A report on

Statistical Simulation and Data Analysis

Submitted By

Nandita Suresh Kamath 2018B4A20868P

Submitted in partial fulfillment of the course

MATH F432 -Applied Statistical Methods

Under the Guidance of

Prof. Sumanta Pasari

Birla Institute of Technology and Science, Pilani

Statistical Simulation and Data Analysis

Contents

1 Introduction

In statistics, simulation is used to assess the performance of a method, typically when there is a lack of theoretical background. With simulations, the statistician *knows and controls the truth.* Simulation is used advantageously in a number of situations. This includes providing the empirical estimation of sampling distributions, studying the misspecification of assumptions in statistical procedures, determining the power in hypothesis tests, etc.

For instance, consider the *overcrowding problem* in Emergency Departments(EDs) in hospitals. So, to understand it better we need to take in account the no. of patients arriving per day, no. of doctors, availability of hospital beds, etc. Since ED is a complex system, we cant use purely mathematical models such as linear programming, nonlinear programming or others to solve this problem. Here, we can take advantage of the power of simulation analysis to develop efficient models to increase patient satisfaction.

The *building block* of a simulation study is the ability to generate random numbers. By constructing a probabilistic model for the problem, we can use random numbers to generate different values of random variables involved in the problem. This way we can run the probabilistic model several times and generate several outcomes which act as data points (samples) that can be analysed using various statistical tools.

Thus, rather than solving a problem analytically, we can use simulation to create enough sample points to estimate required parameters to an arbitrary accuracy.

2 Inverse Transform Method

2.1 Discrete and Continuous Random Number Generators Using MATLAB

MATLAB codes used to convert uniform random variables to any other desired random variable:

Code	Output
U = rand	Returns a pseudorandom value taken from the standard uniform distribution on (0,1)
U = rand(m,n)	Returns an m-by-n matrix containing independent pseudorandom values taken from the standard uniform distribution on (0,1)

2.2 Generating Continuous Probability Distributions from the Uniform Distribution

Inverse Transform Sampling is a statistical technique that generates non-uniform random variables with a desired distribution. This technique is useful for random sampling.

Theorem 1 *Let $U \sim$ Uniform (0, 1) and F be a cumulative distribution function that is strictly increasing. Let X be a random variable defined as $X=F^{-1}(U)$ Then, the CDF of X is F. That is, $X \sim F$*

Proof 1 $P(X \leq x)=P(F^{-1}(U) \leq x)=P(U \leq F(x))=F(x)$

To generate any continuous random variable X with the continuous CDF, F and $F^{-1}(U)$ is calculated.

Example: Generating an exponential random variable to create a random variable X that is distributed exponential with parameter λ using the PDF distribution:

$$F(x) = \begin{cases} e^{-x}, & \text{for } x \geq 0 \\ 0, & \text{for } x < 0 \end{cases}$$

Step 1: Find the CDF

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

Therefore, CDF =

$$F(x) = \begin{cases} 1 - e^{-x}, & \text{for } x \geq 0 \\ 0, & \text{for } x < 0 \end{cases}$$

