Scientific Computing with Case Studies

SIAM Press, 2009
http://www.cs.umd.edu/users/oleary/SCCSwebpage

Lecture Notes for Unit IV

Monte Carlo Computations

Dianne P. O'Leary

What is a Monte-Carlo method?

In a Monte-Carlo method, the desired answer is formulated as a quantity in a stochastic model and estimated by random sampling of the model.

Example:

- If we have a cube, with the sides numbered 1 to 6, we might toss the cube 120 times, observe which side comes up on top each time, and study whether the sides occur with approximately equal frequency.
- If we have a "black box" that takes a number between 0 and 1 as input and emits a number between 0 and 1 as an output, we could feed the box m numbers and observe the m outputs of the box and use the average of the observations as an estimate of the statistical mean of the process defined by the black box. []

Two basic principles

- There is an important difference between
  - Monte Carlo methods, which estimate quantities by random sampling, and
  - pseudo-Monte Carlo methods, which use samples that are more systematically chosen.

  In some sense, all practical computational methods are pseudo-Monte Carlo, since random number generators implemented on machines are generally not truly random. So the distinction between the methods is a bit fuzzy. But we'll use the term Monte Carlo for samples that are generated using pseudorandom numbers generated by a computer program.
- Monte Carlo methods are (at least in some sense) methods of last resort. They are generally quite expensive and only applied to problems that are too difficult to handle by deterministic (non-stochastic) methods.

The Plan:

- Basic statistics: Random and pseudorandom numbers and their generation: Chapter 16.
- Monte Carlo methods for numerical integration: Chapter 18.

- Monte Carlo methods for optimization: Chapter 17.
- An example of Monte Carlo methods for counting: Chapter 17.

Basic statistics: Random and pseudorandom numbers and their generation

- What is a random number?
- What are the mean and variance of a random sample?
- What is a distribution? What are its mean and variance?
- How are pseudorandom numbers generated?

Examples of how to generate random numbers

- Take n papers and number them 1 to n. Put them in a box, and draw one at random. After you record the resulting number, put the paper back in the box. You are taking random numbers that are uniformly distributed among the values $\{1, 2, \ldots, n\}$.
- Make a spinner by anchoring a needle at the center of a circle. Draw a radius line on the circle. Spin the needle, and measure the angle it forms with the radius line. You obtain random numbers that are uniformly distributed on the interval $[0, 2\pi)$.
- There are printed tables of random numbers.
- If, on average, a radioactive substance emits α-particles every μ seconds, then the time between two successive emissions has the exponential distribution with mean μ. (Note: This is a special case of the Gamma distribution.)

- The normal distribution is a good model in many situations:
  - The pattern of leaves that fall from a symmetric tree
  - The IQ measure of intelligence was constructed so that the measures are normally distributed.

    `http://www.awl.com/weiss/e_iprojects/c06/chap06.htm.`
  - Physical characteristics of plants and animals (height, weight, etc.).
  - velocity distribution of molecules in a thermodynamic equilibrium (Maxwell-Boltzmann distribution)

    `http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/disfcn.html`
  - Measures of psychological variables such as reading ability, introversion, job satisfaction, and memory.

    `http://davidmlane.com/hyperstat/normal_distribution.html`

– First studied by DeMoivre (1667-1754), in connection with predicting the probabilities in gambling.

Properties of samples from random distributions

The mean or average value or expected value of a set of samples $\{x_i\}$, $i = 1, \dots, n$, is

$$\mu_n \equiv \frac{1}{n} \sum_{i=1}^{n} x_i$$

and the sample variance is

$$\sigma_n^2 \equiv \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2 \,.$$

If the mean μ is unknown, then one estimate is

$$\sigma_n^2 \equiv \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu_n)^2 \,.$$

Properties of distributions

A random distribution is characterized by a probability density function $f(x)$.

