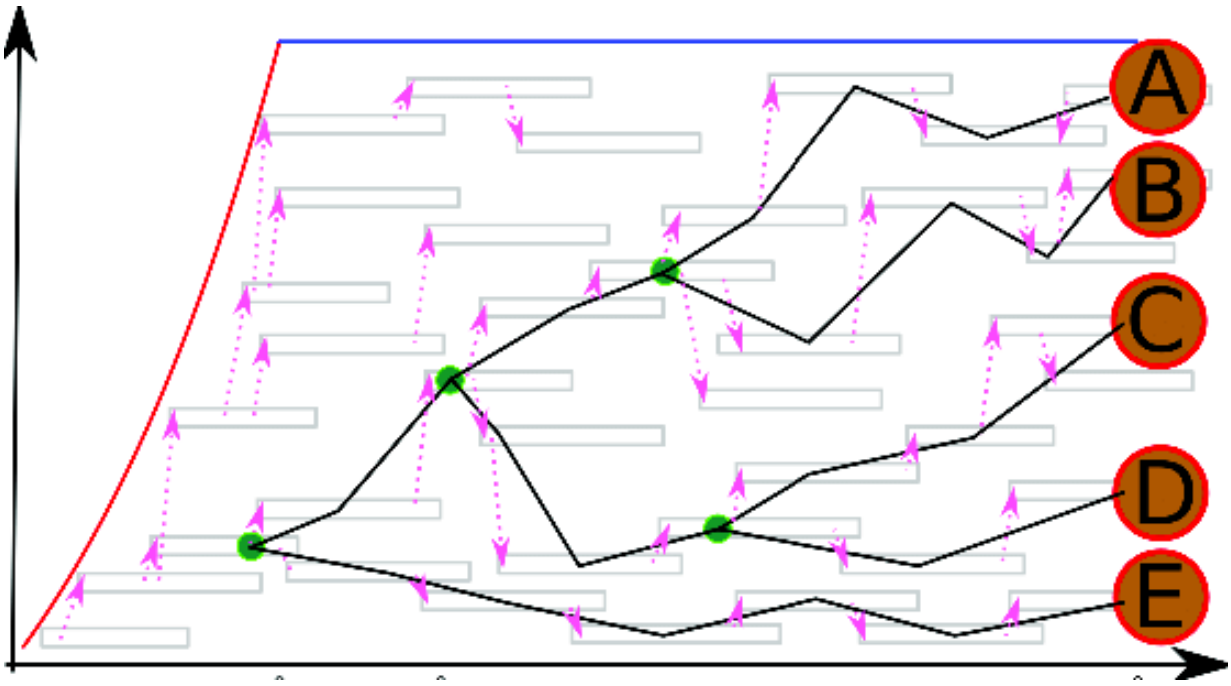


Mathematics – Probabilities & Statistics

Department of Mathematics

Stochastic Simulations

NOTEBOOK

Author
Youthy WANG

May 12, 2023

Contents

Chapter 1

Introduction to Stochastic Simulations

Simulation serves as one of the vital parts of a **stochastic model** nowadays, particularly when it is impossible to find a closed-form analytical solution. With the help of simulation, one can do some judicious guesses on properties of a stochastic model.

Simulation does well in

(i) describing and analyzing the system behaviors,

(ii) answering what-if questions about the system,

(iii) aiding in system design and optimization.

In the very first part of the notes, a brief introduction to stochastic simulations and review of probability and statistical tools are included.

1.1 History of Simulations

- 1777: Buffon's needle experiment: estimate π.
- 1940s: Ulam and Metropolis (**Monte Carlo Method**) to develop H-bomb (hydrogen bomb).
- 1960s: more applications, like manufacturing and supply chain, queuing models for service systems and in natural science (nuclear physics, fluid dynamics, biology, etc).

1.2 Concept of Stochastic Simulations

1.2.1 From Stochastic Processes to Stochastic Simulations

One example often used in simulation is $G/G/1$ queues. Since waiting time and service time distributions are too general (and can be complicated) compared with $M/M/1$ queues, it is hard to do general analytical analyses.

Example 1. Simulation of a $G/G/1$ Queue

Simulation algorithm components:

- **Input**: two i.i.d. sequences $\{U_n\}$ and $\{V_n\}$ corresponding to the inter-arrival (U_n: inter-arrival between the n^{th} and $(n+1)^{\text{th}}$ person) and service time, respectively.
- **Initialization**: since the first customer does not have to wait, $W_1 = 0$
- **System dynamics**: transform the input inter-arrival and service times to customers' waiting times by *(Lindley's recursion)*:

$$W_{n+1} = (W_n - U_n + V_n)^+.$$

- **Output**: Suppose we observe $W_1, \cdots, W_N$ from the simulation, a natural estimate for the steady-state waiting time is the sample mean waiting time. It is also possible to compute a confidence interval.

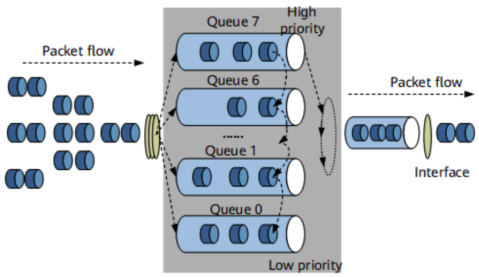
Another example is from the **discrete-event simulation** models. The application background is, in telecommunication business, the service provider usually need to make a commitment in the form of

$$P\{\text{response time} \leq 400\text{ms}\} \geq 0.9995.$$

It is called the **service level agreement (SLA)**. If the condition is not satisfied, the service provider needs to pay back the service fee to the customer. An accurate estimate of the SLA is economically important for the service provider.

Example 2. Simulation of a Telecommunication System

A telecommunication system can be modeled as a complex queuing network. There are 8 classes of customers and a single server. Customers of different class may have different inter-arrival and service time distributions, which are unknown.



Customers are served according to a *non-preemptive static priority rule*:

- Whenever the server becomes idle, it will pick the head-of-line customer from the non-empty queue of highest priority.
- After a customer enters server, its service will not be interrupted until service completion.

Let $W_\infty^{(i)}$ be the steady-state response time of class i customers. Our goal is to estimate the probability

$$P\{W_\infty^{(i)} > 400\text{ms}\}.$$

- Input: inter-arrival time and service time;
- System Dynamics: priority / scheduling;
- Output: $W_\infty^{(i)}$;
- Further Development: biased or unbiased / efficiency.

Stochastic simulation is a method for analyzing the performance of systems whose behavior depends on the interaction of random processes, which can be *fully characterized by probability models*.

We will mainly look at **discrete-event simulation models** with following technical aspects:

1. Identify and generate the input
2. Collect and analyze the output
3. Construct and implement they dynamics, including the events, calendar, etc. using computer programming
4. Consider potential system design improvements via simulation results

Chapter 2

Generate Randomness

2.1 Independent (I.I.D.) Random Variables Generation

For people or computer programs, in order to generate random variables, they follow the very same basic simulation procedures:

(1) Source of Randomness: usually assumed i.i.d. $U[0,1]$.

(2) Input random variables with known distributions.

(3) Output random variables which have the properties we want (with **system mechanics**, e.g., Lindley's recursion).

2.1.1 Generator for Random Numbers

In the first step, we need to generate i.i.d. $U[0,1]$ random numbers. There are several ways to "imitate" the randomness.

- Random Devices: 10-sided dice to generate each digit of your random uniform number.
- Random Number Table: used in history, but is too small and not really random.
- **Pseudo-random Number Generator (RNG)**: a basic and essential ingredient for any stochastic simulation algorithms, deterministically generated sequence of values, but appear to be i.i.d. $U(0,1)$.

To design RNG, **multiple recursive generators (MRG)** are designed:

$$x_i \equiv \sum_{j=1}^{k} a_j x_{i-j} (\text{mod } m) \text{ and } U_i = \frac{x_i}{m}.$$

When $k = 1$, it is called **Linear Congruential Generator (LCG)**.

Usually, m is a large prime number (to avoid short periods), $a_1, a_2, \cdots, a_k$ are integers such that the recursion has cycle length $m^k - 1$.

In general, parameter $\boldsymbol{a}$ and m should be chosen to achieve:

- For any initial seeds(the initial values $x_1, x_2, \cdots, x_k$ determines the whole sequence of random numbers $\rightarrow$ repeatable/comparable results), the resultant sequence $\{U_i\}$ has the appearance of i.i.d. $U[0,1]$.
- Long cycle for any initial seeds.
- Easy to generate on a computer.

2.1.2 Generate Random Variables from Specified Distributions (Special)

Inverse Method

Inverse (transform) method works to generate a random variable with CDF in hand, using the results from RNG.

- For *continuous RVs*, the **inverse transform method** is valid due to **inverse theorem** (details in STA 2001 notes).

Method 1. Inverse Method/Algorithm (Continuous Type)

[Random number generator from arbitrary distribution]
(i) generator a random number y from U(0, 1)
(ii) Take x = F^{-1}(y), then x is a random number generated from the distribution or RV with cdf F(x).
(iii) When it is not one-to-one mapping, no general solutions.

Explicitly, inverse method needs explicit inverse functions, e.g.,

Some densities with distribution functions that are explicitly invertible

Name	Density	Distribution function	Random variate
Exponential	$e^{-x}, x > 0$	$1 - e^{-x}$	$\log(1/U)$
Weibull(a), $a > 0$	$ax^{a-1}e^{-x^a}, x > 0$	$1 - e^{-x^a}$	$(\log(1/U))^{1/a}$
Gumbel	$e^{-x}e^{-e^{-x}}$	$e^{-e^{-x}}$	$-\log\log(1/U)$
Logistic	$1/(2 + e^x + e^{-x})$	$1/(1 + e^{-x})$	$-\log((1-U)/U)$
Cauchy	$1/(\pi(1 + x^2))$	$1/2 + (1/\pi)\arctan x$	$\tan(\pi U)$
Pareto(a), $a > 0$	$a/x^{a+1}, x > 1$	$1 - 1/x^a$	$1/U^{1/a}$

Figure 2.1: Easily Invertible RVs

Sometimes, F^{-1} is not available in explicit form. Standard packages will use some approximations. For example, to inverse $\Phi(x)$ as the CDF of $Z \sim N(0,1)$, here is a rational polynomial estimation.

$$\phi^{-1}(u) \approx y + \frac{p_0 + p_1 y^1 + p_2 y^2 + p_3 y^3 + p_4 y^4}{q_0 + q_1 y^1 + q_2 y^2 + q_3 y^3 + q_4 y^4}, 0.5 < u < 1, \text{ where } y = \sqrt{-2\ln(1-u)}.$$

k	p_k	q_k
0	-0.322232431088	0.099348462606
1	-1	0.588581570495
2	-0.342242088547	0.531103462366
3	-0.0204231210245	0.10353775285
4	-0.0000453642210148	0.0038560700634

- For *discrete RVs*, since its CDF is NOT injective, we cannot directly use $F^{-1}(U)$. The inverse method here becomes: (Let $P\{X = x_j\} = p_j, j = 0, 1, \cdots, \sum_j p_j = 1$.)

Method 2. Inverse Method/Algorithm (Discrete Type)

(i) Generate random variable U from $U[0, 1]$.

(ii) Construct

$$X = \begin{cases} x_0, & \text{if } U < p_0 \\ x_1, & \text{if } p_0 \le U < p_0 + p_1 \\ \vdots \end{cases}.$$

(iii) It has the desired discrete distribution.

Composite Method

Sometimes relations between 2 different RVs are used along with the inverse method for the generation.

Example 3

Poisson random variables can be derived using the relationship with exponential. (Recall inter-arrival time in Poisson processes.)

(*Concrete Steps*). Since $P[Pois(\lambda) \le n] = P(n$ arrivals occur after the unit time). Generate several i.i.d. $Y_1, \cdots, Y_m \sim exp(\lambda)$, then N satisfying $\sum_{i=1}^{N} Y_i \le 1 < \sum_{i=1}^{N+1} Y_i$ follows $Pois(\lambda)$.

2.1.3 Generate Random Variables from Specified Distributions (General)

In the real industry, the majority of CDFs *cannot be inverted efficiently*. Therefore, we use some **acceptance-rejection methods** for doing simulations.

One simple idea is to imitate the estimation of π (Monte Carlo):

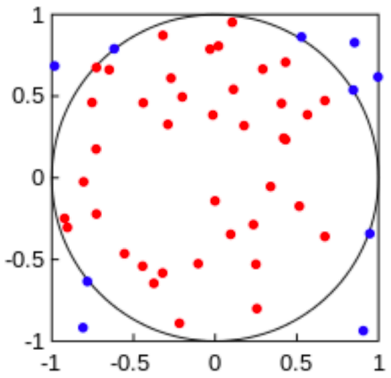


Figure 2.2: Estimation of π

Suppose we have in hand an oracle to sample from a distribution with PDF $g(x)$. Using it to generate samples from another distribution with PDF $f(x)$, is equivalent to collecting uniform points in the area below $f(x)$, given $f \leq g$.

