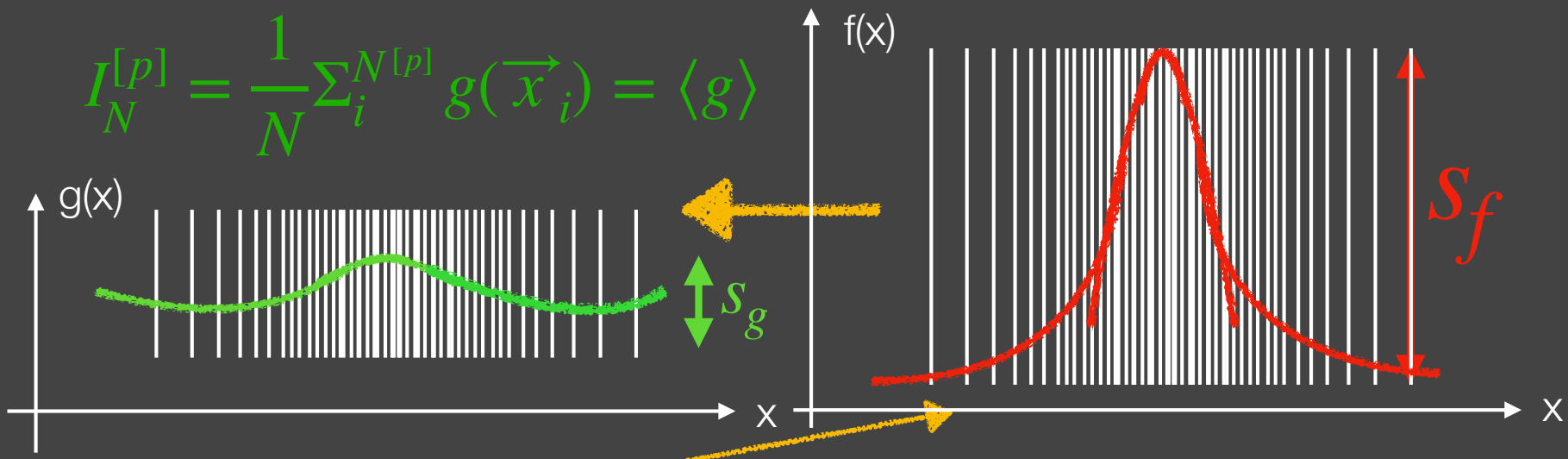
- The latter integral can also be approximated by $I_N^{[p]} = \frac{1}{N} \Sigma_i^N {}^{[p]} g(\vec{x}_i) = \langle g \rangle.$
- Now the sampling is not uniform in $\vec{x}$ but uniformly in $\vec{P}$ instead. $\vec{x}$ is sampled such that per $\vec{P}$ interval $\mathrm{d}^d [P] = p(\vec{x})\mathrm{d}^d x$ is uniform, i.e., $\vec{x}$ follows $p(\vec{x})$.
- Note that in this case the volume V_P of the integration domain ν_p naturally becomes 1, because of the normalization condition for $p(\vec{x})$.

Applications I — Numerical integration

Monte-Carlo Integration

Importance sampling

$$I_N^{[p]} = \frac{1}{N}\Sigma_i^{N\,[p]}\, g(\vec{x}_i) = \langle g \rangle$$



If we choose $p(\vec{x})$ close to $f(\vec{x})$ so that $g(\vec{x}) \equiv \dfrac{f(\vec{x})}{p(\vec{x})}$ stays as flat as possible,

Uniform in P

then $s_g^{[p]} = \sqrt{\langle g^2 \rangle - \langle g \rangle^2}$ can be arbitrarily small!

$\vec{x} \sim p(\vec{x})$

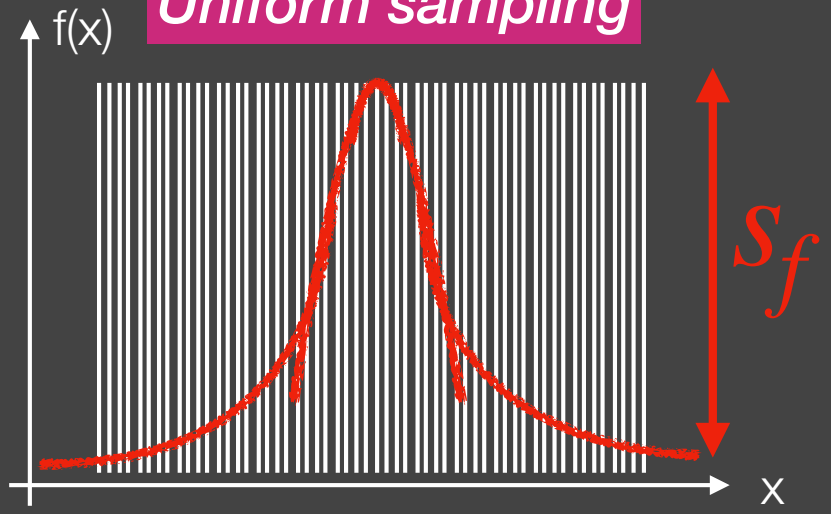
That means $\sigma_{I_N}^{[p]} \propto s_g/\sqrt{N}$ can also be small, while

$\sigma_{I_N}^{[\text{flat x}]} \propto s_f/\sqrt{N}$ (under uniform sampling) is large!