- The domain of the function is the set of possible values that could be obtained by taking random samples of the function. Call this domain Ω.
- The range of the function excludes $[-\infty, 0)$.
- The value

  $$\int_\alpha f(x)\,\mathrm{d}x$$

  is the probability that a sample from this distribution is in the set defined by α. (Replace the integral by a summation if the domain is discrete.)
- Because of these properties, we see that

  $$\int_\Omega f(x)\,\mathrm{d}x = 1\,.$$

  (Or the sum of the probabilities is 1 in the discrete domain case.)

The mean of a distribution and variance of the distribution is the average value you would expect to get for the sample mean and variance if you took a large number of very large sets of samples. They can be computed by

$$\begin{aligned} \mu &= \int_\Omega x f(x)\,\mathrm{d}x \\ \sigma^2 &= \int_\Omega (x-\mu)^2 f(x)\,\mathrm{d}x \end{aligned}$$

Example distribution functions

The uniform distribution over the interval $[0, m]$ has

$$f(x) = \frac{1}{m}$$

Mean:

$$\mu = \int_0^m \frac{x}{m}\,\mathrm{d}x = \frac{m}{2}$$

Variance:

$$\sigma^2 = \int_0^m \frac{1}{m}\left(x - \frac{m}{2}\right)^2 \mathrm{d}x = \frac{m^2}{12}$$

The exponential distribution with parameter μ has

$$f(x) = \frac{1}{\mu}e^{-x/\mu}$$

for $x \in [0, \infty)$.

Mean: μ

Variance: μ^2

The normal distribution with parameters μ and σ: has

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-\mu)^2/(2\sigma^2)}$$

for $x \in (-\infty, \infty)$.

Mean: μ

Variance: σ^2

The world is normal.

The Central Limit Theorem: Let $f(x)$ be any distribution with mean μ and (finite) variance σ^2. Take a random sample of size n from $f(x)$, and call the mean of the sample $\bar{x}_n$. Define the random variable y_n by

$$y_n = \sqrt{n}\frac{\bar{x}_n - \mu}{\sigma}\,.$$

Then the distribution for y_n approaches the normal distribution with mean 0 and variance 1 as n increases.

Therefore, even if we know nothing about a distribution, except that its variance is finite, we can use samples from it To construct samples from a distribution that is near normal.

Generating "random" numbers

In principle, we could generate random samples as discussed above. For example, when we wanted a uniform distribution, we could build a spinner and play with it for a while, writing down our list of samples.

In practice, when the random numbers are used in computer software, this does not work well:

- The samples are irreproducible; if someone else wanted to use the software, they would get a different result and not know whether it was because of their different sequence of random numbers or because of a bug.
- Computers run through enormous quantities of random numbers, and it is just not feasible to generate them manually.
- Sometimes we will be very unlucky and generate a long string of random numbers that are not well-distributed.

Because of this, we use pseudorandom numbers in computer software.

Pseudo-random numbers are generated by the computer using a deterministic (i.e., reproducible) procedure and appear to be random, in the sense that the mean, variance, and other properties of sequences of n samples match what you would expect to obtain from a random process.

But the pseudorandom numbers actually cycle; i.e., if you ask for a long enough sequence, you will see periodicity.

Random number generators on computers usually use a seed (a number) to determine where in the cycle to begin. Thus, if other people wanted to reproduce your results, they would simply use the same seed.

- It is fairly cheap to generate uniformly distributed pseudorandom numbers.
- Pseudo-random number generators for the uniform distribution are easy to write but it is very, very difficult to write a good one. Conclusion: don't try to write your own unless you have a month to devote to it.
- Samples from other distributions are usually generated by taking two or more uniformly distributed pseudorandom numbers and manipulating them using functions such as `sin`, `cos`, and `exp`. This is much more costly.

- A whole course could be taught on how to generate pseudorandom numbers. The book by Knuth is quite enlightening, talking about the generation of pseudorandom numbers as well as how to test whether a program is generating pseudorandom numbers that are a good approximation to random.
- The standard functions in Matlab for generating pseudorandom normal and exponentially distributed sequences are not as solid as I would like.

Example

Example 1: A simple test for random numbers. `demorand.m`

Using pseudorandom numbers

So now we know that standard software exists for generating pseudorandom numbers.

