ECE 4960

Spring 2017

Lecture 18

Nonlinear Equations and Optimization:

Monte Carlo Sampling

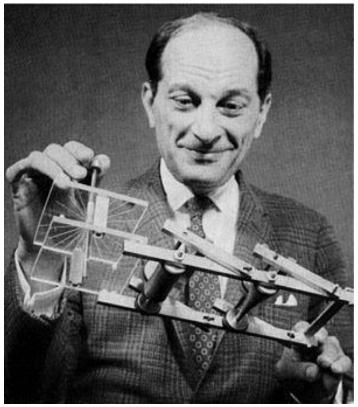
Edwin C. Kan
School of Electrical and Computer Engineering
Cornell University

Why Random and then Statistical?

- Nonlinear solution or optimization often needs an “initial guess”, which affects the eventual solution.
- Often we need to sample the parameter space for initial guess, but it can be still too expensive.
- 30 parameters in least-square fitting with 4 possible samples in each parameter will have 4^{30} or 10^{18}!
- A solitaire card game has to search the sequence of 52 cards, or 10^{52}, about 10^{68}!!!
- How can we sample a huge space and have an idea of what has been covered? This is a problem that has puzzled physicists and mathematians for a long time without a solution, until...

Monte Carlo Out of Manhattan Project

- Stanislaw Ulam is in charge of investigating the reliability of a nuclear reactor. The problem is too complex, and too important not to know the error.
- He was in hospital playing solitaire and tried different sampling to predict the group of outcomes.
- He then presented the initial idea to John von Neumann. Together they invented a new method with code name: Monte Carlo.



Stanislaw Ulam
(1909 – 1984)

John von Neumann
(1903 – 1957)

Large Number of Sampling or Testing

- Let *v* be a random variable (now *v* is just a scalar, but later on it can be generalized to a vector with a large rank) whose distribution density function *p(v)* is **unknown**.
- We can denote E(*v*) = *A* and var(*v*) = σ^2, which we do not know their values either. A Monte Carlo sampling of *v* is represented by a group of *N* random sampling of v_k by defining:

$$\hat{A}_N = \frac{1}{N}\sum_{k=1}^{N} v_k; \qquad R_N = \hat{A}_N - A$$

- R_N is called the **bias** of the present estimator of *N* sampling.

Monte Carlo Simulation

- By the Central Limit Theorem and the Large-Number Theorem:

1. E(R_N) = 0,
   - Errors of v_k is random and equally above and below . The choice of these (v_1, v_2, …, v_N) that satisfies this property is called a consistent estimator.
2. var(R_N) = σ^2/N,
   - when the expected variance will become smaller with increasing N.
3. With $N \rightarrow \infty$, R_N approaches a standard normal distribution function $\mathcal{N}(0,1)$ with zero mean and unit standard deviation:

$$p_{RN}(v) \cong \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{v^2}{2}\right)$$

Main Properties of Monte Carlo

- Even we do not know A, σ or $p(v)$, when we have N samples, we can estimate A by $E(R_N) = 0$ and estimate σ^2 by $\mathrm{var}(R_N) = \sigma^2/N$, i.e., we have an idea of the **error bar**.
- We can think that the distribution of R_N is approximately:

$$\sigma Z / \sqrt{N}$$

where $Z \sim \mathcal{N}(0,1)$, i.e., the “bias” (or error) in the Monte Carlo testing is of the order of $N^{-0.5}$ with a prefactor σ (the variance of the problem under study)!

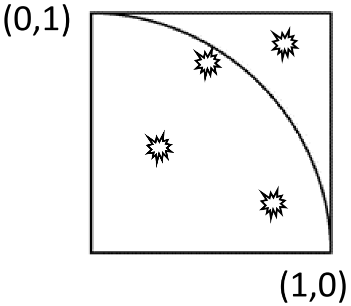
- Power of Monte Carlo: as long as we can do N testings, we can tackle a problem we know little about its behavior.
- Limitation of Monte Carlo: the statistical error is $\propto N^{-0.5}$, i.e., making N to be four times larger can only improve the accuracy by 2 times, worse than even bisection.