Applications I — Numerical integration

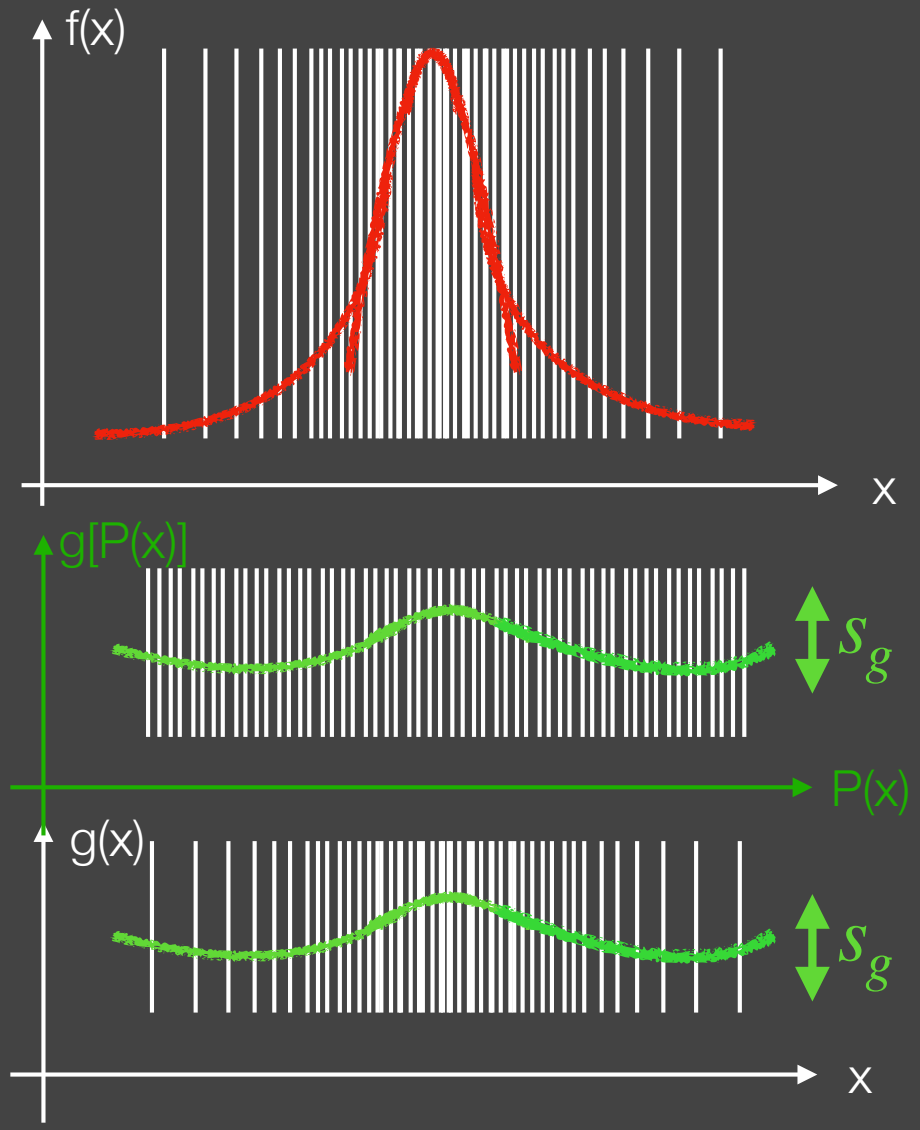
Monte-Carlo Integration

Uniform sampling



$$I = \int_{\nu_x} f(\vec{x})\,\mathrm{d}^d x \quad I_N^{[x]} = \frac{V_x}{N}\Sigma_i^N f(\vec{x}_i) = V_x\langle f\rangle$$

Importance sampling



$$I = \int_{\nu_x} \frac{f(\vec{x})}{p(\vec{x})} p(\vec{x})\,\mathrm{d}^d x, \quad g(\vec{x}) \equiv \frac{f(\vec{x})}{p(\vec{x})}$$

$$I_N^{[p]} = \frac{1}{N}\Sigma_i^{N[p]} g(\vec{x}_i) = \langle g\rangle = V_x\langle f\rangle$$

$\vec{x} \sim p(\vec{x})$

Applications I — Numerical integration

Monte-Carlo Integration

$$I = \int_0^1 \underbrace{(x^{-1/3} + x/10)}_{\text{Integrand f(x)}}\,\mathrm{d}x\,(=1.55)$$

Divergent

Error off *uniform-sampling* MCI: $\sigma_{I_N}^{[\mathrm{x\ flat}]} \propto \sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}} = \frac{0.87}{\sqrt{N}}$

Uniform in X

With *importance sampling* $p(x) = \frac{2}{3}x^{-1/3}\ (0 \leqslant x \leqslant 1),\ g(\vec{x}) \equiv \frac{f(\vec{x})}{p(\vec{x})}.$

Uniform in P

Error of importance-sampling MCI: $\sigma_{I_N}^{[p]} \propto \sqrt{\frac{\langle g^2\rangle - \langle g\rangle^2}{N}} = \frac{0.045}{\sqrt{N}},$

400 times more efficient (less samplings) than the former
to reach a given integration error!

Applications II — Probability distribution

Generate a sample of a given probability distribution

- Simple inversion method

pdf of uniform, Gaussian etc…

We want to use draws of x from a known probability distribution $p_1(x)$ to generate draws of y for a desired distribution $p_2(y)$, how to find $y = y(x)$?

Given $y = y(x)$, interval probabilities must satisfy: $p_1(x)\,\mathrm{d}x = p_2(y)\,\mathrm{d}y$, or the

Integral relation: $$\int_{-\infty}^{x} p_1(x')\,\mathrm{d}x' = \int_{-\infty}^{y=y(x)} p_2(y')\,\mathrm{d}y'$$

$$P(\leqslant x) \equiv \int_{-\infty}^{x} p(x')\,\mathrm{d}x' \in [0,1] \text{ (cumulative distribution function)}$$

$$\rightarrow P_1(\leqslant x) = P_2(\leqslant y) \rightarrow y = P_2^{-1}[P_1(\leqslant x)]$$

From x to yielding final distribution of y through inversion of P_2^{-1} !

Applications II — Probability distribution

Generate a sample of a given probability distribution

- Simple inversion method

Now consider a simple uniform distribution of $p_1(x) = 1$ with $x \in [0, 1)$,

then $P_1(\leqslant x) \equiv \int_0^x p_1(x')\,\mathrm{d}x' = x \in [0,1]$,

i.e., *uniformly draw* $x \in [0, 1)$ *with a RNG*

Then convert x to y through $y = P_2^{-1}[P_1(\leqslant x)] = P_2^{-1}[x]$, where $P_2(\leqslant y) \equiv \int_{-\infty}^{y} p_2(y')\,\mathrm{d}y'$

Exercise: We want RNs from a pdf distribution $p(y) = \frac{1}{4}y^3$ in $y \in [0, 2]$

1) Calculate $P_2(\leqslant y) = \int_0^y \frac{1}{4}y'^3\,\mathrm{d}y' = \frac{y^4}{16}$ (note $P_2(y \leqslant 2) = 1$ normalized)

2) Work out inverse function $y = P_2^{-1}[x] = \sqrt[4]{16x}$.