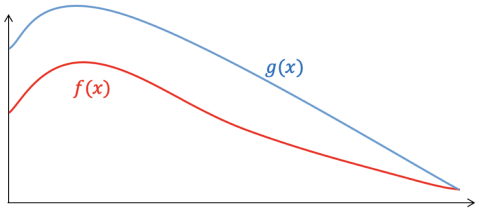


Figure 2.3: Illustration of the Idea behind A-R

Since technically, $f(x) \leq g(x)$ cannot hold for all x, a constant $M > 1$ is chosen such that $f \leq Mg$. $f(x)$ is called the **target distribution**, $g(x)$ is called the **proposed distribution**.

Below lists the most straightforward acceptance-rejection algorithm, (details in STA 2002 notes).

Method 3. Acceptance-Rejection Method

(*Algorithm.* **Accept-Reject** Algorithm) Let $f(x)$ be a pdf. Suppose that Y is a random variable with pdf $g(y)$, $U \sim U[0,1]$, Y and U are independent and $f(x) \leq Mg(x)$, $x \in \mathbb{R}$. The following algorithm generates a random variable X with pdf $f(x)$.
(i) Generate Y and U;
(ii) If $U \leq \dfrac{f(Y)}{Mg(Y)}$ then take $X = Y$. Else return to step (i);
(iii) X has pdf $f(x)$.

Here comes an examples in sampling posterior distribution.

Example 4. Sampling with A-R Algorithm

Consider a Bayesian statistic model in which the data are i.i.d. Poisson random variables with mean Θ. Suppose the prior of the parameter $\Theta > 0$ is a $\Gamma(\alpha, \beta)$ with $\alpha, \beta > 0$. Design an A-R algorithm to sample from the posterior distribution $f_{\Theta|X_1,\cdots,X_n}(\theta)$.

(*Idea.*) Use $exp(\lambda)$ as the proposed distribution, where

$$\lambda = \arg\min_{\mu>0} \sup_{\theta>0} \frac{f_{\Theta|X_1,\cdots,X_n}(\theta)}{\mu e^{-\mu\theta}}.$$

Discussions of the A-R Algorithm

- **Efficiency**: Since the probability for a sample $Y \sim g(y)$ being accepted, called acceptance rate, is (Geometric distribution)

$$P\left(U \leq \frac{f(Y)}{Mg(Y)}\right) = \frac{1}{M}.$$

  To avoid inefficiency, people do NOT want too many rejections. Thus, it is reasonable to minimise M. Practically,

$$M = M^\star = \sup_{x\in\mathcal{X}} \frac{f(x)}{g(x)}.$$

- **Choice of g**: g must have a "heavier tail" than f.
- Powerful and widely applicable to a variety of simulation tasks, (e.g. MCMC). Also fine to use in X with p.d.f. $\frac{f(x)}{D}$ where D is hard to compute.

2.1.4 Coding Part: Python Codes for the Generation of RVs

To do simulations in python, it is important to study how to use NumPy.

```
import numpy as np

# introduction to np.random
# "np.random" is used to generate random numbers following different
    rules
np.random.seed(2013)  # seed for reproducibility
np.random.randint(low, high=None, size=None, dtype=int) # Return random
    integers from `low` (inclusive) to `high` (exclusive).
np.random.rand(d0, d1, ..., dn)  # pseudo-RNG, random values ([0,1)) in
    a given dimension.
np.random.exponential(scale=1.0, size=None) # random values following
    exponential with mean = scale
np.random.normal(loc=0.0, scale=1.0, size=None) # random values
    following Guassian N(loc,scale^2)

# introduction to basic computation in NumPy
np.arange([start,] stop[, step,], dtype=None, *, like=None) # Return
    evenly spaced values within a given interval (1-dim).
np.array(object, dtype=None, *, copy=True, order='K', subok=False,
    ndmin=0, like=None) # multi-dimensional array.
```

```
np.concatenate((a1, a2, ...), axis=0, out=None, dtype=None,
    casting="same_kind") # concatenate 2 arrays
np.vstack(tup) # Stack arrays in sequence vertically (row wise).
np.hstack(tup) # Stack arrays in sequence horizontally (column wise).
np.dstack(tup) # Stack arrays in sequence depth wise (along third
    dimension).
np.where(condition, [x, y]) # Return elements chosen from `x` or `y`
    depending on `condition`.
np.dot(a, b, out=None) # inner product of vectors in 1-d, but matrix
    multiplication for 2-d matrices
```

Listing 2.1: Basic Knowledge of NumPy

We need to plot some figures sometimes for a much more direct view. “matplotlib” works for the aim.

```
import matplotlib.pyplot as plt

# plot some discrete points / Scatter plot
plt.plot([1,2,3,4],[1,4,9,16],'bo') # show 4 points by using vectors /
    numpy arrays
plt.scatter(X_1, X_2, c='blue', label='X') # another way
plt.ylabel('somenumbers')
plt.xlabel("1'axex")
plt.axis([0,6,0,20]) #set axis x 0-6 and axis y 0-20
plt.show()

# Line plot
plt.plot([1,2,3,4],[1,4,9,16]) # matplotlib offers a variety of options
    for color, linestyle, and marker.
# an example for line plotting
radius=np.array([1.0,2.0,3.0,4.0,5.0,6.0])
square=radius**2
area=np.pi*square
plt.plot(radius,area,label='Circle')
plt.plot(radius,square,marker='o',linestyle='--',color='r',label='Square')
plt.xlabel('Radius/Side')
plt.ylabel('Area')
plt.title('Area of Shapes')
plt.legend() # show lable of each line
plt.show()

# Histogram plot
gaussian_numbers = np.random.normal(size=1000)
plt.hist(gaussian_numbers,bins=20, density=True) # set size of bin to 20
plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()

# Working with multiple figures and axes
plt.figure(1)
plt.subplot(number of rows, number of columns, indexes)
plt.plot(...)
# another way to use
fig, axs = plt.subplots(2, 2, figsize=(9,9))
axs[0, 0].plot(...)
```

Listing 2.2: Basic Knowledge of "matplotli.pyplot"

2.2 Correlated Random Variables Generation

2.2.1 Generate Correlated Multi-variate Normal

One special instance is to generate **correlated multi-variate normal** distributions.

Suppose mean vector $\boldsymbol{\mu}$ and covariance matrix Σ are in hand. Since we can generate i.i.d. standard normal $\{Z_i\}_{i\geq 1}$, thus $\boldsymbol{Z} = [Z_1, \cdots, Z_n]^T$ as a joint normal with mean $\mathbf{0}$ and covariance matrix I.

If $\Sigma \succ 0$, do **Cholesky decomposition** of Σ, we get $\Sigma = LL^T$, $L\boldsymbol{Z} + \boldsymbol{\mu}$ follows the joint distribution we aim to get. (For semi-definite matrix, one may need other algorithms to find $\Sigma = LL^T$, which may not exist.)

2.2.2 Generate General Correlated Random Variables

In most of the real cases, joint distribution is NOT in hand (for otherwise, A-R algorithm can be applied). People need to develop other tools by using marginal distributions and covariances to generate joint distribution.

Suppose we would like to generate multivariate RVs $(X_1, \cdots, X_n)$ with specified general marginal distributions F_i for every $i = 1, \cdots, n$, but with correlation between X_i and X_j for any (i, j) pair.

IDEA: Use **correlated joint uniform distribution** and apply inverse method (1). Here we introduce a new tool called **copula**.

Definition: Copula

A copula $Cp(\boldsymbol{x})$ is a a multi-dimensional cumulative distribution function (CDF) that results in *all marginal distributions being uniformly distributed* on $(0, 1)$.

Properties of Copula:

Suppose we have two random variables X and Y with joint distribution function $H(x, y)$ and marginal distribution $F(x)$ and $G(y)$.

(1) The joint distribution function of $F(X)$ and $G(Y)$ is a copula. $\widetilde{Cp}(x, y) = P(1 - F(X), 1 - G(Y))$ is also a copula. It is sometimes called the *copula generated by the tail distributions of X and Y*.

(2) $Cp(x, 1) = x, Cp(1, y) = y, Cp(0, 0) = 0$.

(3) Since F and G are both increasing, $H(x, y) = P(X \leq x, Y \leq y) = P(F(X) \leq F(x), G(Y) \leq G(y)) = Cp(F(x), G(y))$ (comonotonicity).

(4) An appropriate copula Cp can model the presumed dependencies between $F(X)$ and $G(Y)$ – similar to the dependency "we think holds" between X and Y.

(5) If Cp is **absolutely continuous** on $(0,1)^2$, the corresponding density is

$$c(x,y) \overset{\text{defn}}{=\!=\!=} \frac{\partial^2}{\partial x \partial y} Cp(x,y).$$

There are many kinds of copulas, like $C(\Phi(Z_1), \Phi(Z_2))$ coming from standard normal, which is named **Guassian copula**. A tail distribution generated copula that indicates a positive correlation between X and Y and which gives a positive probability that $X = Y$ is the **Marshall-Olkin copula**. The latter comes from $Cp(x,y) = P(1 - F(X), 1 - G(Y))$, where $X = \min\{T_1, T_3\}$ and $Y = \min\{T_2, T_3\}$, where T_1, T_2 and T_3 are i.i.d. exponential.

Method 4. Copula Method

(I) Generate with **Induced Copula**: (i.e., joint distribution H and marginal distributions G_i are in hand.)

(i) Generate $(W_1, \cdots, W_n) \sim H$;

(ii) Compute $U_i = G_i(W_i), \forall\ i$;

(iii) Return $X_i = F^{-1}(U_i), \forall\ i$.

(II) Generate with **Guassian Copula**: (i.e., only with dependencies inside $\boldsymbol{X}$)

(i) Generate the multivariate normal $(W_1, \cdots, W_n) \sim N(\mathbf{0}, \Sigma)$, where $\Sigma_{ij} = \text{Cov}(X_i, X_j)$;

(ii) Compute $U_i = \Phi(W_i), \forall\ i$;

(iii) Return $X_i = F^{-1}(U_i), \forall\ i$.

(III) Generate with the **Marshall-Olkin Copula**: (i.e., marginal distributions F, G are in hand.)

(i) Generate T_1, T_2, T_3, independent exponential random variables with rates $\lambda_1, \lambda_2, \lambda_3$.

(ii) Let $X = \min(T_1, T_3)$, $Y = \min(T_2, T_3)$.

(iii) Set $V = F^{-1}(e^{-(\lambda_1+\lambda_3)X})$, $W = G^{-1}(e^{-(\lambda_2+\lambda_3)X})$.

Remark.

- There is no guarantee that the generated RVs follow the same joint distribution as the proposed one with Guassian copula.
- We use $\text{Cov}(W_i, W_j) = \text{Cov}(X_i, X_j)$, but we can build in other ways to obtain a *similar* correlation relationship.
- It takes a much more complicated way to capture the *same* correlation as $\boldsymbol{X}$, i.e., $\Sigma_{ij} = C(\text{Cov}(X_i, X_j))$, where C is a function (see literature about *"normal to anything"(NORTA)*).
- Copula method is kind of a *heuristic* method. In real world (like finance), it has some limitations.

Discussions of Copula Methods

In more general terms, one can describe copulas as a general tool to model dependence of whatever kind and to separate the dependence structure from the marginal distributions.

Copulas are nonstructural models in the sense that there is a lack of classes that on the one hand are characterized by a finite number of parameters so that statistical estimation is feasible, and on the other are flexible enough to describe a broad range of dependence structure.

2.3 Random Process Generation

Another situation where we need to generate many random variables is with simple **random processes**.

2.3.1 Generating Discrete-Time Markov Chains

The basic idea to generate DTMC, is to decide the next state from the information of the transition matrix and current state, based on the Markov property. The probability in transition matrix can be modeled using RNG.

Algorithm 2.3.1: Generating DTMC

(0) Generate $X_1 \sim \pi$ (initial distribution) and set a list $S := [X_1]$.

(1) For $n = 1, 2, \cdots$, suppose $i = X_n$, generate X_{n+1} according to the categorical distribution P_{ij}.

(2) Append X_{n+1} to S.

2.3.2 Generating Poisson Processes

Since inter-arrival time $\{T_i\}_{i\geq 1}$ can describe a Poisson process, and follow exponential distributions (details in STA 4001 notes), the idea to simulate Poisson process is to determine the inter-arrival time.

Algorithm 2.3.2(i): Generating Homogeneous P.P.

(0) Let $T_0 := 0$.

(1) For $n = 1, 2, \cdots$, generate $U_n \sim U[0, 1]$, $T_n = -\frac{1}{\lambda}\ln(U_n)$ (inverse method).

(2) Arrival time $\{S_i\}_{i\geq 1}$ comes from $S_0 = T_0$, and $S_n = S_{n-1} + T_n, n \geq 1$.

For inhomogeneous Poisson processes, a **thinning algorithm**(a kind of A-R) comes to rescure.