Improvement in Monte Carlo Sampling

- We can indeed improve the way we choose samples in Monte Carlo to make better accuracy improvement.
- This is generally called the **variance reduction** in Monte Carlo,
  - Control variates: Know the mean of the original problem by physical laws or by a similar problem.
  - Antithetic variates: Know another *p(v)* that has the same statistical properties of the problem under study (they are called symmetry pairs).
  - Importance sampling: Know the likelihood of the distribution function.
  - Statistical amplification: Explore more rare cases by give statistical weights.

Program Practice

Calculating π by Monte Carlo: We know the area of a quarter circle is π/4 (and assume that we do not know the value of π). We can transform the calculation into a statistical sampling by estimating the probability of random distributions in a square to be within the circle:



```
double x[i] = random( );
double y[i] = random( );
integer count = 0;
for i = 1, N; i++
      if ( x[i]*x[i] + y[i]*y[i] < 1 ) count++;
double pi = 4*count/N;
```

Now observe your error in estimation for N = 10, 100, , … 10^6.
If you have your power-law fitting still, plot out the N vs. error.

Observation

- To calculate π correctly, *x* and *y* need to be uniformly distributed between (0, 1). Any bias in *x* and *y* will become a bias in the estimate of π. You can see them as distortion in the geometry!
- We can enlarge the search domain without changing the answer. More random points will be wasted. This means the choice of sampling can affect how fast the convergence will be (the prefactor σ we discussed before).
- The precision of π will be proportional to from our previous theory.

Random Number Generation

- Most programming languages support `random()`, which ideally should have the following properties:

  1. Uniformly distributed between [0, 1].
  2. Mean = 1; Variance = 1/12.
  3. Any segment of number sequence will have no correlation with the other.
  4. The sequence is unpredictable, or the sequence has no memory effect.

- Most such functions are only pseudo random numbers
- C/C++ uses a seed (often the program counter as an integer):

```
    x[i+1] = (a*x[i] + c) mod m;
    return double x[i+1]/m;
```

where `a, c` and `m` are large prime numbers.

Pseudo Random Numbers

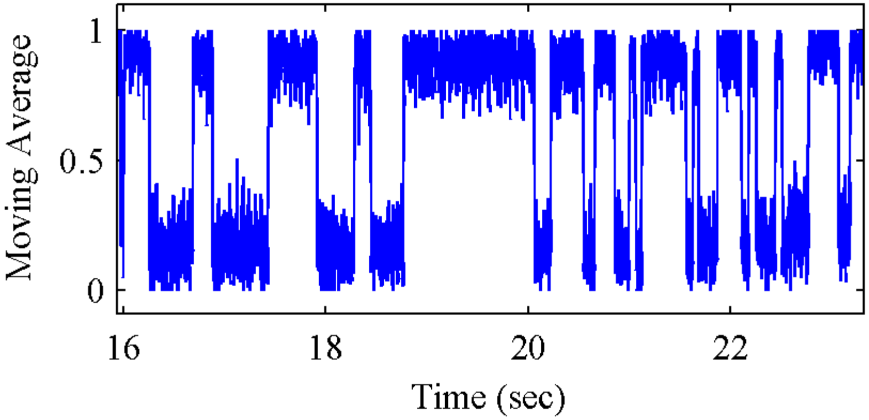
“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number — there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.”

- John von Neumann

John von Neumann
(1903 – 1957)

True Random Numbers

- Thermal noises (vulnerable to nitrogen spraying),
- Single-particle noises (aka random telegraph noises, which may not have a uniform spectrum)
- Quantum noises (based on uncertainty principles)
- De-biasing to obtain a uniform distribution.



Random Telegraph Noise Raw Data

```
do {
    For each up/down-time in raw data
      Output = LSB(up/down-time);
      Shift right up/down-time by one bit;
    End for
} repeat until all up/down time are zero;

Perform von Neumann de-biasing
```

Convert to binary random sequence

Random Numbers with a Given Distribution

- For many sampling problems, we may need a random number that follows a given distribution, instead of uniformly distributed.
- For example, when random numbers are used to sample a gas molecule velocity, we know that *v* would follow the thermodynamic law and has a Boltzmann distribution.
- Transformation from a uniformly distributed variable to another distribution function is a straightforward mathematical procedure.
- Denote the probability density function of a random variable *v* as *p(v)*, and the corresponding cumulative distribution function as *F(v)*,

$$F\left(v\right)=\int_{-\infty}^{v} p(x)dx$$

Cumulative Distribution Function (CDF)

- We know the follow properties of *p(v)* and *F(v)*.