3) Draw random number $x \in [0,1)$, through $y = \sqrt[4]{16x}$ obtain $y \sim p(y)$.

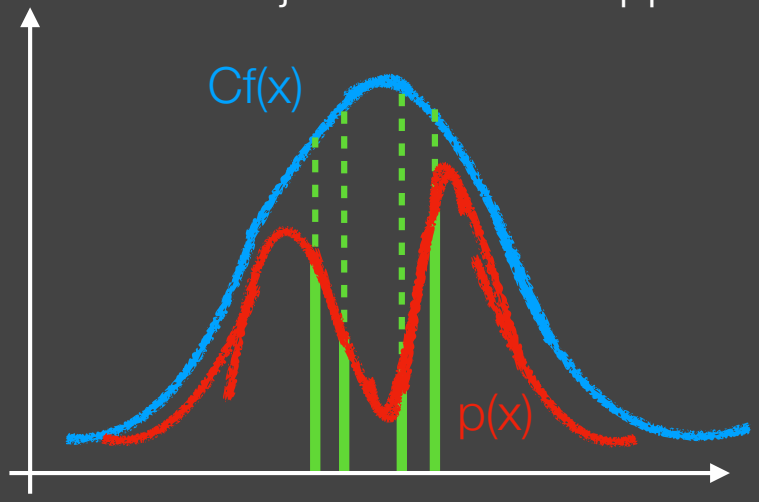
Applications II — Probability distribution

Generate a sample of a given probability distribution

- Rejection Method:

What if the inversion formula $y = P_2^{-1}[x]$ cannot be easily obtained?

Assume $p(x)$ [what was $p_2(y)$ in previous notation] is the desired distribution pdf, and $f(x)$ [what was $p_1(x)$ in previous notation] is a distribution that we can more easily generate (e.g., uniform, Gaussian etc). Rejection can be applied if we have a constant C for which $p(x) \leqslant Cf(x)$



1) generate an x that follows $f(x)$ (using RNG)
2) (for each x) create a z uniformly from $[0, Cf(x))$
3) If $z \leqslant p(x)$, return x as a desired number,
4) If $z > p(x)$, reject and repeat 1).

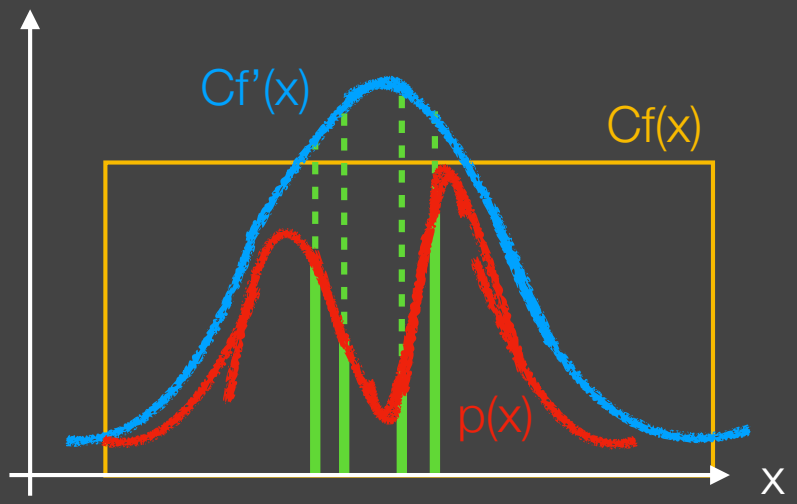
=> Finally the remained draws satisfy $x \sim p(x)$

Applications II — Probability distribution

Generate a sample of a given probability distribution

- Rejection Method:

Why this method works? Because such a sampling yield a probability per $\mathrm{d}x$ interval $\propto \mathrm{d}x\, f(x) \dfrac{p(x)}{Cf(x)} = \dfrac{1}{C} p(x)\, \mathrm{d}x$, meet the desired pdf of $p(x)$.



A common choice of $f(x)$ is a uniform distribution with $C = \max(p(x))$

In general, when $p(\vec{x})$ is complicated in particular, in high dimensions, the rejection method can also be inefficient! A good way to generalize a sample of $\vec{x}$ that follows $p(\vec{x})$ is through Markov Chain and Metropolis-Hastings Algorithm!

Applications III — Advanced Monte Carlo

Via the so-called "Monte Carlo Markov chain (MCMC)"

MCMC is a sampling scheme. It provides an efficient way to draw a sample with given probability. It can be used to assist advanced integration and statistical inference (parameter estimate, model regression) …

$$I = \int_{\nu_x} g(\vec{x})\, p(\vec{x})\, \mathrm{d}^d x = \int_{\nu_P} g(\vec{x})\, \mathrm{d}^d\,[P]$$

Markov chain — Uniform in P

Where $\vec{x} = (x_1, x_2, \ldots x_M)$ (M-dimensional integration space).

With a Markov chain of $\vec{x}$ that is generated according to $p(\vec{x})$, the integral can be ***efficiently*** estimated by summation using N element of $\vec{x}$ in the chain:

$$I_N^{[p]} = \frac{V_P}{N} \Sigma_i^N{}^{[p]} g(\vec{x}_i) = \langle g \rangle \qquad \vec{x} \sim p(\vec{x})$$

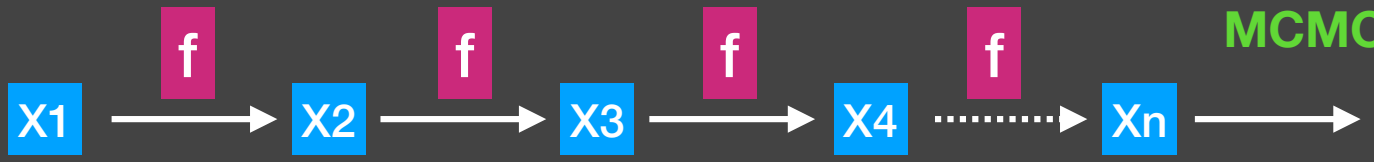
Note: $V_P = 1$ if $p(\vec{x})$ is normalized.