Lemma: Generate One Nonhomogeneous P.P. from Another

Consider a one-dimensional nonhomogeneous Poisson process $\{N^\star_{(t)} : t \geq 0\}$ with intensity function $\lambda^\star(t)$. Let $T^\star_1, \cdots, T^\star_{N^\star_{(t_0)}}$ be the points of the process in the interval $(0, t_0]$. Suppose further that for $0 < t \leq t_0$, $\lambda^\star(t)$ dominates $\lambda(t)$.

For $i = 1, \cdots, n$, delete the point T_i with probability $1 - \dfrac{\lambda(T^\star_i)}{\lambda^\star(T^\star_i)}$; then the remaining points, denoted as T_j, form a nonhomogeneous Poisson process $\{N_{(t)} : t \geq 0\}$ with rate function $\lambda(t)$ in the interval $(0, t_0]$.

Proof. Since $\{N^\star_{(t)} : t \geq 0\}$ is a non-homgeneous Poisson process and points are deleted independently, it is clear that the number of points in $\{N_{(t)} : t \geq 0\}$ in any set of non-overlapping intervals are mutally independent random variables.

Consider conditional distribution of $N_{(t)}$ given $N^\star_{(t)}$,

$$P\left[N_{(t)} = m \middle| N^\star_{(t)} = n\right] = \int_{T_1 < \cdots < T_n} \sum_{1 \leq i_1 < \cdots < i_m \leq n} \prod_{j=1}^{m} \frac{\lambda(T^\star_{i_j})}{\lambda^\star(T^\star_{i_j})} \prod_{j=1}^{n-m} \left(1 - \frac{\lambda(T^\star_{k_j})}{\lambda^\star(T^\star_{k_j})}\right).$$

Since $T^\star_i$ can be seen as order statistics of i.i.d. distribution T with CDF $F(s) = \dfrac{\int_0^s \lambda^\star}{\int_0^t \lambda^\star}$, $s \in [0, t]$ (given $N^\star_{(t)} = n$) and due to symmetry,

$$\text{RHS} = \binom{n}{m} \left(\int_T \frac{\lambda(T)}{\lambda^\star(T)}\right)^m \left(\int_T 1 - \frac{\lambda(T)}{\lambda^\star(T)}\right)^{n-m} \sim Bino\left(n, \frac{\int_0^t \lambda}{\int_0^t \lambda^\star}\right).$$

We can conclude that by decomposition rule, $\{N_{(t)} : t \geq 0\}$ is a non-homgeneous P.P. ■

Algorithm 2.3.2(ii): Generating Nonhomogeneous P.P.

(0) Let $t_0 = T_0 := 0$. Assume that $\lambda_m \equiv \max_t \lambda(t) < \infty$.

(1) For $n = 1, 2, \cdots$, generate *potential arrivals* at the rate λ_m (i.e., $t_n = -\dfrac{1}{\lambda_m} \ln(U_n)$), and accept (keeps) it with probability $\dfrac{\lambda(s_n)}{\lambda_m}$. Note that different t_n corresponds different s_n (A-R algorithm).

(2) Let $T_n = t_{i_n}$, $\forall\ n \geq 1$, where t_{i_j} is the j^{th} accepted time. Arrival time $\{S_i\}_{i \geq 1}$ comes from $S_0 = T_0$, and $S_n = S_{n-1} + T_n, n \geq 1$.

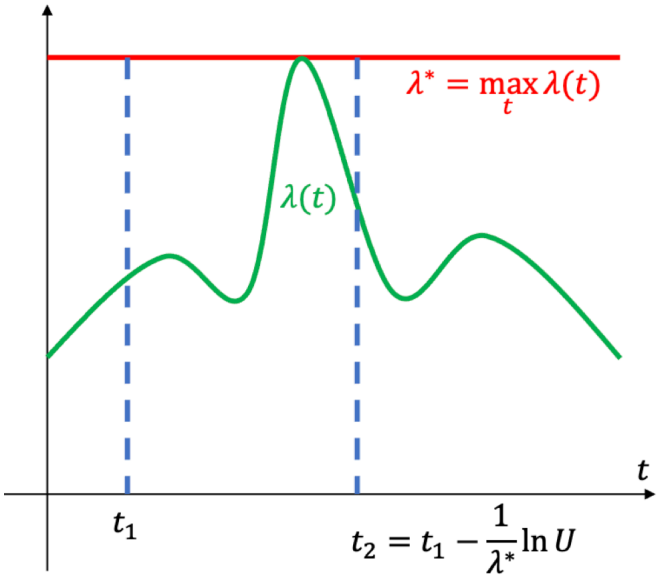


Figure 2.4: Illustration on Algorithm 2.3.2(ii)

2.3.3 Generating Markov Jump Processes

Due to the first definition of MJP (details in STA 4001 notes), we can simulate MJP by state jumpping and time holding.

Algorithm 2.3.3: Generating MJP

(0) Generate $X_1 \sim \pi$ (initial distribution); set $t_1 := 0$ and a list $S := [(X_1, t_1)]$

(1) For $n = 1, 2, \cdots$, suppose $i = X_n$, generate $\Delta \sim \exp(\lambda(i))$ and let $t_{n+1} := t_n + \Delta$.

(2) Generate X_{n+1} according to the jump matrix Q_{ij} of the embedded DTMC.

(3) Append (X_{n+1}, t_{n+1}) to S.

2.3.4 Generating Brownian Motions

We can use the fact that the increments are independently normally distributed random variables.

- If we only wish to simulate $B_{(t)}$ at one fixed value $t > 0$, then we need only generate a standard normal $Z \sim N(0, 1)$ and set $B_{(t)} = \sqrt{t}Z$.
- Typically we would like to simulate k values at time $t_0 = 0 < t_1 < \cdots < t_k$. These can be done by generating k i.i.d. standard normal random variables $Z_1, \cdots, Z_k$, and then let $B_{(t_j)} = \sum_{i=1}^{j} \sqrt{t_i - t_{i-1}} Z_i$ for $j = 1, \cdots, k$.

Nonetheless, Brownian motion evolves continuously in time. If we do not want to be limited in a fixed set of time points, we can use discrete processes to apporoximate them. Note that the discretization can be used for other continuous processes, such as stochastic differential equations (SDE), like pricing financial derivatives and SEIR model for pandemic spread.

- Set a small step size h, let $t_i = h \cdot i$, $i = 1, \cdots, n$.
- Here is a trade-off: large h results in less accurate approximation and small h causes too many transitions, thus less efficient algorithm.

2.4 Simulations of Discrete Events

Discrete-event simulations are based on events, where one event is generated at a time (discrete event, continuous time). People record some necessary statistics composed of the general simulation model.

Key Components in Discrete-event Simulations:

- **System State**: the collection of variables necessary to describe the system at a particular time.
- **Simulation Clock**: a variable giving the current value of simulated time.
- **Event List**: a list containing the next time when every type of event will occur.
- **Timing Routine**: a subprogram that determines the next event from the event list and then advances the simulation clock to the time when that event is to occur.
- **Event Routine**: a subprogram that updates the system state when a particular type of event occurs
- **Statistical Counters**: variables used for storing statistical information about system performance

The simulation clock represents the simulated time. It always moves forward. Times of all known future events are determined and placed in the future events list (FEL), ordered by time.

Here is a simple example (a simulation result from the example of a $G/G/1$ queue in Chapter 1).

Clock	Number of Customers	Event Calendar		Queue
		Next Arrival	Next Service	
0	0	$0 + 3.72 = 3.72$	∞	{}
3.72	1	$3.72 + 3.46 = 7.18$	$3.72 + 2.81 = 6.53$	{}
6.53	0	7.18	∞	{}
7.18	1	$7.18 + 0.83 = 8.01$	$7.18 + 5.80 = 12.98$	{}
8.01	2	∞	12.98	{3}
12.98	1	∞	$12.98 + 1.67 = 14.65$	{}
14.65	0	∞	∞	{}

Figure 2.5: Illustration on Discrete-event Simulation

Another common class of instances is about the simulation of **priority queues**, it is a little bit different due to its special properties.

Example 5. Two-class Priority Queue

Consider a single node in a communication network. The node transmit data at a constant rate of $62,500$ byte/ms. There are two kinds of data packages of size 100 bytes and 1500 bytes. Suppose packages arrival according to some Poisson processes. The arrival rates of the 100-byte and 1500-byte packages are 12.5 and 25 per ms, respectively.

The node transmits these packages with a non-preemptive static priority rule. It always serves the 100-byte packages first. But if a 1500-byte package is in transmission, it will not be interrupted by a newly arriving 100-byte package.

- System state: the number of type 1/2 packages in the system.
- Timing routine: determines next event type and advance clock to the next event's time.
- Event routine: Update the system state (type 1/2 arrival; type 1/2 departure; type 1/2 in the system).
- Statistical counters: the delay of each package (departure time - arrival time - in-service period)

When generating the list of events, one needs efficient list processing (e.g., linked lists). Moreover, there should be some rules to break the ties if two events show up simultaneously.

2.4.1 Coding Part: Python Codes for the Simulation of Priority Queues

Here lists an example of simulating a priority queue with 2 priorities of P.P. arrivals.

```
import numpy as np
import queue
from numpy.random import exponential as exp

class Customer1:
    def __init__(self, arrival_time_1):
        self.arrival_time_1 = arrival_time_1

class Customer2:
    def __init__(self, arrival_time_2):
        self.arrival_time_2 = arrival_time_2

class Simulator:
    def __init__(self, lambda1, lambda2, service_time_1, service_time_2):
        self.lamb1, self.lamb2, self.t1, self.t2 = lambda1, lambda2,
            service_time_1, service_time_2
        self.n1, self.n2, self.t = 0, 0, 0
        self.event_list = queue.PriorityQueue()
        self.queue1 = queue.Queue()
        self.queue2 = queue.Queue()
        self.customers_1 = []
        self.customers_2 = []
```

```
def run(self, max_time):
    self.event_list.put((self.t + exp(1/self.lamb1), 'A1'))
    self.event_list.put((self.t + exp(1/self.lamb2), 'A2'))
    while self.t < max_time:
        self.t, event = self.event_list.get()
        if event == 'A1':
            self.arrival_1()
        elif event == 'A2':
            self.arrival_2()
        elif event == 'D1':
            self.departure_1()
        elif event == 'D2':
            self.departure_2()

def arrival_1(self):
    self.event_list.put((self.t + exp(1/self.lamb1), 'A1'))
    c = Customer1(self.t)
    self.queue1.put(c)
    if self.n1 == 0 and self.n2 == 0:
        self.event_list.put((self.t + self.t1, 'D1'))
    self.n1 += 1

def arrival_2(self):
    self.event_list.put((self.t + exp(1/self.lamb2), 'A2'))
    c = Customer2(self.t)
    self.queue2.put(c)
    if self.n1 == 0 and self.n2 == 0:
        self.event_list.put((self.t + self.t2, 'D2'))
    self.n2 += 1

def departure_1(self):
    self.n1 -= 1
    c = self.queue1.get()
    c.departure_time_1 = self.t
    self.customers_1.append(c)
    if self.n1 > 0:
        self.event_list.put((self.t + self.t1, 'D1'))
    if self.n1 == 0 and self.n2 > 0:
        self.event_list.put((self.t + self.t2, 'D2'))

def departure_2(self):
    self.n2 -= 1
    c = self.queue2.get()
    c.departure_time_2 = self.t
    self.customers_2.append(c)
    if self.n1 > 0:
        self.event_list.put((self.t + self.t1, 'D1'))
    if self.n1 == 0 and self.n2 > 0:
        self.event_list.put((self.t + self.t2, 'D2'))

def delay(self):
    delay1 = np.mean([c.departure_time_1 - c.arrival_time_1 -
       self.t1 for c in self.customers_1])
    delay2 = np.mean([c.departure_time_2 - c.arrival_time_2 -
       self.t2 for c in self.customers_2])
    return delay1, delay2
```

```
if __name__ == '__main__':
    sim = Simulator(lambda1 = 0.1, lambda2 = 0.2, service_time_1 = 0.2,
        service_time_2 = 3)
    sim.run(max_time=1000000)
    delay1, delay2 = sim.delay()
    print(f'The average delay for priority level 1 customers is
        {delay1:.3f}')
    print(f'The average delay for priority level 2 customers is
        {delay2:.3f}')
```

Listing 2.3: Simulation of Simple Priority Queues

Chapter 3

Input & Output Analyses

3.1 Input Analysis

In real life, one always needs to use random variables and stochastic processes to do simulations on discrete events. General process of a discrete-event simulation also contains input & output analyses.