1. $0 \leq p(v), F(v) \leq 1; \qquad \forall v$

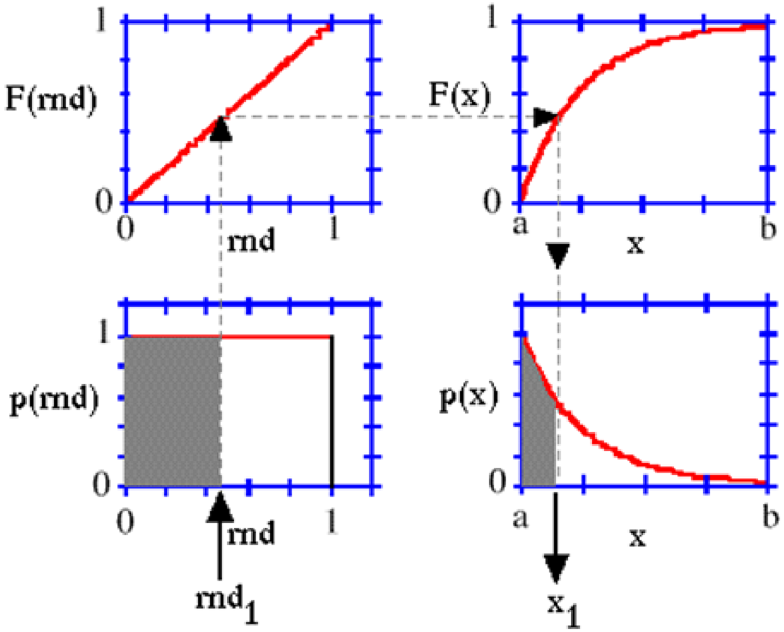
2. $\lim_{v \to -\infty} F(v) = 0; \qquad \lim_{v \to \infty} F(v) = 1;$

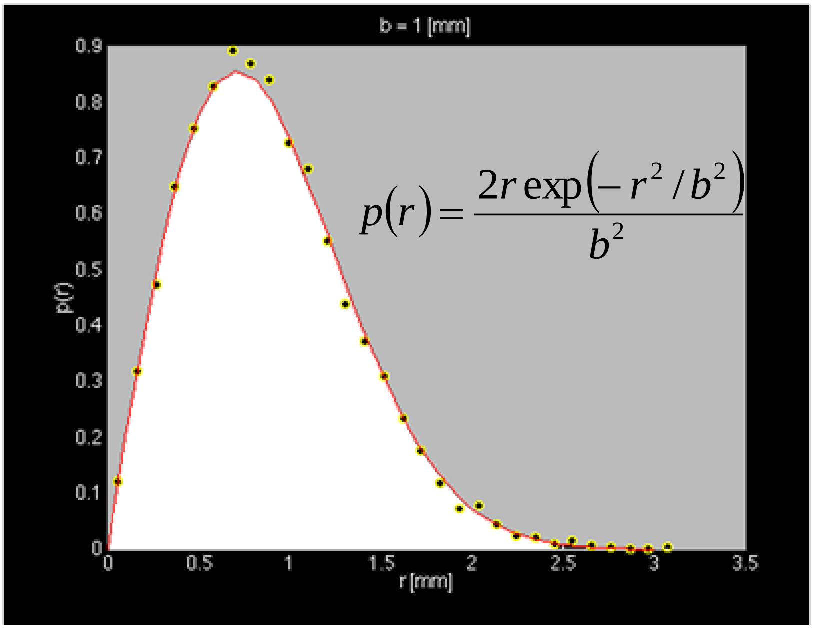
3. *F(v)* is a monotonically increasing function

Transformation of Distribution Functions

- From a uniformly distributed random variable in [0, 1] generated by random():

1. Call random() to obtain a random number *u* between [0, 1].
2. We know that *F(u) = u*.
3. Find *v* that gives *F(v) = u*, or *v = F*$^{-1}$*(u)*. The random variable *v* will now follow *p(v)*.