How to build such a chain?

Importance sampling

Monte Carlo Markov Chain

To simplify, let's consider in 1d: $\vec{x} \to x$. To construct $p(x)$ through stochastic process with $p(x)$ being the equilibrium distribution of x — we use a Markov process to generate a Markov chain, and the chain elements x will follow $p(x)$!



A Markov chain is a *sequence* of states, where **f** is a MC operator, and the *transition probability* $W_f(x \to x') \equiv W_f(x'|x)$ (a conditional probability) describes probability from x state to x' in a stochastic process under detailed balance principle when equilibrium distribution is reached. It depends only on the current state.

- Natural properties of $W_f(x \to x')$: $\int W_f(x \to x')\,\mathrm{d}x' = 1$ while $W_f \geqslant 0$

- We have evolution $p'(x') = \int p(x)\, W_f(x \to x')\,\mathrm{d}x = p(x')$

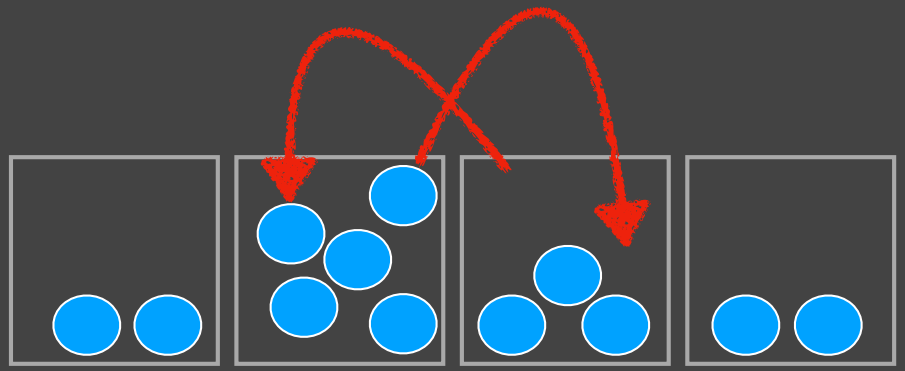
Under equilibrium distribution of Markov process!

The chain has "*Ergodic*" property => any ensemble of states approaches the equilibrium distribution as the number of steps in the chain building up to infinity —> we get $p(x)$.

Monte Carlo Markov Chain

- Under equilibrium distribution of Markov process, detailed balance condition is satisfied:

$$p(x)\, W_f(x \to x') = p(x')\, W_f(x' \to x)$$



$p(x)$: the number of balls in bin x

$W_f(x \to x')$: transition probability of balls from x to x' under equilibrium.

- Natural properties of $W_f(x \to x')$: $\int W_f(x \to x')\, \mathrm{d}x' = 1$ while $W_f \geqslant 0$

- We have evolution $p'(x') = \int p(x)\, W_f(x \to x')\, \mathrm{d}x = p(x')$

Under equilibrium distribution of Markov process!

The chain has "*Ergodic*" property => any ensemble of states approaches the equilibrium distribution as the number of steps in the chain building up to infinity —> we get $p(x)$.

Monte Carlo Markov Chain

Metropolis-Hastings Algorithm

is an algorithm that generates a Markov Chain

1) When the current state is x, propose a new state x' with a *proposal probability* $q(x \to x')$ *[function as in its name!]*

2) Calculate the *Hastings ratio*: $r = \min\left(\frac{p(x')\,q(x' \to x)}{p(x)\,q(x \to x')}, 1\right)$ acceptance probability

3) Accept *the proposed* state with a probability r. In the case of rejecting x' (with a probability of $1 - r$) the old state x is kept and used as the next element in the chain!

Metropolis update

Often a *Gaussian distribution* centered on x is often used for a *Metropolis-type* update for the proposal probability $q(x \to x')$.

Monte Carlo Markov Chain

A spacial case is: $q(x \to x') = q(x' \to x)$, i.e., a symmetric proposal probability form, then we simply have the *Hasting ratio*: $r = \min\left(\frac{p(x')}{p(x)}, 1\right)$

Advantage: normalization is NOT required!

Let's look at an example to understand the procedure better.

Suppose we want $p(x) \propto \exp\left(-\frac{x^2}{2}\right)$, let's take a simple symmetric Gaussian proposal $q(x \to x')$ such that $x' = x + 0.1\,(2u - 1)$, where $u \in [0,1)$ uniform.

Draw a random number from a uniform distribution $w \in [0,1)$, if $w \leqslant r$, accept; otherwise reject!

1) start with x for which $p(x) > 0$;
2) Draw u randomly from a uniform distribution, through $x \to x'$ to obtain x';
3) Calculate Hastings ratio $r = \min\left(1, \frac{\exp\left(-\frac{x'^2}{2}\right)}{\exp\left(-\frac{x^2}{2}\right)}\right)$;
4) Take x' as the next element with probability r, otherwise take x;
5) Go back to 2).

Monte Carlo Markov Chain

Does Metropolis-Hastings algorithm fulfill detailed balance?

Transition probability of $x \to x'$ among elements in the chain:

$$W_f(x \to x') = q(x \to x')\, r = q(x \to x') \min\left(1, \frac{p(x')\, q(x' \to x)}{p(x)\, q(x \to x')}\right)$$

proposal probability

acceptance probability

$$W_f(x \to x')\, p(x) = q(x \to x')\, \frac{p(x')\, q(x' \to x)}{p(x)\, q(x \to x')}\, p(x) = q(x' \to x)\, p(x')$$

Transition probability $x' \to x$ among elements in the chain:

$$W_f(x' \to x) = q(x' \to x) \times 1$$

proposal probability acceptance probability

Reversing the ratio above must obtain a value $\geqslant 1$, and thus $r = 1$

$$W_f(x' \to x)\, p(x') = q(x' \to x)\, p(x') = W_f(x \to x')\, p(x)$$

Detailed balance!