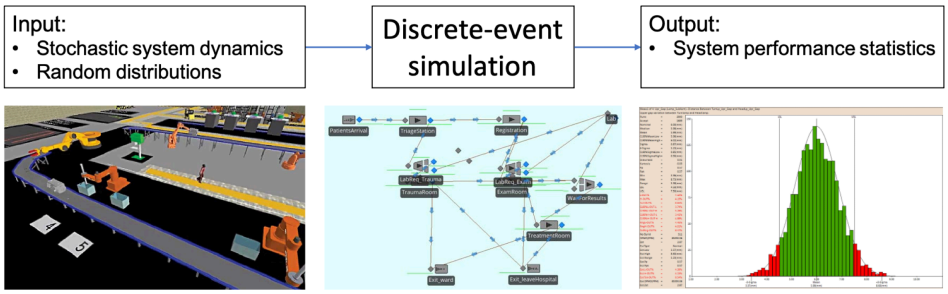


Figure 3.1: Process of Discrete Event Simulation

Motivation of input analysis:

Suppose we have a single-server system.

- We wonder whether the inter-arrival times exponential, Weibull, stationary and independent, etc.
- Other questions like what if one applies wrong model or correct model with wrong rate.

To get to know the simulation results under thoses different inputs, one need input analysis to help.

3.1.1 Techniques in Input Analysis

The aim to do input analysis it to avoid **Garbage-In-Garbage-Out (GIGO)**.

To build a reasonable model, one should firstly do some analyses, either from expert's opinion or data collection (learning from data). There are some basic steps to follow:

(1) **Visualization** from histogram: With enough observations, the histogram will eventually converge to the true distribution.

(2) Three ways to use data: use **sample distribution** (**bootstrapping**, potential drawbacks: not enough data) vs. use **continuous empirical distribution** (potential drawbacks: cannot generate beyond the bounds) vs. fit a **theoretical distribution**.

(3) Parameter estimation (always use MoM, MLE, etc.).

Below is a general guideline of distribution selection:

- *Discrete* or *Continuous*
- *Univariate* or *Multivariate*
- The Amount of Available Data for Guess
- Experts' Opinions or Previous Experience

As mentioned in the guide, there are some previous experience of fitting, like

⋆ **Uniform** (not much known from the data, except maybe min and max possible values)

⋆ **Triangular** (know min, max, most likely values)

⋆ **Exponential** (inter-arrival times from a Poisson process)

⋆ **Normal** (good model for heights, weights, IQs, sample means, etc.)

⋆ **Beta** (good for specifying bounded data)

⋆ **Gamma, Weibull, Gumbel, lognormal** (reliability data)

Example 6. MLE for Gamma Distributions

Let n samples $X_1, \cdots, X_n \sim \Gamma(\alpha, \beta)$ i.i.d, then likelihood function

$$L = \frac{\beta^{n\alpha}}{\Gamma^n(\alpha)} \prod_{i=1}^{n} X_i^{\alpha-1} e^{-\beta \sum_{i=1}^{n} X_i}.$$

To maximize the likelihood function, take derivatives,

$$\frac{\partial}{\partial \alpha} \ln(L) = \frac{n\alpha}{\beta} - \sum_{i=1}^{n} X_i,$$

$$\frac{\partial}{\partial \alpha} \ln(L) = \ln \frac{\beta^n}{\prod_{i=1}^{n} X_i} - n\frac{\Gamma'(\alpha)}{\Gamma(\alpha)}.$$

Thus, $\hat{\beta} = \dfrac{\hat{\alpha}}{\overline{X}}$, and $\hat{\alpha}$ satisfies $\ln(\hat{\alpha}) + \overline{\ln(X)} - \ln(\overline{X}) - \dfrac{d}{d\alpha} \ln(\Gamma(\hat{\alpha})) = 0$.

Since $\Gamma(\alpha)$ is complicated, we use techs in numerical differentiation

$$f'(x) \approx \frac{f(x+h) - f(x)}{h},$$

for small h.

Some results can be applied for saving people's work in finding statistics for estimating parameters.

Theorem 1. Invariance Property

If $\widehat{\theta}$ is the MLE of some parameter θ and $h(\cdot)$ is any reasonable function, then $h(\widehat{\theta})$ is the MLE of $h(\theta)$.

3.1.2 Check the Models in Input Analysis

Now we have guessed a reasonable distribution estimated the relevant parameters. The task here is to check how well the model is, i.e., the test hypothesis is,

$$H_0 : X_1, \cdots, X_n \sim \text{p.m.f/p.d.f } f(x).$$

Qualitative Ways: **Order statistics & QQ-plot**.

Quantitative Ways: **Goodness of Fit Tests**, like **Chi-square Goodness of Fit** and **Kolmogorov-Smirnov(K-S) Test**.

Real world is much more complaticed and we may encounter the following problems:

- No data: try to at least get minimum, maximum, quantiles and most likely distribution values out of experts or have an educated guess.
- Multivariate: try to estimate the covariance/marginal CDF from data and use copula method.
- Non-stationary: try to model the process as a nonhomogeneous Poisson process.
- No closed-form distribution: attempt to model as a mixture of reasonable distributions

3.2 Output Analysis

In the real industry, we may take interests in some quantities related to the Simulation. However, the simulation process only produces outputs which are random variables.

The output analysis aims to

- make inference on the quantity of interest given the output of a simulation process
- assess the accuracy/efficiency of a simulation process

3.2.1 Estimating the Mean

Suppose we want to estimate the value of expectation $z = \mathbb{E}[Z]$. We run a simulation algorithm n times and obtain **i.i.d.** samples $Z_1, \cdots, Z_n \sim Z$.

The Monte Carlo estimate for z is

$$z \approx \hat{z}_n = \frac{1}{n}\sum_{i=1}^{n} Z_i.$$

Since all samples are i.i.d., that is, one can compute a confidence interval for $\hat{z}_n$ with significance level α (s.d. $\sigma = \mathrm{Var}(Z)$ in hand),

$$\left(\hat{z}_n - Z_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}, \hat{z}_n + Z_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}\right).$$

In concrete applications, we usually require certain level of **accuracy** ε for the Monte Carlo estimation:

- **Absolute accuracy**: $|\mathrm{CI}| < 2\varepsilon$, where $|\mathrm{CI}|$ means length of *confidence interval*.
- **Relative accuracy**: $|\mathrm{CI}| < 2\varepsilon|z| \approx 2\varepsilon|\hat{z}_n|$.

Relative accuracy is used when $|z|$ is extremely large or small. Thus, the algorithm converges with rate $O\left(\frac{1}{\epsilon^2}\right)$ or $O\left(\frac{1}{\epsilon^2|z|^2}\right)$ in theory.

For the hard cases:

- *Dependent Samples*: still use CLT, but σ^2 might be different from $\mathrm{Var}(Z)$, e.g., ergodic Markov processes, time series models.
- *σ Not in Hand*: estimate σ^2 by **sample variance** (i.i.d.) or other methods like batch (dependent samples).

Example 7. CLT for AR(1)

Consider an **auto-regression model** with order 1, i.e. AR(1) model:

$$Z_n = aZ_{n-1} + \varepsilon_n, \text{ where } \varepsilon_k, \forall\ k \in \mathbb{N} \text{ are i.i.d. standard normal.}$$

Suppose $Z_1 \sim N\left(0, \frac{1}{1-a^2}\right)$. Consider the limiting distribution of $\sqrt{n}(\overline{Z}_n)_{(n\to\infty)}$.

Note here $\overline{Z}_n$ can be directly computed as

$$\frac{1}{n}\left(\frac{1-a^n}{1-a}Z_1 + \sum_{i=2}^{n}\frac{1-a^{n+1-i}}{1-a}\varepsilon_i\right).$$

Thus, its mean is still 0 and its variance

$$\mathrm{Var}(\sqrt{n}\overline{Z}_n) = n\mathrm{Var}(\overline{Z}_n) = \frac{1}{n}\left[\frac{n(1-a^2) - 2a + 2a^{n+1}}{(1-a)^2(1-a^2)}\right] \to \frac{1}{(1-a)^2}, n \to \infty.$$

In general, we apply a 2-stage method for estimating the mean.

Method 5. Two-Stage Sampling Procedure

1) Run a small number of **trial runs** to obtain $\hat{\sigma}$ and $\hat{z}$;
2) Determine the number of repetitions n of **production runs** from the CI with targeted accuracy.
3) Simulate $Z_1, \cdots, Z_n$.

Remark. In the two-stage procedure, n is a random variable and *contains random error* in the trial simulation runs. In addition, in the relative accuracy case, the random error in the estimated $\hat{\sigma}^2$ and $\hat{z}$ are dependent. Rigorously, the **constructed CI should be adjusted**. There is a significant literature on this topic.

Another way is to simulate several times until the targeted accuracy, using practical approach that gradually increases sample size and updates CI (kind of heuristic).

Method 6. Sequential Sampling Procedure

1) Choose two positive integers m and k;
2) Simulate $Z_1, \cdots, Z_m$.
3) Compute the CI from simulation data.
4) While the CI does not meet the accuracy criteria: Simulate k more samples and recalculate the CI from all simulation data.

3.2.2 Estimating the Variance

In this sub-section, we consider the estimation of the variance for Non-i.i.d. samples.

A method usually used here is **independent replications**, i.e., with the s*ame computational budget*, we can simulate k independent replications of m samples. For each replication $i = 1, \cdots, k$, we compute the sample mean by

$$\widehat{W}_m^{(i)} = \frac{1}{m} \sum_{j=1}^{m} W_{i,j}.$$

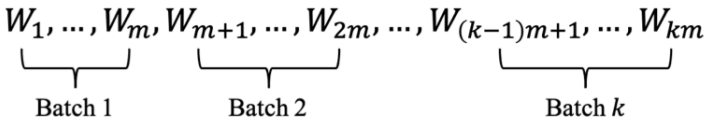
Every replications are *independent* and we can obtain sample replication mean and variance as:

$$\widehat{W}_m = \frac{1}{k} \sum_{i=1}^{k} \widehat{W}_m^{(i)}, \text{ and } S_R^2 = \frac{1}{k-1} \sum_{i=1}^{k} \left(\widehat{W}_m^{(i)} - \widehat{W}_m \right)^2.$$

Thus, here we can derive the $(1-\alpha)$ confidence interval

$$\left[\widehat{W}_n - \frac{S_R}{\sqrt{k}} t_{(k-1,1-\frac{\alpha}{2})}, \widehat{W}_n + \frac{S_R}{\sqrt{k}} t_{(k-1,1-\frac{\alpha}{2})} \right].$$

Another way is to use **sectioning / batch means**. we divide one long simulation run into k contiguous batches with m samples each ($n = mk$):



Similar to the independent replications, we can *consider a batch same as a "replication"*, then we obtain the i^{th} batch mean, mean and variance estimator as:

$$\overline{W}_m^{(i)} = \frac{1}{m}\sum_{j=1}^{m} W_{(i-1)m+j},$$

and

$$\widehat{W}_n = \frac{1}{k}\sum_{i=1}^{k}\overline{W}_m^{(i)}, \text{ and } S_B^2 = \frac{1}{k-1}\sum_{i=1}^{k}\left(\overline{W}_m^{(i)} - \widehat{W}_m\right)^2.$$

Thus, here we can derive the $(1-\alpha)$ confidence interval

$$\left[\widehat{W}_n - \frac{S_B}{\sqrt{k}}t_{(k-1,1-\frac{\alpha}{2})}, \widehat{W}_n + \frac{S_B}{\sqrt{k}}t_{(k-1,1-\frac{\alpha}{2})}\right].$$

Suppose our goal now is to estimate a smooth function $h(\cdot)$ at z. The value of z is unknown, but we can simulate i.i.d. samples of Z and $\mathbb{E}[Z] = z$.

Theorem 2. Delta Method

Let $\widehat{z}_n$ be the sample mean of n i.i.d. copies of Z and $\text{Var}(Z) = \sigma^2$. Then, for smooth $h(\cdot)$, we have

$$\sqrt{n}[h(\widehat{z}_n) - h(z)] \text{ converges in distribution to } N(0, h'(z)\sigma^2).$$

Remark.

- $h(\widehat{z}_n)$ is not necessarily an unbiased estimate of $h(z)$.
- If $h'(z)$ has a closed-form expression, we can estimate $h'(z)$ by $h'(\widehat{z}_n)$.
- Otherwise, we can use sectioning methods to directly estimate the variance without specifying $h'(z)$.

Eventually, sectioning method for Delta method works like: divide the data into k groups with m samples in each group,

$$\overline{z}_m^{(i)} = \frac{1}{m}\sum_{j=1}^{m} z_{(i-1)m+j},$$

If m large enough, the Delta method indicates

$$h(\overline{z}_m^{(i)}) \approx h(z) + \frac{\sigma}{\sqrt{m}}W_i, \text{ where i.i.d. } W_i \sim N(0,1).$$

Since $h(\widehat{z}_n) \approx h(z)$, we can estimate

$$\widehat{\sigma}^2 = \frac{m}{k-1}\sum_{i=1}^{k}\left[h(\overline{z}_m^{(i)}) - h(\widehat{z}_n)\right]^2$$

A third way is **bootstrap** to compute the confidence interval for any function of parameter z, by sampling with replacement from the empirical distribution. It is always applied in such cases:

- For some function h, it may be hard to compute $h'(z)$ if we want to use the Delta method
- Even if we simply want to estimate the mean, when the *data sample is small and contains "outliers"*, the normal confidence interval might not be good.

