Introduction to Monte Carlo and MCMC

Andrei Afilipoaei

May 31, 2024

Introduction

- Can we use randomness to analyze or solve deterministic situations?
- We know that (in many cases) large sets of random samples will begin to approximate some kind of deterministic behavior.
- For example, flipping a fair coin enough times will approximate a 50-50 distribution of heads to tails.

- The idea for Monte Carlo methods was invented by Stanislaw Ulam, named after the Monte Carlo Casino and his uncle's gambling addiction.

Stanislaw Ulam (1909-1984)

What does Monte Carlo do?

It uses random sampling in an effort to solve a variety of deterministic problems, including:

- Creating a desired probability distribution
- Approximating unknown values
- Calculating (multidimensional) integrals

- In computing, random number selection is only pseudo-random.
- We assume that these pseudo-random generators are de facto truly random, and behave identically.
- We can 'set seed' of the pseudo-random generator

Generating Random Numbers with Desired Distribution:

- Inversion Method
- Acceptance-Rejection Method

Inversion Method:

- For density function $f(x)$, calculate the cumulative distribution function $F(x) = \int f(y)dy$.
- Take $U = F(x)$, which yields $x = Q(U) = F^{-1}(U)$.
- Generate U on uniform Distribution on $[0, 1]$, then calculate x, which will follow the desired distribution.
- This only works if $f(x)$ can be integrated, which is not always the case!

Acceptance-Rejection Method:

- This method is more feasible in some cases, as it doesn't need integration.
- Generate x and y on uniform distributions $[0, a]$ and $[0, b]$, respectively.
- If $y_i \leq f(x_i)$ for point (x_i, y_i), then accept the point, and return x_i.
- If not, then reject the point.
- If $f(x)$ is integrable, the two methods work similarly.

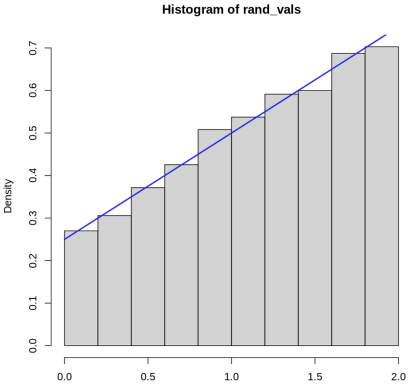
Example:

Take $f(x) = \frac{1}{4}x + \frac{1}{4}$ for $x \in [0, 2]$ and 0 otherwise.

Inversion Method:

- We see that $F(x) = \int_0^x f(y)dy = \frac{1}{8}x^2 + \frac{1}{4}x$.
- Hence,
  $U = \frac{1}{8}x^2 + \frac{1}{4}x => 8U + 1 = (x+1)^2 => x = \sqrt{8U+1} - 1$
- Generate U on uniform distribution on [0,1], and x will follow the distribution of $f(x)$.

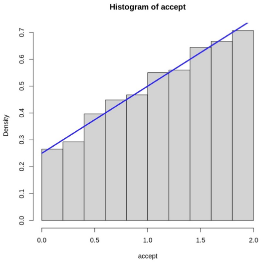
```
set.seed(316)
u <- runif(10000, min=0, max=1)
rand_vals <- -1 + sqrt(8*u+1)
# rand_vals
hist(rand_vals, prob=T, breaks=10)
dens <- function(x){1/4*x+1/4}
lines(rand_vals, dens(rand_vals), col="blue", lwd=2)
```



Acceptance-Rejection Method:

We generate random values x on $[0,2]$ and random values of y on $[0,3/4]$ (since $f(x) \leq 3/4$). If $y_i \leq f(x_i)$, accept the point and return x_i; else reject the point.

```
set.seed(316)
x <- runif(10000, min=0, max=2)
y <- runif(10000, min=0, max=3/4)
dens <- function(x){1/4*x+1/4}
accept <- ifelse(y<=dens(x),x, NA)
# accept
hist(accept, prob=T, breaks=10)
dens <- function(x){1/4*x+1/4}
lines(accept, dens(accept), col="blue", lwd=2)
```



Finding unknown deterministic value using random sampling:

- Now, we turn to finding a deterministic value using Monte Carlo random sampling.
- For this, we need the estimator to be *consistent*: i.e. T_N converges to θ *in probability*:

$$P[|T_N - \theta| \geq \epsilon] \to 0, \forall \epsilon > 0$$

- To determine the number of Monte Carlo samples needed to get a desired level of accuracy, we have the **Chebyshev** and **Hoeffding** inequalities.
- **Chebyshev Inequality:**

$$P[|T_N - \theta| \geq \epsilon] \leq \frac{Var(T_N)}{\epsilon^2} = \frac{\sigma^2}{N\epsilon^2} \leq \frac{1}{4N\epsilon^2}$$

- **Hoeffding Inequality:**

$$P[|T_N - \theta| \geq \epsilon] \leq 2e^{-\frac{2N\epsilon^2}{(b-a)^2}} \text{ when } T_N \in [a, b] .$$

- Hoeffding needs fewer samples that Chebyshev, so it is typically used.