Metropolis-Hastings Algorithm is an algorithm that generates a Markov Chain

Monte Carlo Markov Chain

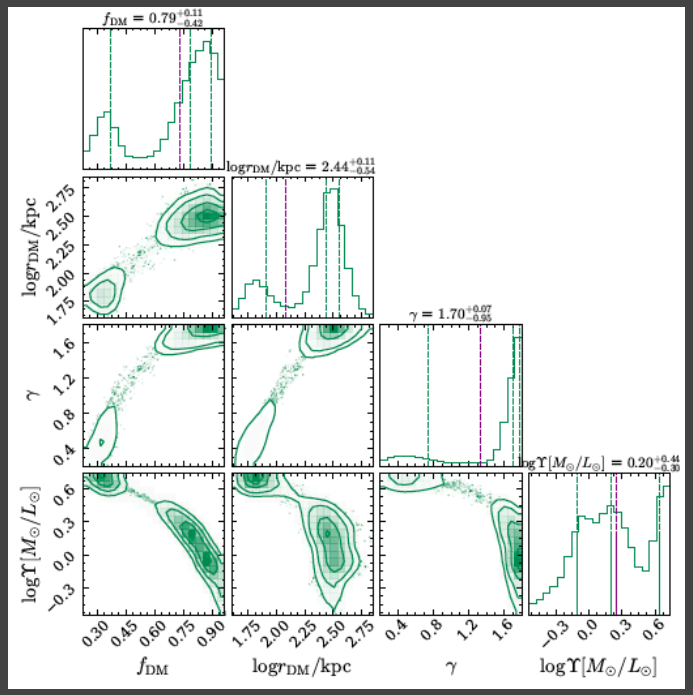
Note 1: Common practices: (1) The initial "burn-in" stage in the chain segment ***shall be discarded***, as they may have not yet reach "equilibrium"; (2) Run many chains to compose a very large sample to minimize bias!

Note 2: MCMC is a sampling scheme - it provides an efficient way to draw a sample with a given probability distribution, thus can be used to assist advanced integration etc. It is not an optimization procedure. However, when combined with optimization algorithms, it can be used for parameter estimation and to constrain confidence intervals in model regression.

Note 3: same elements can exist multiple times in the chain. Each element is also in fact *highly correlated* with previous elements! Therefore only the collection of this long chain makes sense, following the desired distribution. One has to *randomize* the collection in the end in order to truly generate a random sequence which follows a given distribution as *RNG* generates!

e.g., … 2, 1, 3, 4, 4 , 5, 6, 5, 6, 7, 7 , 8 ,7, 8, 9, …
Among the 14 numbers in the chain above, the sequence has only 10 unique number, and correlation exists among them.

Monte Carlo Markov Chain



Now combining with what we learned last two weeks, we can use MCMC to sample a likelihood function or a posterior, the sampling would allow us to find maximum and credible range of parameters!

A MCMC chain would converge but it is impossible to test if a MCMC sampler has converged towards its (global) equilibrium distribution or a temporary meta-stable distribution.

Depending on the proposal scheme and initial guess, MCMC samplers may get trapped in local "meta-stable" distribution for a long time before escaping. Many proposals aim at escaping local minima.

Various MCMC sampling methods to generate a chain:

- Original Metropolis-Hastings sampling: e.g., PyMC.
- Updated M-H sampling from Goodman & Weare (2010): e.g., emcee.
- Hamiltonian Monte Carlo sampling: gradient-based sampling scheme, i.e., specifying $\log p(\vec{\theta})$ but also its gradient. Advantage: short burn-in stage, more efficient than M-H sampling: e.g., pyhmc, PyTorch - hamiltorch
- Gibbs sampling (through conditional probabilities, pair updating).

Applications of Markov Chain

Let $p(\vec{\theta}) \equiv p(\vec{\theta}\,|D,I) \propto p(D|\,\vec{\theta},I)\,p(\vec{\theta}\,|I)$

Many estimates are integrals of the following form:

$$I = \int g(\vec{\theta})\,p(\vec{\theta})\,\mathrm{d}\vec{\theta}$$

- According to the *posterior* $p(\vec{\theta})$ to sample a Markov chain of a total of N elements (excluding initial burn-in steps), each member in the chain is given by $\vec{\theta}_m = (\theta_1, \theta_2, \ldots \theta_k)_m$.

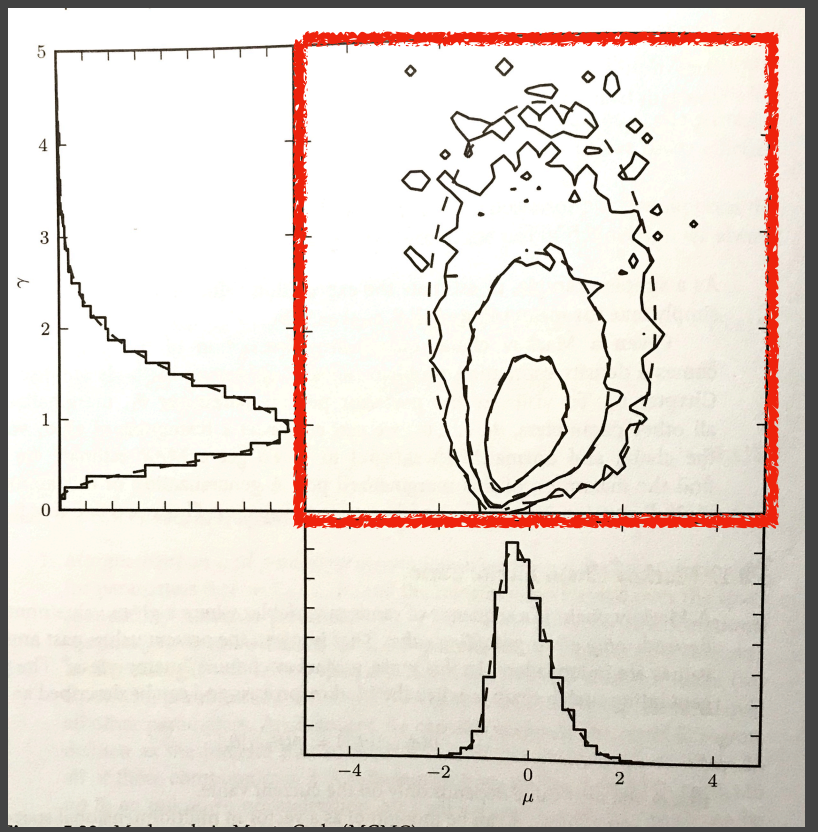
- To obtain the posterior *mean* for θ_i:

$$\overline{\theta_i} = \int \theta_i\,p(\vec{\theta})\,\mathrm{d}\theta_1\,\mathrm{d}\theta_2 \ldots \mathrm{d}\theta_k$$

i.e., $g(\vec{\theta}) = \theta_i$, then the integral is given by:

$$I = \frac{1}{N}\Sigma_{m=0}^{N}\overset{\vec{x}\sim p(\vec{x})}{[p]}\,g(\vec{\theta}_m) = \langle g\rangle = \langle\theta_i\rangle$$

$\ln[p(\mu,\gamma\,|\,\{x_i\},I)] = \text{const} + (N-1)\ln\gamma - \Sigma_{i=1}^{N}\ln[\gamma^2 + (x_i-\mu)^2]$



Joint posterior from $N = 10$ measurements of $\{x_i\}$, which was generated to follow a Cauchy distribution with $\mu = 0, \gamma = 2$. Bayesian inference with flat priors on μ, γ.

Applications of Markov Chain

Let $p(\vec{\theta}) \equiv p(\vec{\theta}\,|D,I) \propto p(D|\,\vec{\theta},I)\,p(\vec{\theta}\,|I)$

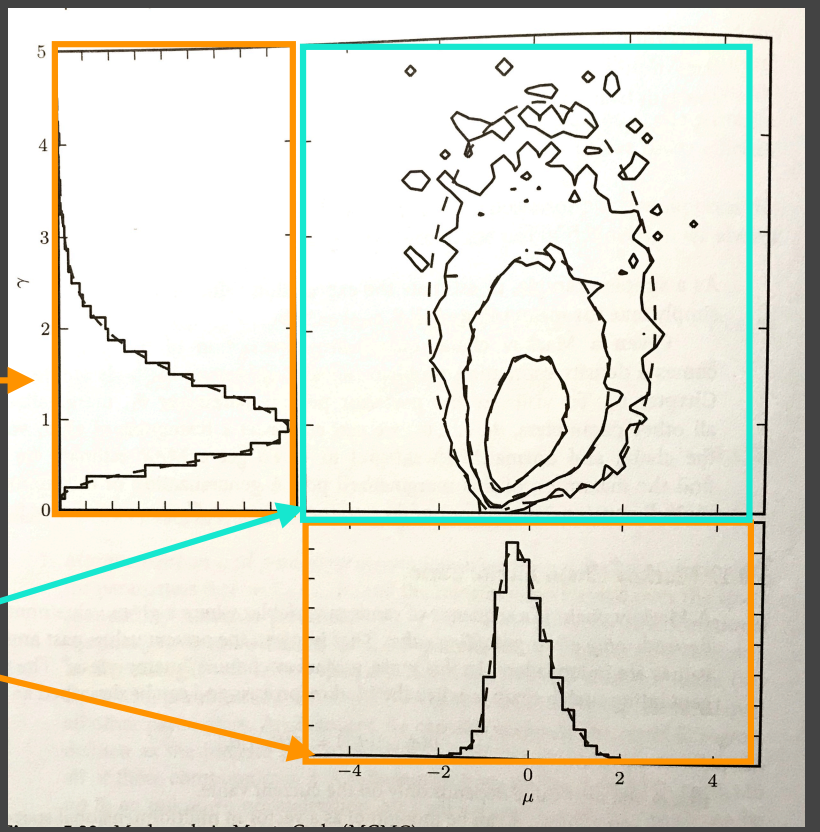
Many estimates are integrals of the following form:

$$I = \int g(\vec{\theta})\,p(\vec{\theta})\,\mathrm{d}\vec{\theta}$$

- To obtain the marginalized posterior of θ_i, one simply compute the *histogram* of θ_i using all members (after initial burn-in) in the chain, through binning the number counts or kernel estimation in the relevant parameter space.

- To obtain the covariance between any two parameters θ_i and θ_j (i.e., corner-plot), one simply takes the two relevant parameters of the chain members and plot contours.

$\ln[p(\mu,\gamma\,|\,\{x_i\},I)] = \mathrm{const} + (N-1)\ln\gamma - \Sigma_{i=1}^{N}\,\ln[\gamma^2 + (x_i-\mu)^2]$



Joint posterior from $N = 10$ measurements of $\{x_i\}$, which was generated to follow a Cauchy distribution with $\mu = 0, \gamma = 2$. Bayesian inference with flat priors on μ, γ.

Last Week

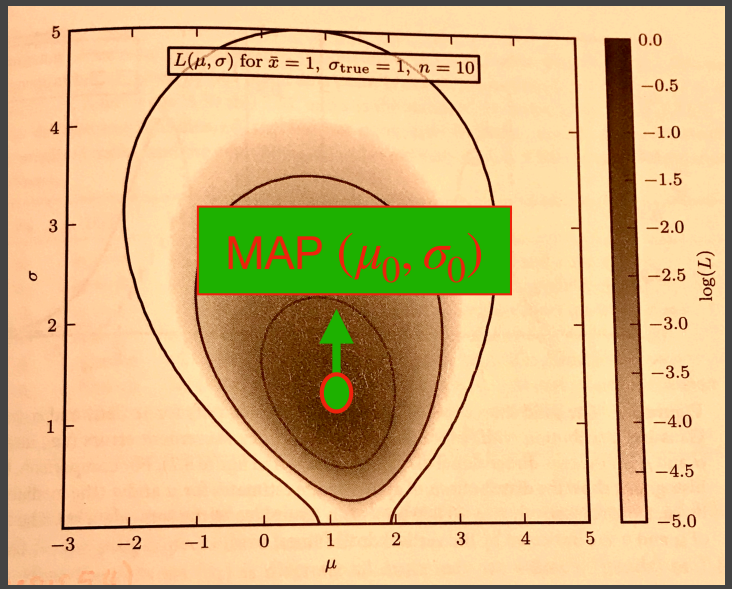
Bayesian parameter estimate and uncertainty

Case III: measurement error $\{e_i\}$ is known but σ is ***unknown***, we seek for a 2d posterior pdf $p(\mu, \sigma \,|\, \{x_i\}, \{e_i\})$.