Method 7. Algorithm of Bootstrap

1. Input: data , confidence level α, number of resampling rounds m.
2. For i from 1 to m: draw n random samples $X'_1, \cdots, X'_k$ with replacement from $\{X_1, \cdots, X_n\}$, compute the estimate $Y_i = g(X'_1, \cdots, X'_k)$.
3. Compute the sample quantiles $\pi_{\frac{\alpha}{2}}$ and $\pi_{1-\frac{\alpha}{2}}$ of $\{Y_1, \cdots, Y_m\}$.
4. Output: CI $= [\pi_{\frac{\alpha}{2}}, \pi_{1-\frac{\alpha}{2}}]$.

Chapter 4

Variance Reductions

The motivation behind variance reduction is to decrease the **required number of samples** for a given accuracy level ε, and thus improving efficiency, (e.g., relative accuracy in estimation of the mean).

In actual simulation, it is hard to reduce variance without upgrading the hardware But in computer simulation, things becomes easier by *manipulating the input random variables*.

Due to the popularity of simulation-based method in machine learning and AI algorithm, variance reduction becomes a hot topic in the related areas.

4.1 Variance Reduction Techniques (VRTs)

4.1.1 Antithetic Sampling

Antithetic sampling is a technique by constructing **negatively correlated pairs** instead of i.i.d. samples to reduce the variance.

Method 8. Antithetic Sampling

Generate pairs $(Z_1, Z_2), (Z_3, Z_4), \cdots, (Z_{2n-1}, Z_{2n})$, such that:

- All Z_i's are **identically** distributed.
- The pairs are independent of one another.
- Within each pair k, $\text{Corr}(Z_{2k-1}, Z_{2k}) < 0$

To generate identically distributed negatively correlated pairs, one essential technique is to use **antithetic random numbers**, i.e., use $(U, 1-U)$ pair in the generation.

Example 8. Antithetic RN

e.g.1) In *inverse method*, generate $\{U_i\}_{i=1}^k$, let $(Z_{2i-1}, Z_{2i}) = [F^{-1}(U_i), F^{-1}(1-U_i)]$, $\forall$

$i = 1, \cdots, k$.

e.g.2) Suppose we were interested in using simulation to compute the expected value of $\theta = \mathbb{E}[e^U]$, we can still generate e^{U_k} and e^{1-U_k} for some pairs.

To use antithetic sampling, here are some notes:

Remark.
- It saves half of the effort generating random numbers.
- If the RV is generated from some other **monotonically increasing function** h, then antithetic sampling can also be used.
- It is Not often used with acceptance-rejection.
- Be careful when h is not monotone or when you have a complicated RV generation process.

4.1.2 Controling Variate

To estimate $\mathbb{E}(Z)$, use $Z + c(Y - y)$ (UBE) instead of Z alone (recall controling variable techs in point estimation), where $y = \mathbb{E}(Y)$ is in hand. Suppose Y, Z are dependent, variance can be reduced:

$$\mathrm{Var}[Z + c(Y - y)] = \mathrm{Var}(Z) + c^2\mathrm{Var}(Y) + 2c\mathrm{Cov}(Z, Y),$$

$c = -\dfrac{\mathrm{Cov}(Z, Y)}{\mathrm{Var}(Y)}$ minimizes the variance to

$$\mathrm{Var}(Z) - \frac{\mathrm{Cov}^2(Z, Y)}{\mathrm{Var}(Y)} < \mathrm{Var}(Z).$$

Remark.
- Y is called a **control variate** for the simulation estimator Z.
- $\dfrac{\mathrm{Var}[Z + c(Y - y)]}{\mathrm{Var}(Z)} = 1 - \mathrm{Corr}(Z, Y)^2$, meaning that variance reduction percentage is relevant to their correlations.
- The idea behind is to correct the samples Z by adding/subtracting some value negatively/positively related to it.

4.1.3 Conditioning

The idea of conditioning comes from **theorem of iterated variance**. Choose proper X, and condition Z on X, that is, sample $\mathbb{E}[Z|X]$ (i.e., sample X instead of Z). (Precautions and examples in STA2002 Note).

Example 9. Simulation of Option Price

A **barrier option** (path-dependent exotics) is an option whose payoff is conditional upon the underlying asset's price breaching a barrier level during the option's lifetime.

The payoff of the barrier option at expiration time T is

$h(S_{(\frac{T}{2})}, S_{(T)}) = \begin{cases} (S_{(T)} - K_1)_+ & S_{(\frac{T}{2})} \leq L \\ (S_{(T)} - K_2)_+ & \text{otherwise} \end{cases}$, where $S_{(t)}$ is a geometric Brownian motion.

Given the risk-free interest rate r, the price of the option is

$$C_B = \mathbb{E}^Q[e^{-rT} h(S_{(\frac{T}{2})}, S_{(T)})].$$

Conditional on $S_{(\frac{T}{2})}$ compute C_B the same as that of a call option utilizing the Black-Scholes formula.

4.1.4 Stratified Sampling

In statistics, stratified sampling is a method of sampling from a population which can be partitioned into subpopulations. The idea of stratified sampling is also from **theorem of iterated variance**, since $\mathbb{E}[\text{Var}(Z|X)] < \text{Var}(Z)$.

Suppose X is a discrete random variable such that $p_k = P(Y = k)$ is known for all k (**strata**), and $Z|Y = k$, $\forall\ k$ from the conditional distribution can be sampled.

Let n be the number of total samples. For each k, we generate $n_k = np_k$ samples from $Z|Y = k$ and calculate their mean as $\overline{Z}_k$. We can then directly get an UBE of Z as

$$\widehat{z}_n^{\text{st}} = \sum_k \overline{Z}_k p_k.$$

Here the variance is reduced to (independence of samples)

$$\text{Var}(\widehat{z}_n^{\text{st}}) = \frac{\mathbb{E}[\text{Var}(Z|X)]}{n} < \frac{\text{Var}(Z)}{n} = \text{Var}(\overline{Z}_n).$$

This saving can be substantial when *the value of X strongly affects the conditional expectation of Z*.

Note that the sample variance of np_k samples S_k^2 is an unbiased estimator of $\text{Var}(Z|Y = k)$, and consequently, $\frac{1}{n}\sum_k p_k S_k^2$ is an unbiased estimator of $\text{Var}(\widehat{z}_n^{\text{st}})$.

Optimal Stratified Sampling

To decide optimal n_k (instead of np_k) given n, one can solve the below optimization function.

$$\begin{aligned} \min_{n_k} \quad & \sum_k \frac{p_k^2 \sigma_k^2}{n_k} \\ \text{s.t.} \quad & \sum_k n_k = n \\ & n_k \geq 0, \forall\ k \end{aligned}$$

From this we know that $n_k^\star = \frac{np_k}{\frac{1}{\sigma_k}\sum_i p_i \sigma_i}$, $\forall\ k$. Nevertheless, we do not know the exact values of σ_k in practice.

Remark.

- If you think σ_k varies a lot, you can run some trial simulations to estimate σ_k for each k. This will introduce extra simulation cost.
- If you think σ_k does not vary a lot, you can assume they are equal and simply use $n_k = np_k$.

4.1.5 Importance Sampling

Importance sampling is a Monte Carlo method for **evaluating properties of a particular distribution**, while *only having samples generated from a different distribution than the distribution of interest.*

Let $X : \Omega \to \mathbb{R}$ be a random variable in some probability space $(\Omega, \mathcal{F}, P)$. We wish to estimate the expected value of X under P, denoted $\mathbb{E}[X; P]$.

The basic idea of importance sampling is to *sample the states from a different distribution to lower the variance of the estimation* of $\mathbb{E}[X; P]$, or when sampling from P is difficult. This is accomplished by first choosing a random variable $L \geq 0$ such that $\mathbb{E}[L; P] = 1$. and that P-almost everywhere $L(\omega) \neq 0$. With the variable L we define a probability $P^{(L)}$ that satisfies

$$\mathbb{E}[X; P] = \mathbb{E}\left[\frac{X}{L}; P^{(L)}\right].$$

Method 9. Importance Sampling

Suppose one want to estimate $z = \mathbb{E}[h(X)]$. Presuppose that X has p.d.f. $f(\cdot)$ and Y has p.d.f. $g(\cdot)$.

1. Generate $Y_1, \cdots, Y_n$ i.i.d (as Y).
2. Estimate $\widehat{z}_n = \dfrac{1}{n}\sum_{i=1}^{n} \dfrac{h(Y_i)f(Y_i)}{g(Y_i)}$.
3. $\widehat{z}_n$ is an UBE of z.

$\dfrac{f}{g}(\cdot)$ is called the **likelihood ratio**, whose value affects the variance of the sampling result.

The reduced variance is then

$$\mathrm{Var}(\overline{h(X)}) - \mathrm{Var}(\widehat{z}_n) = \frac{\displaystyle\int_\Omega h(y)^2 f(y)\left[1 - \frac{f(y)}{g(y)}\right] dy}{n^2}.$$

In order to achieve a good reduction, the integral in equation should be positive:

- $\dfrac{f(x)}{g(x)} > 1$ when $h(x)^2 f(x)$ is small.
- $\dfrac{f(x)}{g(x)} \leq 1$ when $h(x)^2 f(x)$ is large.

Note that if $g(x) = \dfrac{f \cdot h(x)}{z}$ is chosen, the variance is reduced to 0. However, one cannot know exact z (otherwise no need to do sampling). According to the zero variance case, the rule of thumb appears to be: choose a $g(\cdot)$ that has *a similar shape to* $h(\cdot)f(\cdot)$.

In particular, we could choose so that g and $h \cdot f$ both take on their maximum values at the same value x_m. This is called the **maximum principle**.

Example 10. Sampling for Rare Events

At the beginning of the financial crisis in August 2007, Goldman Sachs Asset Management's (GSAM) Global Alpha fund incurred steep losses. In explaining these losses, the CFO of Goldman Sachs claimed they had seen 25 standard deviation moves several days in a row.

Suppose we wish to estimate $z = P(X > 25)$ where $X \sim N(0, 1)$.

1. Use $\dfrac{1}{n}\sum_i \mathbb{1}_{\{X_i > 25\}}$ as the statistics: need $n \approx 3.26 \times 10^{137}$ samples in order to obtain just one non-zero value of I.
2. $Y \sim N(25, 1)$ with importance sampling, instead.

Sampling for Conditional Expectation

We have seen importance sampling works efficaciously in simulating rare events. Hence, it does well in conditional expectation simulation, on rare events.

Suppose we want to estimate $z = \mathbb{E}[h(X)|X \in A]$ where A is a rare event explicitly involves X. The sampling of z with intentionally chosen $Y \sim g(\cdot)$, should be

$$\frac{\sum_{i, Y_i \in A} h(Y_i)\frac{f(Y_i)}{g(Y_i)}}{\sum_{i, Y_i \in A} \frac{f(Y_i)}{g(Y_i)}}.$$

Exponential Tilting (ET) / Exponential Change of Measure (ECM)

For light-tailed distribution (compared with exponential family), a common way to design the importance distribution is to do distribution shifting using ET technique. The different exponential tiltings of a random variable X is known as the **natural exponential family** of X.

Given a random variable X with probability distribution $\mathbb{P}$, density f, and moment generating function (MGF)

$$M_X(\theta) = \mathbb{E}[e^{\theta X}] < \infty,$$

the exponentially tilted measure $\mathbb{P}_\theta$ is defined as follows:

$$\mathbb{P}_\theta(X \in dx) = \frac{\mathbb{E}[e^{\theta X}\mathbb{1}_{\{X \in dx\}}]}{M_X(\theta)} = e^{\theta x - \kappa(\theta)}\mathbb{P}(X \in dx),$$

where $\kappa(\theta)$ is the **cumulant generating function (CGF)**defined as

$$\kappa(\theta) = \ln \mathbb{E}[e^{\theta X}] = \ln M_X(\theta).$$

It has some advantages:

- In many cases, the tilted distribution belongs to the same parametric family as the original (particularly for exponential family). Since the θ -tilted density of X satisfies $f_\theta(x) \propto e^{\theta x} f(x)$.
- Relationship between the original and tilted CFG:

$$\kappa_\theta(\eta) = \log(\mathbb{E}_\theta[e^{\eta X}]) = \kappa(\theta + \eta) - \kappa(\theta).$$

  Thus, one can calculate $\mathbb{E}_\theta[X] = \kappa'(\theta)$ and $\mathrm{Var}_\theta[X] = \kappa''(\theta)$.

Exponential Tilting is used in Monte Carlo Estimation for *rare-event simulation*, and *rejection and importance sampling* in particular. Below is a simple example in importance sampling.