Example: *A die is tossed 8 times. What is the probability of obtaining 3 or more ones and at most 1 six?*

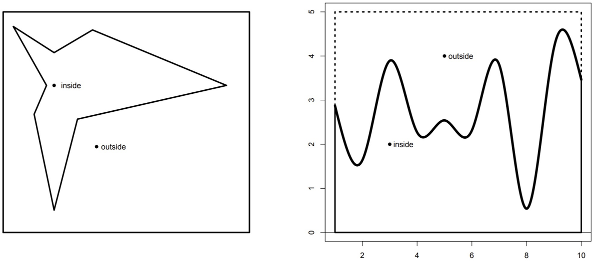
- To get 2 decimal precision with probability 0.975 or higher, we have $\epsilon = \frac{0.01}{2} = 0.005, \delta = 0.025$.
- Chebyshev takes $\frac{1}{4N*0.005^2} \leq 0.025 => N \geq 400,000$.
- Hoeffding takes $2e^{-\frac{2N*0.005^2}{(1-0)^2}} \leq 0.025 => N \geq 87,641$.
- We use $N = 87,641$ samples for Monte Carlo estimation.

```
repet <- 87641
output <- function(n){
 replicate(n, sample(x=1:6, size=8, replace=TRUE))
}
set.seed(316)
x <- output(repet)
Cond <- function(x){
 ifelse(sum(x==1)>=3 & sum(x==6)<=1,TRUE,FALSE)
}
z <- apply(x,2,Cond)
mean(z)
```

0.100603598772264

Monte Carlo Integration

- It is possible to estimate an area or integral using Monte Carlo methods.
- We know that calculating an integral is equivalent to calculating an expected value.
- As such, we can use a strategy similar to the acceptance-rejection method to estimate integrals.
- For this, the estimate must be unbiased ($E(T_N) = \theta$).



- To estimate an area, again generate random values of x and y on $[a_1, a_2]$ and $[b_1, b_2]$, respectively.
- If point (x_i, y_i) is inside the area, accept it; else reject it.

- In that case, you can calculate the area as the proportion of accepted points to total points multiplied by the large area:

$$Area = p_{accept} * (a_2 - a_1)(b_2 - b_1)$$

- Although not as efficient as the rectangular method for estimating 1-D integrals, Monte Carlo is excellent for calculating multi-dimensional integrals which would be computationally impossible with traditional methods.

Markov Chain Monte Carlo

- Now that we have established how Monte Carlo methods work, we turn our attention to Markov Chains.
- In particular, while in Monte Carlo we assumed that each point was independent of the others, in Markov Chains that is not the case.
- Instead, we incorporate a discrete time component into the random sampling process of Monte Carlo.

- Markov Chains fulfil the **Markov Property**:
- For a sequence of values $(X_1, X_2, ..., X_N)$, the stochastic outcome of X_t depends on *and only on* the value of X_{t-1}.
- The distribution of X_t is described using the *transition probability* $p_x(y) = P[X_t = y|X_{t-1} = x]$.
- The transition probabilities can be summarized in a *transition matrix*, which allows the invariant probability to be solved using linear algebra.
- **Brownian motion** is an example of Markov Chains!

Example: *Three out of every four trucks on the road are followed by a car, while only one out of every five cars is followed by a truck. What fraction of vehicles on the road are trucks?*

The transition matrix takes the form:

$$\begin{array}{cc} & \begin{array}{cc} \mathrm{C} & \mathrm{T} \end{array} \\ \begin{array}{c} \mathrm{C} \\ \mathrm{T} \end{array} & \begin{pmatrix} 4/5 & 1/5 \\ 3/4 & 1/4 \end{pmatrix} \end{array}$$

- We construct an R code to estimate the proportion of trucks on the road using Markov Chain Monte Carlo.
- As an initial guess, we say that 50% of vehicles are trucks.

```
Nsteps <- 10

Mat <- matrix(c(0.8,0.75,0.2,0.25),2,2)
probs <- matrix(0,2,Nsteps)
probs[,1]=c(0.5,0.5)      # Initial guess of probability

for (k in 2:Nsteps) probs[,k] = t(Mat) %*% probs[,k-1]
probs
```

A matrix: 2 × 10 of type dbl

0.5	0.775	0.78875	0.7894375	0.7894719	0.7894736	0.7894737	0.7894737	0.7894737	0.7894737
0.5	0.225	0.21125	0.2105625	0.2105281	0.2105264	0.2105263	0.2105263	0.2105263	0.2105263

- As can be seen, the estimate quickly reaches the values $(0.78947, 0.21053)$, which we determine is the solution.
- Checking mathematically, we know the invariant point is $(15/19, 4/19)$, which confirms our results.
- Hence, the Markov chain (for loop with transitions) has accurately estimated the proportion of trucks on the road.