From now on, we'll abbreviate by leaving the pseudo prefix off of random numbers.

And we'll switch our focus from what random numbers are to how random numbers help us solve problems in computational science.

Monte Carlo methods for numerical integration

Monte Carlo methods for numerical integration

A motivating example

Suppose we are asked to estimate the value

$$\begin{aligned} I &= \int_0^1 \dots \int_0^1 f(x_1,\dots,x_{10})p(x_1,\dots,x_{10})\mathrm{d}x_1\mathrm{d}x_2\mathrm{d}x_3\mathrm{d}x_4\mathrm{d}x_5\mathrm{d}x_6\mathrm{d}x_7\mathrm{d}x_8\mathrm{d}x_9\mathrm{d}x_{10} \\ &= \int_\Omega f(\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x} \end{aligned}$$

Notation:

- $\mathbf{x} = [x_1,\dots,x_{10}]$.
- $\Omega = [0,1] \times ... \times [0,1]$ is the region of integration, the unit hypercube in $\mathcal{R}^{10}$. It can actually be any region, but this will do fine as an example.
- Usually $p(\mathbf{x})$ is a constant, equal to 1 divided by the volume of Ω, but we'll use more general functions p later.

We just need $p(\mathbf{x})$ to be a probability density function, so it should be nonnegative with

$$\int_\Omega p(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1\,.$$

How might we approach the problem of computing I?

Option 1: Interpolation

Fit a polynomial (or your favorite type of function) to $f(\mathbf{x})p(\mathbf{x})$ using sample values of the function, and then integrate the polynomial analytically.

For example, a polynomial of degree 2 in each variable would have terms of the form

$$x_1^{\Box}x_2^{\Box}x_3^{\Box}x_4^{\Box}x_5^{\Box}x_6^{\Box}x_7^{\Box}x_8^{\Box}x_9^{\Box}x_{10}^{\Box}$$

where the number in each box is 0, 1, or 2. So it has $3^{10} = 59,049$ coefficients, and we would need 59,049 function values to determine these.

But recall from 460 that usually you need to divide the region into small boxes so that a polynomial is a good approximation within each box.

If we divide the interval $[0, 1]$ into 5 pieces, we make 5^{10} boxes, with 59,049 function evaluations in each!

Clearly, this method is expensive!

Option 2: product rules

Some functions $f(\mathbf{x})p(\mathbf{x})$ can be well approximated by a separable function

$$f(\mathbf{x})p(\mathbf{x}) \approx f_1(x_1)f_2(x_2)\ldots f_{10}(x_{10})\,.$$

In that case we can approximate our integral by

$$I \approx \int_0^1 f_1(x_1)\,\mathrm{d}x_1 \ldots \int_0^1 f_{10}(x_{10})\,\mathrm{d}x_{10}$$

If this works, it is great, but we aren't often that lucky.

Option 3: Use your favorite 1-d method

If we have a function quad that integrates functions of a single variable, then we can use quad to compute

$$\int_0^1 g(x_1)\,\mathrm{d}x_1$$

where

$$g(z) = \int_0^1 \ldots \int_0^1 f(z, x_2, \ldots, x_{10})p(z, x_2, \ldots, x_{10})\,\mathrm{d}x_2 \ldots \mathrm{d}x_{10}$$

as long as we can evaluate $g(z)$!

But $g(z)$ is just an integration, so we can evaluate it using quad, too!

We end up with 10 nested calls to quad. Again, this is very expensive!

See Pointer 18.1.

We need another option! The methods we have discussed are either too expensive or very special-purpose.

If the function has many variables and is not well-approximated by a separable function, we need a method of last resort: Monte Carlo integration.