Data likelihood $$p(\{x_i\}, \{e_i\} \,|\, \mu, \sigma, I) = \Pi_{i=1}^{N} \frac{1}{\sqrt{2\pi}(\sigma^2 + e_i^2)^{1/2}} \exp\left(\frac{-(x_i - \mu)^2}{2(\sigma^2 + e_i^2)}\right)$$

Prior of μ, σ: $$p(\mu, \sigma \,|\, I) = \text{const, for } \mu_{\min} \leqslant \mu \leqslant \mu_{\max} \text{ and } \sigma_{\min} \leqslant \sigma \leqslant \sigma_{\max}$$

$$L_p = \ln[p(\mu, \sigma \,|\, \{x_i\}, \{e_i\}, I) = \text{constant} - \frac{1}{2}\Sigma_{i=1}^{N}\left(\ln(\sigma^2 + e_i^2) + \frac{(x_i - \mu)^2}{(\sigma^2 + e_i^2)}\right)$$



Requesting $\left(\mathrm{d}L_p/\mathrm{d}\mu\right)|_{(\mu=\mu_0)} = 0$

$$\mu_0 = \frac{\Sigma_{i=1}^{N}\omega_i x_i}{\Sigma_{i=1}^{N}\omega_i} \qquad \omega_i = \frac{1}{(\sigma_0^2 + e_i^2)}$$

Requesting $\left(\mathrm{d}L_p/\mathrm{d}\sigma\right)|_{(\sigma=\sigma_0)} = 0$

$$\Sigma_{i=1}^{N}\frac{1}{\sigma_0^2 + e_i^2} = \Sigma_{i=1}^{N}\frac{(x_i - \mu_0)^2}{(\sigma_0^2 + e_i^2)^2}$$

One can go for Solving posterior MAP iteratively!

Alternatively one can sample the joint posterior $p(\vec{\theta} = \{\mu, \sigma\})$ *with MCMC.*

This Week

Bayesian parameter estimate and uncertainty

Case Extreme: measurement errors $\{e_i\}$ are also unknown (N+2 parameters)!

we seek for a 2d posterior pdf (covariance) of $p(\mu, \sigma \,|\, \{x_i\}, I)$

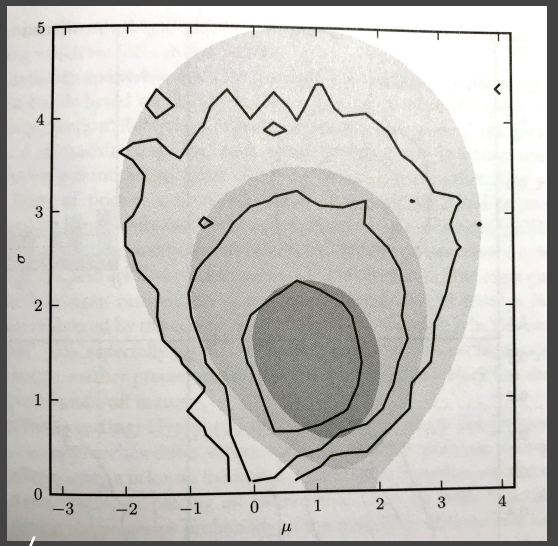
by marginalizing the joint posterior pdf of $p(\mu, \sigma, \{e_i\} \,|\, \{x_i\}, I)$ over $\{e_i\}$

Data likelihood $p(\{x_i\}, |\mu, \sigma, \{e_i\}, I) = \Pi_{i=1}^{N} \frac{1}{\sqrt{2\pi}(\sigma^2 + e_i^2)^{1/2}} \exp\left(\frac{-(x_i - \mu)^2}{2(\sigma^2 + e_i^2)}\right)$

Flat prior of μ, and

scale-invariant prior for σ and $\{e_i\}$

Now straightforward to compute using MCMC!



Let's look at a simple python example using Jupyter notebook…
Also see Python Package Index for useful basic information, https://pypi.org/

- ★ Download anaconda at: https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/
- ★ Install anaconda
- ★ In terminal, type "ipython notebook" to launch jupyter notebook
- ★ An example application of *emcee* python script is given with the class, for more see https://emcee.readthedocs.io/en/stable/index.html

```
# One way to have some initial guess of model parameters (always needed):
# Run optimization in order to obtain a reference point to generate some
# initial guess for parameters in MCMC chains in later steps.
import scipy.optimize as opt
# any reasonable initial guess for 4 parameters in optimization procedure:
param_init=[1.0,1.0,1.0,1.0]
# nll is the negative log likelihood -- to be minizied:
nll= lambda *args: -lnlike(*args) # settings in opt.minimize...
result_opt = opt.minimize(nll, param_init, args=(x_true, y_true, z_obs))
print("optimazation:", result_opt['x'])
print("true value:",alpha_true, beta_x_true, beta_y_true, eps_true)
```

```
optimazation: [4.91230561 1.00815157 9.99252593 0.98342451]
true value: 5.0 1.0 10.0 1.0
```

```
# Now let's prepare many walkers/chains (to later sample the distribution)!
# It is unadvisable to use fewer walkers than twice the number of parameter-space dimensions
# Nparam = 4 here, suggested Nwalkers shall > 2 Nparam (= 8)
Nwalker,Ndim = 30,Nparam
# Set up (slightly different) initial parameter guess p0 for different walker/chain!!
rel_error_InitGuess=0.1   # relative (fractional) uncertainty for inital parameter guess around optimization results
#p0 = [param_true*(1+random.uniform(-rel_error_InitGuess,rel_error_InitGuess, size=Nparam)) for i in range(Nwalker)]
p0 = [result_opt['x']*(1+random.uniform(-rel_error_InitGuess,rel_error_InitGuess, size=Nparam)) for i in range(Nwalker)]
#p0 = [result_ls.x*(1+random.uniform(-rel_error_InitGuess,rel_error_InitGuess, size=Nparam)) for i in range(Nwalker)]
```