Program Practice

- To generate *v* with distribution function of *p(v)* from `random()`
- Use $\lambda = 0.2$.

$$p(v) \quad \begin{cases} = \lambda e^{-\lambda v} & v \ge 0 \\ = 0 & v < 0 \end{cases}$$

$$F(v) = \int_0^v \lambda e^{-\lambda x} dx = 1 - e^{-\lambda v} = u$$

Known Closed-form of CDF!!

$$v = -\frac{1}{\lambda} \ln(1-u)$$

- Generate 1,000 instances of *v* and sort the vector *v*. Plot *p(v)* and *F(v)* for $0 \le v < 10$ with a bin size of 0.5.
- Beside visualization, is there another way for verification?

CDF Does Not Have Closed Forms

- To generate *v* with distribution function of *p(v)* from `random()`

$$p(v) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{v^2}{2} \right)$$

Unknown Closed-form of CDF!!

$$F(v) = \int_{-\infty}^{v} p(x)dx$$

- Create a table to find $F^{-1}(u) = v$.
  - Numerical integration
  - Interpolation from local analysis

Confidence Interval

- Monte Carlo sampling is to generate *v* from the random number function with the knowledge of *F(v)*.
- We do not need to know the closed form of *F(v)*, but just a way to **compute** $F^{-1}(u)$ from formula, table lookup or rejection rules.
- After obtaining *N* samples of *v*, we can use them to understand the moment, the physical phenomena governed by *v*, or simply to evaluate *A* (1st moment) and σ (2nd moment).
- As long as *N* is sufficiently large, we know our *N* samples will give us a reasonable estimate, as R_N will approach a standard normal distribution function.
- The 67% **confidence interval** will be: $\left[\hat{A}_N - \frac{\sigma}{\sqrt{N}}, \hat{A}_N + \frac{\sigma}{\sqrt{N}}\right]$
- The 99.7% confidence interval will be: $\left[\hat{A}_N - \frac{3\sigma}{\sqrt{N}}, \hat{A}_N + \frac{3\sigma}{\sqrt{N}}\right]$

Monte Carlo for Initial Guess Generation

- For optimization on parameter extraction, when we need an initial guess in a large parameter space (say *m*= 20), we will use 20 uniformly generated random numbers to be transformed to the known or guessed distribution function of each parameter. That would serve as one initial guess.
- We can repeat the process for *N* = 200 times to obtain a reasonable confidence that our minimal can be close to the global minimum.
- The quality of the initial guess (or the coverage of the confidence interval) will depend on our understanding of the distribution function of each parameter.

Monte Carlo for Optimization Search: Simulated Annealing

- We can use the Monte Carlo procedure to the optimization search process directly.
- The most famous example is the **simulated annealing**, which resembles the natural process of relaxation to the energy minimum.
- For example, we know that diamond is the lowest-possible energy form for the group of carbon atoms.
- However, in many natural scenarios, the group of carbon atoms will just become a local minimum of soot (such as the product of burning, where the product is quenched quickly).
- To form diamond, the nature would have a high temperature first to allow the atoms to go to their preferred position (actually a high pressure is needed as well, so that they do not go too wildly), and then gradually cool down to formulate diamond.

Optimization by Simulated Annealing

- A correction step is determined by the Newton method or the steepest descent method,
- Use the line search method to determine *t* for the most improvement possible in this step
- When no improvement can be found in all searched *t*, instead of declaring the minimum has been found, we will take the step with the size of *t* anyway according to the probability function:

$$p(t) = \exp\left(-\frac{\Delta V}{T} \right)$$

$$\Delta V \equiv \left\| V\left(\vec{x}^{(k)} + t\Delta\vec{x}^{(k)}\right) - V\left(\vec{x}^{(k)}\right) \right\|_2$$

- The value of *t* will be determined by the Monte Carlo sampling with the distribution function *p(t)*.

Temperature in Simulated Annealing

- The temperature parameter *T* is analogous to the natural annealing process.
  - When *T* is very large, large penalty on *ΔV* has a higher probability to be taken.
  - When *T* is nearly zero, only very small *ΔV* will have reasonable probability.
  - For a reasonable search of the global minimum, *T* can be initially large, and then gradually becomes small during the final quenching process.
- Simulated annealing has been broadly applied to problems that have very large parameter space and possibly many local minima (aka, mixed determined or ambiguity).
- For many problems such as chess, the strategy of setting *T* can be learned from the existing examples to reduce the searching time. This machine learning strategy is for sure left to students who will take the specific course in artificial intelligence or machine learning.