Step 2: Find the inverse of CDF

$$F(x) = y = 1 - e^{-x}$$

$$x = \frac{-ln(1-y)}{\lambda} = F^{-1}(y)$$

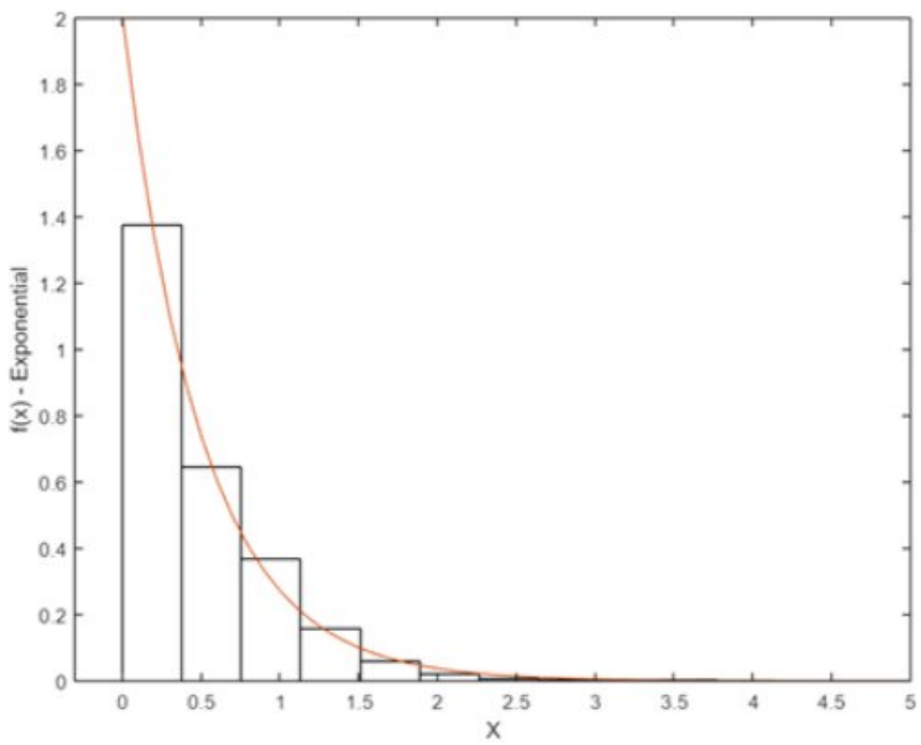
Step 3: Substitute random variable U into the function

$$x = F^{-1}(U) = \frac{-ln(1-U)}{\lambda}, U \sim Uniform(0,1)$$

Code used to generate sample from the distribution when λ=2:
The formula is simplified since 1-U $\sim$ Uniform(0,1) Therefore, $x = F^{-1}(U) = \frac{-ln(1-U)}{\lambda} = \frac{-lnU}{2}$

The following MATLAB code will generate 1000 exponential random variables for λ =2 and plot them on a histogram:

```
% Set up the parameters
lam = 2;
n = 1000;
% Generate the random variables
uni = rand(1,n);
X = -log(uni)/lam;
% Get the values to draw the theoretical curve
x = 0:.1:5;
% This is a function in the Statistics Toolbox
y = exppdf(x,1/2);
% Get the information for the histogram
[N,h] = hist(x,10);
% Change the bar heights to make it correspond
% to the theoretical density
N = N/(h(2)-h(1))/n;
% Plot
bar(h,N,1,'w')
hold on
plot(x,y)
hold off
xlabel('X')
ylabel('f(x) - Exponential')
```



3 Markov Chain Monte Carlo Method

3.1 Terminology

3.1.1 Markov Chains

Markov Chains consists of a set of possibly attainable states known as a *state space* and their defining property is that the probability of moving to a state is dependent entirely on the previous state i.e. it has the so-called **memory-less** property. Mathematically,

$$P(X_n = i_n | X_{n-1} = i_{n-1}) = P(X_n = i_n | X_0 = i_0, X_1 = i_1, ..., X_{n-1} = i_{n-1})$$

Certain Markov Chains converge to a *stationary distribution* for large time steps.

3.1.2 Monte Carlo Methods

Monte Carlo methods refer to a mathod that makes use of a combination of numerical simulation and random number generation to obtain the desired outcome. It utilizes the *principle of large numbers* , i.e. repeating an experiment/simulation a large number of times results in the simulation returning the required result even though the sampling is *random*.

If it is known that an event will occur with some probability in certain conditions, MC simulation makes use of high computational power of modern devices in order to generate those conditions repeatedly.

Therefore, the idea is to use a random simulation to estimate a quantity of interest by generating many random variables according to a desired distribution.

(A process is called **stochastic** if its next step depends on both previous states and some random event.)