We know that MCMC is expected to fully sample the constructed (equilibrium) probability distribution $p(\vec{\theta})$, i.e., MCMC essentially samples the $\chi^2 = -\ln p(\vec{\theta})$ surface (for a Gaussian $p(\vec{\theta})$). Practically, we can first run an optimization procedure to get some initial guess for MCMC. However, there might be many local minima of χ^2 surface. Although a single chain might be able to jump between different local minima if it is long enough and with efficient sampling/update scheme, in order to maximize the chance to broadly sample the parameter space (to capture as many local minima as possible), we want to exploit multiple chains, each starting from a some different positions in the parameter space.

Model selection through Bayesian Evidence

Applications of Markov Chain

Normalized posterior distribution: $p(\vec{\theta}\,|\,D,M,I) = \dfrac{p(D\,|\,\vec{\theta},M,I)\,p(\vec{\theta}\,|\,M,I)}{p(D\,|\,M,I)}$,

where $E(M) \equiv p(D\,|\,M,I) = \int p(D\,|\,\vec{\theta},M,I)\,p(\vec{\theta}\,|\,M,I)\,\mathrm{d}\vec{\theta}$ is the model ***evidence***.

MCMC can be used to calculate $E(M)$, with which we can (1) normalize the posterior $p(\vec{\theta})$ given model M; (2) compare between models using Bayes factor $B_{21} = \dfrac{E(M_2)}{E(M_1)}$

- Let's define unnormalized posterior distribution: $p'(\vec{\theta}) \equiv p(D\,|\,\vec{\theta},M,I)\,p(\vec{\theta}\,|\,M,I)$
- Sample a chain of $\vec{\theta} = (\theta_1, \theta_2, \ldots \theta_k)$ according to $p'(\vec{\theta})$ [Note: through the *Hasting ratio*, the chain built-up process does not depends on normalization!]
- The "steady" chain has a number density $\rho(\vec{\theta})$, which satisfy $\int \rho(\vec{\theta})\,\mathrm{d}\vec{\theta} = N$,

  N being the total number of elements in the chain; while $\rho(\vec{\theta})$ can be calculated using the chain elements through various *density estimation methods.*

Importantly, $\dfrac{p'(\vec{\theta})}{E(M)} = \dfrac{\rho(\vec{\theta})}{N}$, therefore $E(M) = \dfrac{N\,p'(\vec{\theta})}{\rho(\vec{\theta})}$.

Note: for N sets of $\vec{\theta}$ in the chain, we essentially have N estimators of $E(M)$!

Monte-Carlo Method

1. **What is MC method?**

   To use randomness to solve deterministic problems!

2. **Key technique:**

   Random Number Generator

3. **Main applications:**

   3.1 Numerical integration (basic)

   3.2 Generating draws with a given probability distribution

   3.3 Advanced integration and Bayesian inference (MCMC)

Importance sampling

Metropolis-Hasting algorithm

Mont Carlo Markov Chain

More application of MC sampling and MCMC…

Combine importance sampling with Markov Chain

Hierarchical Bayesian Inference

Care for hyper-parameters

$$p(\alpha \mid D) = \frac{\int p(D \mid \alpha, \theta) p(\theta \mid \alpha) \, \mathrm{d}\theta \, p(\alpha)}{p(D)}$$

Care for direct model parameters

$$p(\theta \mid D) = \frac{p(D \mid \theta) \int p(\theta \mid \alpha) \; p(\alpha) \; \mathrm{d}\alpha}{p(D)}$$

In either case, in order to carry out (hyper-) parameter estimates, the posterior needs to be sampled using *MCMC*. But in order to build this chain, the posterior shall be evaluated given α, θ and D, the calculation including an integration over all possible θ or α.

This integration can be done through: ***Importance sampling***

Hierarchical Bayesian Inference

Survey of gravitationally-lensed objects in HSC imaging (SuGOHI). III. Statistical strong lensing constraints on the stellar IMF of CMASS galaxies

Show affiliations

https://arxiv.org/pdf/1904.10465.pdf

Sonnenfeld, Alessandro ; Jaelani, Anton T. ; Chan, James ; More, Anupreeta ; Suyu, Sherry H. ; Wong, Kenneth C. ; Oguri, Masamune ; Lee, Chien-Hsiu

Constraining properties (parameterized with α) of a population instead of individual systems (parameterized with θ).

Importance sampling in Hierarchical inference

$$p(\alpha \mid D) = \frac{\int p(D \mid \alpha, \theta) p(\theta \mid \alpha)\, \mathrm{d}\theta\, p(\alpha)}{p(D)}$$

$$I = \int_{\nu_\theta} g(\theta) f(\theta)\, \mathrm{d}\theta = \int_{\nu_F} g(\theta)\, \mathrm{d}\,[F]$$

Uniform in $F(\theta) = \int_{-\infty}^{\theta} f(\theta')\mathrm{d}\theta'$

$$I \approx I_N^{[F]} = \frac{1}{N} \Sigma_i^N g(\theta_i^{[F]}) \quad \theta \sim f(\theta)$$

Sum $g(\theta_i^{[F]})$ over all N element of θ that follows $f(\theta)$.

Note: normalization $\int_{-\infty}^{\infty} f(\theta)\mathrm{d}\theta = 1$ is important here!

(1) make $g(\theta) = p(D \mid \alpha, \theta)$ and make $f(\theta) = p(\theta \mid \alpha)$;

(2) Draw a sample of $\{\theta_i^{[F]}\}$ according to $f(\theta) = p(\theta \mid \alpha)$;

(3) Approximate I with summation of $g(\theta) = p(D \mid \alpha, \theta)$ using the sampling points.