Monte Carlo integration

Idea:

- Generate n points $\{\mathbf{z}^{(i)}\}$ that are randomly distributed with probability density function p
  For our example integration problem, if $p(\mathbf{x})$ is constant, this requires generating $10n$ random numbers, uniformly distributed in $[0, 1]$.
- Then
  $$\mu_n = \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{z}^{(i)})$$
  is an approximation to the mean value of f in the region, and therefore the value of the integral is
  $$I \approx \mu_n \int_\Omega p(\mathbf{x})\, \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}x_3 \mathrm{d}x_4 \mathrm{d}x_5 \mathrm{d}x_6 \mathrm{d}x_7 \mathrm{d}x_8 \mathrm{d}x_9 \mathrm{d}x_{10} = \mu_n .$$

Error estimate

- The expected value of this estimate is the true value of the integral; very nice!
- In fact, for large n, the estimates have a distribution of $\sigma/\sqrt{n}$ times a normal distribution (with mean 0, variance 1), where
  $$\sigma^2 = \int_\Omega (f(\mathbf{x}) - I)^2 p(\mathbf{x})\, \mathrm{d}\mathbf{x} \, ,$$
  where Ω is the domain of the integral we are estimating and
  $$\int_\Omega f(\mathbf{x}) p(\mathbf{x}) \mathrm{d}\mathbf{x} = I \, .$$
  Note that the variance is a constant independent of the dimension d of the integration!

Example

Example 2: Monte Carlo Integration. Estimation of

$$\int_0^{\sqrt{.8}} \sqrt{.8 - x^2}\,\mathrm{d}x$$

by testing whether points in unit square are inside or outside this region.

See 1st challenge of Chapter 17. Note that the error, multiplied by the square root of the number of points, is approximately constant.

The expected value of our estimate is equal to the value we are looking for.

There is a non-zero variance to our estimate; we aren't likely to get the exact value of the integral. But most of the time, the value will be close, if n is big enough.

If we could reduce the variance of our estimate, then we could get by with a smaller n: less work!

Variance-reduction by importance sampling

Variance-reduction methods

Suppose that we want to estimate

$$I = \int_\Omega f(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

where Ω is a region in $\mathcal{R}^{10}$ with volume equal to one.

Method 1: Our Monte Carlo estimate of this integral involves taking uniformly distributed samples from Ω and taking the average value of $f(\mathbf{x})$ at these samples.

Method 2: Let's choose a function $p(\mathbf{x})$ satisfying $p(\mathbf{x}) > 0$ for all $\mathbf{x} \in \Omega$, normalized so that

$$\int_\Omega p(\mathbf{x}) = 1.$$

Then

$$I = \int_\Omega \frac{f(\mathbf{x})}{p(\mathbf{x})} p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

We can get a Monte Carlo estimate of this integral by taking samples from the distribution with probability density $p(\mathbf{x})$ and taking the average value of $f(\mathbf{x})/p(\mathbf{x})$ at these samples.

When will Method 2 be better than Method 1?

Recall that the variance of our estimate is proportional to

$$\sigma^2 = \int_\Omega \left(\frac{f(\mathbf{x})}{p(\mathbf{x})} - I \right)^2 p(\mathbf{x})\, \mathrm{d}\mathbf{x}\,.$$

so if we chose p so that $f(\mathbf{x})/p(\mathbf{x})$ is close to constant, then σ is close to zero!

Note that this requires that $f(\mathbf{x})$ should be close to having a constant sign.

Intuitively, why does importance sampling work?

- In regions where $f(\mathbf{x})$ is big, $p(\mathbf{x})$ will also be big, so there is a high probability that we will sample from these regions.
- In regions where $f(\mathbf{x})$ is small, the $p(\mathbf{x})$ will also be small, so we won't waste time sampling from regions that don't contribute much to the integral.

Example

Example 3 : Monte Carlo Integration by Importance Sampling

$$\int_0^{\sqrt{.8}} \sqrt{.8 - x^2}\, \mathrm{d}x$$

See 4th challenge of Chapter 17.

Summary of importance sampling

- Importance sampling is very good for decreasing the variance of the Monte Carlo estimates.
- In order to use it effectively,
  - we need to be able to choose $p(\mathbf{x})$ appropriately.
  - we need to be able to sample efficiently from the distribution with density $p(\mathbf{x})$.