Example 11. ET in Importance Sampling

Assume independent and identically distributed $\{X_i\}$ such that $\kappa(\theta) < \infty$. In order to estimate $\mathbb{P}(X_1 + \cdots + X_n > c)$, we can employ importance sampling by taking

$$h(X) = \mathbb{1}_{(\sum_{i=1}^n X_i > c)}.$$

The constant c can be rewritten as na for some other constant a. Then,

$$\mathbb{P}(\sum_{i=1}^n X_i > na) = \mathbb{E}_{\theta_a}\left[\exp\{-\theta_a \sum_{i=1}^n X_i + n\kappa(\theta_a)\}\mathbb{1}_{(\sum_{i=1}^n X_i > na)}\right],$$

where θ_a denotes the θ defined by the saddle-point equation

$$\kappa'(\theta_a) = a.$$

4.2 Comparing Systems

One of the most important uses of simulation output analysis regards the comparison of competing systems or alternative system configurations.

4.2.1 Common Random Numbers

One can use **common random numbers (CRN)** to create such pair for the simulation of two systems. CRN has also been called *correlated sampling*, *matched streams* or *matched pairs*.

CRN requires *synchronization of the random number streams*, which ensures that in addition to using the same random numbers to simulate all configurations, a specific random number used for a specific purpose in one configuration is used for exactly the same purpose in all other configurations.

Suppose X_{1j} and X_{2j} are the observations from the first and second configurations on the j-th independent replication. We want to estimate

$$\xi = E(X_{1j}) - E(X_{2j}) = \mu_1 - \mu_2.$$

If we perform n replications of each configuration and let

$$Z_j = X_{1j} - X_{2j} \quad \text{for } j = 1, 2, \ldots, n,$$

then $E(Z_j) = \xi$ and $Z(n) = \frac{\sum_{j=1,\ldots,n} Z_j}{n}$ is an unbiased estimator of ξ. And since the Z_j's are independent identically distributed random variables,

$$\text{Var}[Z(n)] = \frac{\text{Var}(Z_j)}{n} = \frac{\text{Var}[X_{1j}] + \text{Var}[X_{2j}] - 2\,\text{Cov}[X_{1j}, X_{2j}]}{n}.$$

In case of independent sampling, i.e., no common random numbers used then $Cov(X_{1j}, X_{2j}) = 0$. But if we succeed to induce an element of positive correlation between X_1 and X_2 such that $Cov(X_{1j}, X_{2j}) > 0$, it can be seen from the equation above that the variance is reduced.

It can also be observed that if the CRN induces a negative correlation, i.e., $Cov(X_{1j}, X_{2j}) < 0$, this technique can actually backfire, where the variance is increased and not decreased (as intended).

4.2.2 Comparing Multiple Systems

Suppose now we are interested in selecting the best of a number ≥ 2 of competing processes. We want to specify the desired probability of correctly selecting the best process. A method called **ranking and selection** is used here.

R&S selects the best system, or a subset of systems that includes the best, with a probabilistic guarantee of a correct selection.

Formal problem formulation:

- Suppose there are k systems, which are denoted as system $1, \cdots, k$.
- We would like to compare a parameter z and we assume that a larger z is better.
- Suppose the ordered (but unknown) z_i 's are $z_{[1]} \leq \cdots \leq z_{[k]}$
- We are interested in finding the best system, $[k]$.

Example 12. Normal with The Largest Mean

We would like to give procedures for selecting that one of k normal distributions has the largest mean, with assumptions that the k systems are mutually independent.

A **correct selection (CS)** is made if the goal is achieved. We use an approach called "*indifference-zone*".

(1) For specified constants $P^\star$, $\delta^\star$ with $\delta^\star > 0$ and $\frac{1}{k} < P^\star < 1$, we require:

$$P(CS) \geq P^\star \text{ whenever } \mu_{[k]} - \mu_{[k-1]} \geq \delta.$$

(2) The constant $\delta^\star$ can be considered as the smallest difference worth detecting.

(3) The probability P(CS) depends on the differences $\mu_i - \mu_j$ for every pair $i, j = 1, \cdots, k$, the sample size n_i and σ_i^2 for each $i = 1, \cdots, k$.

Parameter configurations $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_k)$ satisfying $\mu_{[k]} - \mu_{[k-1]} \geq \delta^\star$ are in the **preference-zone** for a correct selection. Otherwise, they are in the **indifference-zone**.

There are many of such procedures and we highlight a few:

- Single-stage procedure
- Two-stage procedure
- Sequential procedure

Method 10. Single-Stage R&S Procedure (Normal)

Assumption: Systems have *equal known variance*.

(i) For a given k and specified $P^\star$, $\delta^\star$, and σ, first determine the sample size n.

(ii) Sample n outputs Y_{ij}, $j = 1, \cdots, n$ independently from each system $i = 1, \cdots, k$.

(iii) Select the system that yields the largest sample mean, as the one associated with $\mu_{[k]}$.

The **least favorable configuration** minimizes $P(CS)$ among all $\boldsymbol{\mu}$ in the preference-zone.

For the single-stage R&S, it is for sure

$$\mu_{[1]} = \cdots = \mu_{[k-1]} = \mu_{[k]} - \delta^\star.$$

One needs to make sure that $P(CS|LFC) \geq P^\star$, that is,

$$P^\star \leq P_{LFC}\left(\frac{\overline{Y}_{[i]} - \mu_{[k]}}{\sigma/\sqrt{n}} \leq \frac{\overline{Y}_{[k]} - \mu_{[k]}}{\sigma/\sqrt{n}}, \forall\ i \neq k\right) = \int_{\mathbb{R}} \phi^{k-1}\left(x + \frac{\sqrt{n}\delta^\star}{\sigma}\right)\phi(x)dx.$$

Let $c_{k,P^\star}$ (can be found in some table) satisfy

$$P^\star = \int_{\mathbb{R}} \phi^{k-1}(x + c_{k,P^\star})\phi(x)dx,$$

then

$$n = \lceil \left(\frac{c_{k,P^\star}\sigma}{\delta^\star}\right)^2 \rceil.$$

The single-Stage R&S procedure is limited by its requirement of independence of every Y_i and equal known variance σ^2. Two-stage R&S procedure appears for unknown and unequal variances.

Method 11. Two-Stage R&S Procedure (Normal)

(i) For a given k and specified $P^\star$, $\delta^\star$, and a common first-stage sample size $N_0 \geq 2$,

$$P^\star = \int_0^\infty \left\{\int_0^\infty \phi\left(\frac{h_{k,P^\star,N_0}}{[(N_0-1)(x^{-1}+y^{-1})]^{\frac{1}{2}}}\right) f(x)dx\right\}^{k-1} f(y)dy,$$

where $f(\cdot)$ is the $\chi^2_{N_0-1}$ pdf.

(ii) Sample N_0 outputs Y_{ij}, $j = 1, \cdots, N_0$ independently from each system $i = 1, \cdots, k$; Calculate the first-stage sample means $\overline{Y}_i(N_0)$, and sample variances S_i^2. Eventually, calculate the final sample sizes

$$N_i = \max\left\{N_0, \lceil \left(\frac{h_{k,P^\star,N_0}}{\delta^\star}\right)^2 S_i^2 \rceil\right\}.$$

(iii) Take $N_i - N_0$ additional i.i.d. samples Y_{ij}, $j = 1, \cdots, N_i - N_0$ from system i independently of the first-stage sample and the other systems.

(iv) Compute overall sample means $\overline{Y}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} Y_{ij}$ and select the system largest $\overline{Y}_i$ as the one associated with $\mu_{[k]}$.

Consider again

$$P^\star \leq P(CS|LFC) = P_{LFC}\left\{\frac{\overline{Y}_{[i]} - \overline{Y}_{[k]} + \delta^\star}{\left(\frac{\sigma_{[i]}^2}{N_i} + \frac{\sigma_{[k]}^2}{N_k}\right)^{1/2}} \leq \frac{\delta^\star}{\left(\frac{\sigma_{[i]}^2}{N_i} + \frac{\sigma_{[k]}^2}{N_k}\right)^{1/2}}, \forall\ i \neq k\right\}.$$

Let

$$Z_i = \frac{\overline{Y}_{[i]} - \overline{Y}_{[k]} + \delta^\star}{\left(\frac{\sigma_{[i]}^2}{N_i} + \frac{\sigma_{[k]}^2}{N_k}\right)^{1/2}}, Q_i = \frac{h_{k,P^\star,N_0}}{\left(\frac{\sigma_{[i]}^2}{S_{[i]}} + \frac{\sigma_{[k]}^2}{S_{[k]}}\right)^{1/2}}.$$

Since $N_i \geq \frac{h_{k,P^\star,N_0}^2 S_{[i]}^2}{\delta^{\star 2}}$, $Q_i \leq \frac{\delta^\star}{\left(\frac{\sigma_{[i]}^2}{N_i} + \frac{\sigma_{[k]}^2}{N_k}\right)^{1/2}}$, a stronger version is that $P^\star \leq P_{LFC}(Z_i \leq Q_i, \forall\ i \neq k)$.

Here, $Z_1, \cdots, Z_n$ are multi-variate normal, with marginal distribution being $N(0,1)$ (under LFC). By **Slepian's Inequality**,

$$P_{LFC}(Z_i \leq Q_i, \forall\ i \neq k | S_{[1]}^2, \cdots, S_{[k]}^2) \geq \prod_{i=1}^{k-1} P_{LFC}(Z_i \leq Q_i | S_{[1]}^2, \cdots, S_{[k]}^2).$$

Thus,

$$P_{LFC}(Z_i \leq Q_i, \forall\ i \neq k) = \mathbb{E}(P_{LFC}(Z_i \leq Q_i, \forall\ i \neq k)|S_{[1]}, \cdots, S_{[k]}) \geq \mathbb{E}(\prod_{i=1}^{k-1}\phi(Q_i)).$$

$$\text{RHS} = \mathbb{E}[\mathbb{E}(\prod_{i=1}^{k-1}\phi(Q_i))|\tfrac{(N_0-1)S_{[k]}^2}{\sigma_{[k]}}] = \int_0^\infty \left\{\int_0^\infty \phi\left(\frac{h_{k,P^\star,N_0}}{[(N_0-1)(x^{-1}+y^{-1})]^{\frac{1}{2}}}\right) f(x)dx\right\}^{k-1} f(y)dy.$$

We can improve the efficiency by not having to take many samples from every system. We only take one sample at a time from each system, and eliminates systems that appear to be noncompetitive along the way. The idea is developed into a strategy called sequential procedure.

Method 12. Sequential R&S Procedure (Normal)

(i) Set surviving systems as I, initialized as $\{1, \cdots, k\}$. Take an initial $N_0 \geq 2$ samples Y_{ij} from system i. Compute ($c \in \mathbb{N}^+$)

$$h^2 = c(N_0 - 1)\left(\left(\frac{2 - 2P^\star}{k-1}\right)^{-\frac{2}{N_0-1}} - 1\right).$$

(ii) For system i, we calculate the sample mean and *the sample variance of the difference* between system i and l when $l \neq i$:

$$S_{il}^2 = \frac{1}{N_0 - 1}\sum_{i=1}^{N_0}\left(Y_{ij} - Y_{lj} - \left(\overline{Y}_i(N_0) - \overline{Y}_j(N_0)\right)\right)^2.$$

For $i \neq l$, set $N_{il} = \lfloor \frac{h^2 S_{il}^2}{\delta^{\star 2}} \rfloor$ and $N_i = \max_{i \neq l} N_{il}$.

(iii) If $N_0 > \max_i N_i$, then we can stop and select the system with the largest sample mean. If not, we set the sequential counter $r = N_0$ and do the screening phase.

(iv) Screening: select the system i from old I to form a new I such that:

$$\overline{Y}_i(r) \geq \overline{Y}_l(r) - W_{il}(r),$$

where $W_{il}(r) = \max\left\{0, \frac{\delta^\star}{2cr}\left(\frac{h^2 S_{il}^2}{\delta^{\star 2}} - r\right)\right\}$.

(v) Stop when $|I| = 1$. Otherwise, take one additional observation $Y_{i,r+1}$ from each system $i \in I$ and set $r = r + 1$. If $r = \max_i N_i + 1$, then stop and select the system whose index is in I and has the largest $\overline{Y}_i(r)$ as the best. Otherwise, go to Screening.

Example 13. Bernuolli with The Largest Success Probability

Use "indifference-zone" approaches to give procedures for selecting that one of k Bernuolli distributions has the largest success probability, with assumptions that the k systems are mutually independent.

Method 13. Single-Stage R&S Procedure (Bernuolli)

(i) For a given k and specified $P^\star$, $\delta^\star$, find $n = n(k, P^\star, \delta^\star)$ from the table.

(ii) We sample n outputs X_{ij}, $j = 1, \cdots, n$ independently from each system $i = 1, \cdots, k$.

(iii) Calculate the k sample sums, $Y_{i,n} = \sum_{j=i}^{n} X_{ij}$ for $i = 1, \cdots, k$.