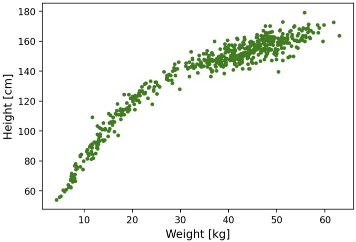
Important terms:

- **Prior:** Our initial guess X_1 of the parameter values (in example, $(0.5, 0.5)$).
- **Posterior:** Subsequent parameter values following one or more transition steps $(X_2, ..., X_N)$.
- In our example, the first-step posterior is $(0.775, 0.225)$, and the final posterior estimate is $(0.78947, 0.21053)$.
- Hence, Markov Chain Monte Carlo can be used to estimate parameters and solve equations which cannot be solved using traditional methods!

Data Fitting Using MCMC

- The same processes that allow MCMC to find particular values can also be used to fit to data.
- This works by running the Markov Chain and finding the parameter values with best data fit.
- In this case, we specify the type of function we want to fit, and set some **priors** for the initial guess of the parameter values.

We want to fit a regression model to the height vs. weight data below:



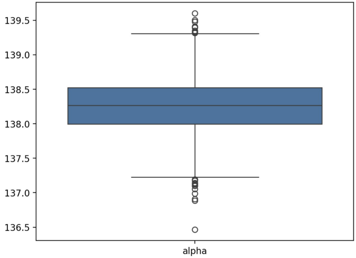
The data was taken from `https://raw.githubusercontent.com/newby-jay/MATH509-Winter2024-JupyterNotebooks/main/Data/Howell1.csv`.

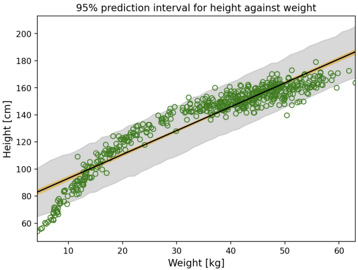
- To fit the data, we first choose a linear regression model for height vs. weight.
- We take the intercept α to be normally distributed, and take the slope β to be log-normally distributed so it is positive.
- Since the mean height is approximately 138 cm, we set a **prior** that α is normally distributed with mean 138 and standard deviation 20.

In particular, the prior is:

$$
\begin{aligned}
h_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \alpha + e^{log(\beta)}(x_i - \bar{x}) \\
\alpha &\sim \text{Normal}(138, 20) \\
\log(\beta) &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Uniform}(0, 50)
\end{aligned}
$$

- Running MCMC using PyMC (Python), the estimated parameter value of α is given in the boxplot below (left).
- The mean regression curve, 95% confidence interval plot of the mean, and 95% confidence interval of the data is given below (right).

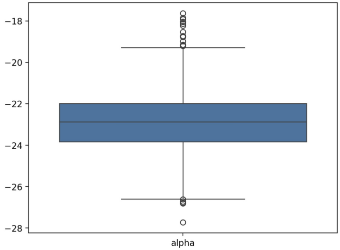


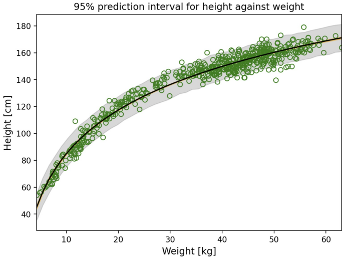


- The linear fit is not very good, as the data is nonlinear.
- To get a better fit, we use a regression model between height and the log-transformed weight.
- The **prior** is:

$$
\begin{aligned}
h_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \alpha + e^{\log \beta} \log(w_i) \\
\alpha &\sim \text{Normal}(178, 20) \\
\log(\beta) &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Uniform}(0, 50)
\end{aligned}
$$

- Running MCMC using PyMC (Python), the estimated parameter value of α is given in the boxplot below (left).
- The mean regression curve, 95% confidence interval plot of the mean, and 95% confidence interval of the data is given below (right).
- The fit is much better!





Thank You