Monte Carlo methods for optimization

- Monte Carlo methods for (continuous) optimization.
- Metropolis algorithms for (discrete) optimization.

Monte Carlo methods for continuous optimization

Monte Carlo methods provide a good means for generating starting points for optimization problems that are non-convex.

A single starting point may result in an algorithm converging to a local minimizer rather than a global one.

If we take a set of random samples over the domain of the function we are trying to minimize, we increase the probability that we will eventually get the global minimizer.

Metropolis algorithms for discrete optimization

There are many optimization problems that are simple to state but difficult to solve.

An example is the Traveling Salesperson Problem (TSP). This person needs to visit n cities exactly once, and wants to minimize the total distance traveled and finish the trip at the starting point.

To solve the problem, we need to deliver the permutation of the list of cities that corresponds to the shortest total distance traveled.

If, for definiteness, we specify the first city, then among the $(n-1)!$ permutations, we want to choose the best.

This is an enormous number of possibilities, and it is not practical to test each of them.

Example

Example 4: Traveling Salesperson Problem. Matlab provides a naive solution algorithm: randomly generate a pair of cities and interchange them if it lowers the distance. `travel.m`

Metropolis algorithm

A motivating digression: If we have a box of atoms that have been allowed to slowly cool, then the potential energy of the system will be small. (For example, if you make ice in your freezer, the crystal structure that results is one that has a lower potential energy than most of the alternatives.)

This physical process of slow cooling is called annealing.

Idea: If we want to minimize some function other than energy, can we simulate this annealing process?

Basic features of annealing

- If the temperature is high, then each particle has a lot of kinetic energy, and can easily move to positions that increase the potential energy of the system.

  This enables a system to avoid getting stuck in configurations that are local minimizers but not global ones.

- But as the temperature is decreased, it becomes less likely that a particle will make a move that gives an increase in energy.

  This enables a system to do the fine-tuning that produces an optimal final configuration.

- If we drop the temperature too fast, then the system can easily get stuck in an unfavorable configuration.

What a Metropolis algorithm for TSP might look like

1. Start with an initial ordering of cities and an initial temperature T.
2. Randomly choose two cities.
   - If interchanging those cities decreases the length of the circuit, then interchange them!
   - If not, then interchange them with a probability that depends on the amount of increase and the current temperature.
3. Decrease the temperature.

The art of the method is determining the probability function and the temperature sequence appropriate to the specific problem.

An example of Monte Carlo methods for counting

An example of Monte Carlo methods for counting

Suppose we have a lattice with $2k$ nodes. This represents a crystal structure. Each node is occupied in one of two ways:

- by a monomer which covers it.
- by a dimer, which covers it and one of its neighbors.

Each node has either a monomer or a dimer but not both.

The problem that is interesting to physicists is to tabulate the partition function, $C(j)$, the number of different arrangements of j dimers on the lattice, for $j = 1, 2, \ldots, k$.

When the lattice lies in a plane (2-d), this problem is pretty well-understood.

But for 3-d lattices, there are few closed-form solutions, and direct counting is far too expensive unless k is very small.

The best alternative is to do some sort of random sampling, and base our estimates on the results of these samples. One such algorithm is due to Kenyon, Randall, and Sinclair and looks something like this:

Start with a lattice of monomers. At each step:

- Choose an adjacent pair of nodes.
- If both nodes have monomers, add this dimer with some probability.
- If the nodes are occupied by a single dimer, delete it with some probability.
- If one node is occupied with a dimer, swap that dimer for this one with some probability.
- If it has been long enough since our last recording, then add the resulting configuration to the record.

What is long enough? We want the samples to be uncorrelated, in other words, not to depend on the previous ones.

Example

Example 5: KRS algorithm. `KRS.m` from website: one of the sample programs for Case Study: Monte-Carlo Minimization and Counting: One, Two, Too Many

---

Final Words

- Monte Carlo methods are methods of last resort, used when standard methods fail or when analysis is inadequate.
- Success depends on the pseudorandom number generator having appropriate properties.
- These methods are used in integration, minimization, simulation, and counting.