(iv) Select the system that yields the largest sample sum; in the case of ties, ran-

domize.

To reduce the samples, one does the single-stage procedure, except stop sampling when the system in second place can **at best tie**. This is called **curtailment**, you might as well stop because it will not be possible for the outcome to change. It turns out curtailment gives the same $P(CS)$ as the single-stage procedure, but a lower expected number of observations.

For a sequential R&S procedure of Bernoulli selection, we use a new indifference-zone can be set up as: for specified constants $P^\star$ and $\theta^\star$ with $\frac{1}{k} < P^\star < 1$ and $\theta^\star > 1$:

$$P(CS) \geq P^\star, \text{ whenever } \frac{p_{[k]}}{1-p_{[k]}} \frac{1-p_{[k-1]}}{p_{[k-1]}} \geq \theta^\star.$$

Method 14. Sequential R&S Procedure (Bernuolli)

(i) For a given k and specified $P^\star$, $\theta^\star$, send $m = 1$ ($m+ = 1$ every step).

(ii) We sample k outputs X_{im}, $i = 1, \cdots, k$ independently.

(iii) Calculate the k sample sums, $Y_{i,m} = \sum_{j=i}^{m} X_{ij}$ for $i = 1, \cdots, k$. Denote the ordered sums by $Y_{[1],m} \leq \cdots \leq Y_{[k],m}$

(iv) Stop if

$$Z_m = \sum_{i=1}^{k-1} \theta^{\star(Y_{[i],m} - Y_{[k],m})} \leq \frac{1}{P^\star} - 1.$$

Select the system that yields the $Y_{[k],m}$; in the case of ties, randomize.

Example 14. Multi-nomial with The Largest Probability

A k-variate discrete multi-nomial random variable $\boldsymbol{Y} = (Y_1, Y_2, \cdots, Y_k)$ has the probability mass function:

$$f(y_1, \cdots, y_k) = \frac{n!}{\prod_{j=1}^{k} y_j!} \prod_{i=1}^{k} p_i^{y_k}, \text{ where } \sum_{j=1}^{k} y_j = n \text{ and } \sum_{j=1}^{k} p_j = 1.$$

To use the single-stage R&S procedure here, one new indifference-zone will be introduced as: or specified constants $P^\star$ and $\theta^\star$ with $\frac{1}{k} < P^\star < 1$ and $\theta^\star > 1$:

$$P(CS) \geq P^\star, \text{ whenever } \frac{p_{[k]}}{p_{[k-1]}} \geq \theta^\star.$$

Method 15. Single Stage R&S Procedure (Multi-nomial)

(i) For the given k,$P^\star$, and $\theta^\star$, find $n = n(k, P^\star, \theta^\star)$ from the table.

(ii) Take n multinomial samples $\boldsymbol{X}_j = (X_{1j}, \cdots, X_{kj}), j = 1, \cdots, n$ where each

$X_{ij} \in \{0, 1\}$ if category i does (not) occur in j^{th} sample.

(iii) Calculate the cumulative sum for category i: $Y_{i,n} = \sum_{j=1}^{n} X_{ij}$ and order the sample sums $Y_{[1],n} \leq \cdots \leq Y_{[k],n}$.

(iv) Select the category with the largest sum, $Y_{[k],n}$, as the one associated with $p_{[k]}$, randomizing to break ties.

Chapter 5

Seleted Topics

5.1 Multi-Armed Bandits

In probability theory and machine learning, the **multi-armed bandit problem** (sometimes called the **K-** or **N-armed bandit problem**) is a problem in which a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice.

Figure 5.1: Illustration of MAB

5.1.1 Basic Multi-armed Bandit Model (MAB)

The multi-armed bandit (short: bandit or MAB) can be seen as a set of real distributions $B = \{R_1, \ldots, R_K\}$, each distribution being associated with the rewards delivered by one of the $K \in \mathbb{N}^+$ levers. Let $\mu_1, \ldots, \mu_K$ be the mean values associated with these reward distributions. The gambler iteratively plays one lever per round and observes the associated reward. The objective is to maximize the sum of the collected rewards.

The ***horizon*** H is the number of rounds that remain to be played. The bandit problem is formally equivalent to a one-state Markov decision process. The ***regret*** ρ after T rounds is defined as the expected difference between the reward sum associated with an optimal strategy and the sum of the collected rewards:

$$\rho = T\mu^\star - \sum_{t=1}^{T} \widehat{r}_t,$$

where $\mu^\star$ is the maximal reward mean, $\mu^\star = \max_k\{\mu_k\}$, and $\widehat{r}_t$ is the reward in round t.

A *zero-regret strategy* is a strategy whose average regret per round ρ/T tends to zero with probability 1 when the number of played rounds tends to infinity. Intuitively, zero-regret strategies are guaranteed to converge to a (not necessarily unique) optimal strategy if enough rounds are played.

5.1.2 Approximated Solutions

Sub-uniform Strategies

All those strategies have in common a *greedy behavior* where the best lever (based on previous observations) is always pulled except when a (uniformly) random action is taken.

- **Epsilon-greedy strategy**: The best lever is selected for a proportion $1 - \epsilon$ of the trials, and a lever is selected at random (with uniform probability) for a proportion ϵ.
- **Epsilon-first strategy**: A pure exploration phase is followed by a pure exploitation phase. For N trials in total, the exploration phase occupies ϵN trials and the exploitation phase $(1-\epsilon)N$ trials. During the exploration phase, a lever is randomly selected (with uniform probability); during the exploitation phase, the best lever is always selected.
- **Epsilon-decreasing strategy**: Similar to the epsilon-greedy strategy, except that the value of ϵ decreases as the experiment progresses, resulting in highly explorative behaviour at the start and highly exploitative behaviour at the finish.
- **Adaptive epsilon-greedy strategy based on value differences (VDBE)**: Similar to the epsilon-decreasing strategy, except that epsilon is reduced on basis of the learning progress instead of manual tuning. High fluctuations in the value estimates lead to a high epsilon (high exploration, low exploitation); low fluctuations to a low epsilon (low exploration, high exploitation). Further improvements can be achieved by a softmax-weighted action selection in case of exploratory actions
- **Adaptive epsilon-greedy strategy based on Bayesian ensembles (Epsilon-BMC)**: An adaptive epsilon adaptation strategy for reinforcement learning similar to VBDE, with monotone convergence guarantees. In this framework, the epsilon parameter is viewed as the expectation of a posterior distribution weighting a greedy agent (that fully trusts the learned reward) and uniform learning agent (that distrusts the learned reward). This posterior is approximated using a suitable Beta distribution under the assumption of normality of observed rewards.

Upper Confidence Bounds

To avoid inefficient ϵ-greedy exploration, one approach is ϵ-decreasing, and the other is to be optimistic about options with *high uncertainty* and thus to prefer actions for which we haven't had a confident value estimation yet. Or in other words, we favor exploration of actions with a strong potential to have a optimal value.

The **Upper Confidence Bounds** (UCB) algorithm is introduced, which measures this potential by an upper confidence bound of the reward value, i.e., $\widehat{r}_t^{(n)} \leq UCB(t,n) = \widehat{r}_t^{(n)} + \widehat{w}_t^{(n)}$ with high probability. In UCB algorithm, we always select the greediest action to maximize the upper confidence bound:

$$\arg\max_t \widehat{r}_t^{(n)} + \widehat{w}_t^{(n)}.$$

Hoefdding's inequality plays a vital role in the choice of UCB (or the confidence radius), since it gives the relationship (bounds) bewteen confidence interval and significance level.

Theorem 3. Hoefdding's Inequality

Let $Z_1, \cdots, Z_n$ be independent bounded random variables with $Z_i \in [a,b]$ for all i, where $-\infty < a \leq b < \infty$. Then

$$P\left[\frac{1}{n}\left(\sum_{i=1}^{n} Z_i - \mathbb{E}(Z_i)\right) \geq t\right] \leq \exp\left\{-\frac{2nt^2}{(b-a)^2}\right\}.$$

Proof. Use Hoeffding's lemma: $\mathbb{E}\left[e^{\lambda(Z-\mathbb{E}[Z])}\right] \leq \exp\left\{\frac{\lambda(b-a)^2}{8}\right\}$. ■

With the above inequality, under a tiny threshold δ, a reasonable choice of $\widehat{w}_t^{(n)}$ can be chosen with bounded reward distributions.

Probability Matching Strategies

Probability matching strategies reflect the idea that the number of pulls for a given lever should match its actual probability of being the optimal lever. Probability matching strategies are also known as **Thompson sampling** or **Bayesian Bandits**, and are surprisingly easy to implement if you can sample from the posterior for the mean value of each alternative.

Probability matching strategies also admit solutions to so-called contextual bandit problems.

Thompson sampling consists in playing the action $a^* \in \mathcal{A}$ according to the probability that it maximizes the expected reward; action a^* is chosen with probability

$$\int \mathbb{1}\left[\mathbb{E}(r|a^*,x,\theta) = \max_{a'} \mathbb{E}(r|a',x,\theta)\right] P(\theta|\mathcal{D})d\theta,$$

where past observations triplets $\mathcal{D} = \{(x;a;r)\}$, a set Θ of parameters θ of the distribution of r.

Pricing Strategies

Pricing strategies establish a price for each lever. For example, as illustrated with the POKER algorithm, the price can be the sum of the expected reward plus an estimation of extra future rewards that will gain through the additional knowledge. The lever of highest price is always pulled.

5.1.3 Contextual Bandit

A useful generalization of the multi-armed bandit is the **contextual multi-armed bandit**. At each iteration an agent still has to choose between arms, but they also see a d-dimensional feature vector, the **context vector** they can use together with the rewards of the arms played in the past to make the choice of the arm to play. Over time, the learner's aim is to collect enough information about how the context vectors and rewards relate to each other, so that it can predict the next best arm to play by looking at the feature vectors.

People use the idea of "function approximation" to model the relation between rewards, arms and contexts:

$$\widehat{r}_t = f_\theta(a_t, \boldsymbol{c}_t) + \eta_t,$$

where a_t is the arm; $\boldsymbol{c}_t$ represents the context vector and η_t is a random, zero-mean noise.

Many strategies exist that provide an approximate solution to the contextual bandit problem, and can be put into two broad categories: **online linear** and **online non-linear**.

5.2 Derivarive Estimation

5.2.1 Gradient Simulation

Suppose one wants to do optimization in the random sense, that is

$$\min_\theta \mathbb{E}_\xi[z(\theta;\xi)],$$

where ξ is a random variable and θ is a parameter.

There are three main methods for gradient simulation:

(i) **Finite Difference (FD)** Method: model-free

(ii) **Pathwise Method / Infinitesimal Perturbation Analysis (IPA)**: model-dependent but efficient

(iii) **Likelihood Ratio (LR)** Method: : model-dependent but efficient

Let $z(\theta) = \mathbb{E}_\xi[z(\theta;\xi)]$.

Method 16. Finite Difference

1. Choose a sample size n and a finite step size h.

2. Generaten samples, ξ_i^+, ξ_i^-, $i = 1, \cdots, n$, corresponding to $\theta + h$ and $\theta - h$, respectively.
3. Estimate $z'(\theta)$ as

$$\widehat{g}_n^{\mathrm{FD}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \frac{z(\theta + h; \xi_i^+) - z(\theta - h; \xi_i^-)}{2h}.$$

Remark. Bias and Variance of the finite difference algorithm:

- Bias: Assume $z(\theta)$ to be analytic, $\mathbb{E}_\xi\left[\widehat{g}_n^{\mathrm{FD}}(\theta)\right] - z'(\theta) = O(h^2)$.
- Variance: Suppose all ξs are sampled independently,

$$\mathrm{Var}_\xi(\widehat{g}_n(\theta)) = \frac{\mathrm{Var}_\xi\left[z(\theta + h; \xi)\right] + \mathrm{Var}_\xi\left[z(\theta - h; \xi)\right]}{4nh^2}.$$

- Given the total number of simulation samples n (with above assumptions), to minimize MSE, the optimal choice of h satisfies

$$h^\star = O(n^{-\frac{1}{6}}),$$

  with which $\sqrt{\mathrm{MSE}} = O(n^{-\frac{1}{3}})$.
- Use common random number (i.e., $\xi_i^+ = \xi_i^-$). It helps in reducing the variance (e.g., if variance is reduced to $O(h^{-1})$, optimal choice of $h^\star$ becomes $O(n^{-\frac{1}{5}})$ and corresponding $\sqrt{\mathrm{MSE}} = O(n^{-\frac{2}{5}})$.)
- Variant methods:

  (1) **Gaussian Smoothing**: Suppose $W \sim N(0, 1)$. Define $G(\theta; W) = \frac{\sqrt{2\pi}}{h} z(\theta + h \cdot W)$, then $\mathbb{E}_W[G(\theta; W)] \approx g(\theta)$.

  (2) **Kernel Methods**: Let $k(x)$ be a kernel (typically a smooth and symmetric probability density). Do estimations

$$z(\theta) \approx \frac{1}{nh}\sum_{r=0}^{n} z\left(\frac{r}{n}; \xi_r\right) k\left[\frac{\theta - r/n}{h}\right],$$

  and

$$z'(\theta) \approx \frac{1}{(n+1)h^2}\sum_{r=0}^{n} z\left(\frac{r}{n}; \xi_r\right) k'\left[\frac{\theta - r/n}{h}\right].$$

Suppose the closed form of $\dfrac{\partial z}{\partial \theta}$ is in hand.

Method 17. Pathwise

1. Simulate n i.i.d. samples of $z'(\theta; \xi_i)$, $i = 1, \cdots, n$ and

2. Calculate the following estimator for the gradient $z'(\theta)$:

$$\widehat{g}_n^{\mathrm{IPA}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} z'(\theta;\xi_i).$$

Remark.

- Consistency (UBE): Assume $z(\theta;\xi)$ is partial differentiable at θ_0 and Lipschitz continuous (with Lipschitz constant L satisfying $\mathbb{E}[L] < \infty$) in some neighborhood of θ_0. Then $\mathbb{E}_\xi[z_\theta(\theta_0;\xi)] = z'(\theta_0)$, and $\widehat{g}_n^{\mathrm{IPA}}(\theta_0) \to z'(\theta_0)$.
- Variance: standard deviation of IPA is $O(n^{-\frac{1}{2}})$, prefer to FD.
- IPA is widely used in data-based numerical methods. For example, using backpropagation to compute gradient of NN for given data is essentially an IPA method.

Now consider a new case that the random variable ξ has some dependency of the parameter θ. Thus, one can simplify the notation $Z(\theta) = z[\theta;\xi(\theta)]$, where Z is a random variable dependent on θ. The key feature is that the dependence of the expectation on θ is expressed in the measure $\mathbb{P}_\theta$. Now $z(\theta) = \mathbb{E}_{\mathcal{Z}}[Z(\theta)]$.

To compute $z'(\theta)$, let $f(x;\theta)$ be the pdf of $Z(\theta)$ and $\dfrac{\partial f}{\partial \theta}$ is continuous. $Y(\theta) = Y[Z(\theta)]$.

$$\begin{aligned} z'(\theta) &= \frac{d}{d\theta}\int_{\mathcal{Z}} x f(x;\theta)dx = \int_{\mathcal{Z}} x\frac{f_\theta(x;\theta)}{f(x;\theta)}(x;\theta)dx \\ &= \mathbb{E}\left\{Z(\theta)\left[\ln f(Z(\theta);\theta)\right]'\right\} \overset{\triangle}{=\!=} \mathbb{E}\left\{Z(\theta)S[Z(\theta);\theta]\right\}. \end{aligned}$$

The LR estimator $Z \cdot S(Z;\theta)$ an also unbiased (for $z'(\theta)$) with standard deviation of $O(n^{-\frac{1}{2}})$, where $S(\theta_0)$ is called the **score function** evaluated at θ_0. The score function has additive property in high dimensional case.

For composed random variable $Y = Y[Z(\theta)]$, $\dfrac{\partial Y}{\partial \theta}$ can be computer using LR as $Y(Z) \cdot S(Z;\theta)$, but it may be biased.

Method 18. Likelihood Ratio Estimation

1. Choose a sample size n.
2. Generate n i.i.d. samples of $Z_i(\theta)$, $i = 1, \cdots, n$.
3. Output (estimator for $z'(\theta)$):

$$\widehat{g}_n^{\mathrm{LR}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} Z_i(\theta)S(Z_i;\theta).$$

Remark.

- LR and IPA are both model dependent.
- If both LR and IPA are nice, we usually prefer IPA as it often has a smaller variance.
- Similar to IPA, LR method is also widely applied in data-based algorithms. For instance, in reinforcement learning, the famous policy-gradient method is derived from LR gradient estimate.

The next are several examples relevant to the gradient estimation.

Example 15. G/G/1 Queues

Recall the G/G/1 queue example in Chapter 1. Let $\{U_n\}$ denote i.i.d. samples for inter-arrival time, and $\{V_n\}$ for i.i.d. job sizes. If the service rate is μ, the service time of a customer with job size V is $\frac{V}{\mu}$.

(1) Suppose a cost of $c(\mu)$ per unit time is paid for servers, and a cost of h_0 per customer in the queue per unit time. Formulate a model to decide how much capacity μ should be invested.

(2) Consider the waiting time W of a customer. Let W_0 and V be the waiting time and job size of the previous customer. Let U be the inter-arrival time of the two customers. Derive a pathwise estimate for $\dfrac{d}{d\mu}\mathbb{E}(W)$.

The minimization question for (1) is

$$\min_{\mu>0} c(\mu) + h_0\mathbb{E}_\mu[Q_\infty], \text{ where } Q_\infty \text{ is the long run average size of the queue.}$$

For the second question, the pathwise estimation is

$$\frac{1}{N}\sum_{i=1}^{N} -\frac{V_i}{\mu^2}\mathbb{1}_{\{W_0+V_i/\mu>U_i\}}$$

Example 16. Black Scholes Delta

Consider the case of a European call option with strike K and maturity T in the Black-Scholes framework. The option payoff is

$$Y = e^{-rT}(S_T - K)^+; \quad S_T = S_0 \exp\left\{(r - \frac{\sigma^2}{2})T + \sigma\sqrt{T}Z\right\}, \text{ where } Z \sim N(0,1).$$

Derive estimations of $\dfrac{\partial}{\partial S_0}\mathbb{E}[Y]$.

IPA Estimation:

$$\frac{1}{M}\sum_{i=1}^{M} \mathbb{1}_{\{S_{T,i}>K\}} \exp\left\{-\frac{\sigma^2}{2}T + \sigma\sqrt{T}Z_i\right\}.$$

LR Estimation:

$$\frac{1}{M}\sum_{i=1}^{M} Y_i \frac{Z_i}{\sigma\sqrt{T}S_0}.$$

5.3 Stochastic Gradient Descent

5.3.1 Simulation Approximation

Now turn to the optimization question:

$$\begin{array}{ll} \text{minimize}_\theta & z(\theta) \\ \text{subject to} & \theta \in \Theta \end{array}$$

Under some further pre-assumptions:

- $z(\cdot)$ is smooth over a region $\Theta \subset \mathbb{R}$.
- There is a unique $\theta^\star \in \Theta$ such that $z(\theta^\star) = 0$.

Robbins and Monro (1951) proposed a iterative simulation-based algorithm that converges to $\theta^\star$ in probability.

Method 19. SA Iteration (Robbins and Monro)

Every step let

$$\theta_{n+1} = \text{Proj}_\Theta(\theta_n - \varepsilon_n Z_n),$$

where

- $\{Z_n\}$ is a collection of random variables;
- The first conditional moment $\mathbb{E}(Z_n|\theta_n) = z(\theta_n)$;
- The second conditional moment $\mathbb{E}(Z_n^2|\theta_n)$ is bounded, for all possible $\theta_n \in \Theta$
- $\{\varepsilon_n\}_{n=1}^\infty$ is diminishing, i.e., $\sum_n \varepsilon_n$ diverges but $\sum_n \varepsilon_n^2$ converges.

5.3.2 Stochastic Gradient Descent and Variant Methods

SGD can be viewed as a special case of SA(Robbin-Monro). Since $g(\theta^\star) = f'(\theta^\star) = 0$, with critical point $\theta^\star$.

Theorem 4. L_2 **Convergence under Strong Convexity**

Assume $f(\cdot)$ to be strong convex with a constant $K > 0$. Then

$$\mathbb{E}(\theta_n - \theta^\star)^2 \longrightarrow 0$$

Proof. $\mathbb{E}(\theta_{n+1} - \theta^\star)^2 \leq \mathbb{E}(\theta_n - \varepsilon_n G_n - \theta^\star)^2 \leq (1 - \varepsilon_n K)\mathbb{E}(\theta_n - \theta^\star)^2 + \varepsilon_n^2 M.$ ■

There are also some variant methods, like **Polyak averaging**: use larger step-size $\varepsilon_n = O(n^{-\gamma})$ where $\gamma < 1$, but approximate

$$\theta^\star = \frac{1}{n}\sum_{i=1}^n \theta_i.$$

5.3.3 SGD with Baised Gradient Estimator

If the gradient is estimated using FD, the estimator is biased. SGD with FD gradient estimator is called **Kiefer-Wolfwitz (KW) Algorithm**.

Method 20. Kiefer-Wolfwitz Iteration

Under the same assumption as above, but do update following

$$\theta_{n+1} = \text{Proj}_\Theta(\theta_n - \varepsilon_n G_n), \text{ and } G_n = \frac{Z(\theta_n + \delta_n/2) - Z(\theta_n - \delta_n/2)}{\delta_n}.$$

In addition to ε_n, we have an extra hyper-parameter δ_n.

Remark.

- It is not surprising that KW converges slower than SA with unbiased gradient estimator.
- When $z(\theta)$ is nice, and i.i.d. sampled for all Z the optimal choice of hyper-parameters is $\varepsilon_n = O(n^{-1})$, $\delta_n = O(n^{-1/6})$, and the fastest convergence rate is $\mathbb{E}[(\theta_n - \theta^\star)^2] = O(n^{-2/3})$.
- If we use CRN to sample Zs, the optimal choice of hyper-parameters is changed to $\varepsilon_n = O(n^{-1})$ and $\delta_n = O(n^{-1/5})$. The fastest convergence rate now changes to $\mathbb{E}[(\theta_n - \theta^\star)^2] = O(n^{-4/5})$.
- Bias in gradient estimator is not always bad. For instance, with **Polyak momentum**,
$$\theta_{n+1} = \theta_n - \varepsilon_n G_n + \eta_n(\theta_n - \theta_{n-1})$$

5.4 Heuristic Methods in Discrete Optimization via Simulations

5.4.1 Challenges in Discrete Optimizations

There are some challenges to solve discrete optimization via simulations:

- Feasible solution sets are too large.
- Hard to solve a large-scale mixed-integer programming (MIP).
- Not have an "LP relaxation equivalent" that is easy to calculate and generates bounds.
- Solve general form MIP is NP-hard.

Heuristics come to rescue! How to use heuristic methods often depends on the problem structure:

- Traveling salesperson problem (TSP): nearest neighbors
- Knapsack: greedy algorithm to select items with the highest value-to-weight ratio
- Job shop scheduling: set a priority rule for jobs (e.g., earliest due date)

5.4.2 (Meta)Heuristic

For discrete optimization via simulation, we use a set of general problem-solving strategy named **metaheuristics**. Metaheuristics work across a wide range of problem domains.

Random Search

Random search is a method of searching the best direction along the feasible set. Suppose now we focus on a one-dimensional decision variable which is an integer. We can transform all feasible solutions into the one-dimensional space $\theta \in \mathbb{N} = \{1, 2, \cdots\}$.

Method 21. One Dimensian Random Search

1. Select a starting point k_0. Let the iteration number $m = 0$, the list recording the number of visits $N_0(k_0) = 1$ and $N_0(k) = 0$, $\forall\ k \in \mathbb{N}\backslash\{k_0\}$.
2. Generate a corresponding Bernoulli R.V. to determine whether we want to check the up or the down direction.
3. If we decide to check the up direction, sample Y_{k_m} and Y_{km+1}. If $Y_{k_m} > Y_{k_{m+1}}$, then we move up by letting $k_{m+1} = k_m + 1$; else we stay put by letting $k_{m+1} = k_m$. Same principle applies if we decide to check the down direction. Record $N_{m+1}(k_{m+1}) = N_m(k_{m+1}) + 1$.
4. Repeat Step 2 and 3 until we run out of the simulation budget B. Output $k^\star = \arg\max_{k\in\mathbb{N}} N_B(k)$.

Remark. Random search algorithm has a global optimality guarantee if $B \to \infty$. However, it is not always easy to obtain such a Y , especially when we transform the multi-dimensional decision space into a one-dimensional space.

Tabu Search

Tabu (i.e., "taboo", forbidden) search is another method to create a **hill-climbing opportunity**. Tabu search excludes such "comeback" actions to go back to some local mins. It records a memory of moves and historical visits.

The tabu list (tenure) is constructed specific to the problem structure and sometimes it might be hard to design such a list to achieve efficiency.

Method 22. General Tabu Search

1. Choose an initial solution x_0. Set the current solution $x = x_0$ and best solution $x_{\text{best}} = x_0$. Start with empty tabu lists.
2. Generate the neighbors of x, $x' \in N(x)$ and call the simulation model to estimate the cost function $f(x')$.
3. Select the best neighbor x'_{best} and update the x_{best} if necessary.

4. If the move to x'_{best} is tabu and $f(x'_{\text{best}}) > f(x_{\text{best}})$, then set $x'_{\text{best}} = \infty$ and return to step 3; else update the current solution $x = x'_{\text{best}}$
5. Update the tabu list.

Simulated Annealing

Details in "MAT-3300 Modeling" notes.