3.1.3 MCMC

It is used for *simulating sampling* from complex probability distributions. We simulate a markov chain such that the **target distribution** that we sample from *is* the **stationary distribution** of the chain.

3.1.4 SIR Model

The SIR Model used to simulate *disease outbreaks* consists of three states, *Susceptible*, *Infected* and *Removed*. *S(t), I(t), R(t)* represent the number of susceptible, infected and removed individuals respectively at time *'t'*.

A susceptible individual can get infected by coming in contact with an infected individual. An infected individual may either die or recover from the disease. We would consider a dead and recovered person as

the equal i.e. he would become a removed individual. The rate of infection is denoted by a and b is the rate of removal.

The model is governed by the equations:

- $\frac{dS}{dt} = -a(\frac{SI}{N})$
- $\frac{dI}{dt} = a(\frac{SI}{N}) - b(I)$
- $\frac{dR}{dt} = b(I)$

Since the total population is assumed to be constant N, we can reduce the model to only two variables $S(t)$ and $I(t)$ by computing $R(t)$ as N-$S(t)$-$I(t)$.

Now $S(t)$, $I(t)$ can take values from {0,1,2,3,. . . ,N} and t from $[0,\infty)$. Since there are only two random variables, we can call {S(t), I(t)} a bivariate process, and we denote the joint probability function by $P_{(s,i)}(t) = P[S(t) = s, I(t) = i]$.

The transition probability from state (s,i) to (s+k, i+j) in time δ t can be defined as

$$P_{(s,i),(s+k,i+j)}(\delta t) = P[S(t+\delta t) = s+k, I(t+\delta t) = i+j | S(t) = s, I(t) = i]$$

Here, the transition probability does not depend upon the specific t, rather it depends on δ t thereby following the Markovian property.

So the transition probabilities for different values of k and j can be defined as:

$$P = \begin{cases} a(\frac{IS}{N})\delta t + o(\delta t), & \text{for } (k,j) = (-1,1) \\ b(I)\delta t + o(\delta t), & \text{for } (k,j) = (0,-1) \\ 1 - (a(\frac{IS}{N}) + b(I))\delta t + o(\delta t), & \text{for } (k,j) = (0,0) \\ o(\delta t), & \text{otherwise} \end{cases}$$

This forms the CTMC SIR model.

3.1.5 Gillespie Algorithm

Since solving the transition equations for stable conditions is quite difficult, we numerically simulate the stochastic realizations of the process via the help of Gillespie algorithm. It generates possible solutions of stochastic equation systems which have known reaction rates. It is a variant of a *dynamic MC method* i.e. it is used to study non-equilibrium systems.

For our model, we take two random numbers u_1,$u_2 \in$ U[0,1] where u_1 is used to simulate the intervent time and u_2 is used to simulate the occurence of a particular event.

Now, the markov property implies that the intervent time T has an exponential distribution i.e. T ~ λ $e^{-\lambda}$ $twhere\lambda$, the sum of rates of all possible events is $a(\frac{SI}{N}) + b(I)$

So t can be estimated by using the cumulative distribution function as,

$$u_1 = P[T > t] = 1 - F(t) = e^{-\lambda t} \implies t = -\frac{ln(u_1)}{\lambda}$$

And by using the value of λ, we can divide the two events of infection and removal into separate probabilities, $p_1 = \frac{a(SI)}{N}/\lambda$ and $p_2 = (bI)/\lambda$. Now the interval [0,1] can be split into [0,p_1] and (p_1, 1] and if u_2 belongs in the first interval then (s,i) $\rightarrow$ (s-1, i+1) otherwise (s,i) $\rightarrow$ (s,i-1)

This is how the Gillespie Algorithm can be used to simulate the SIR model.

3.2 Implementation

We used data from Maharashtra from the period of 1 February, 2021 to 1 April from this dataset from Kaggle for the SIR model.

3.2.1 Inputs

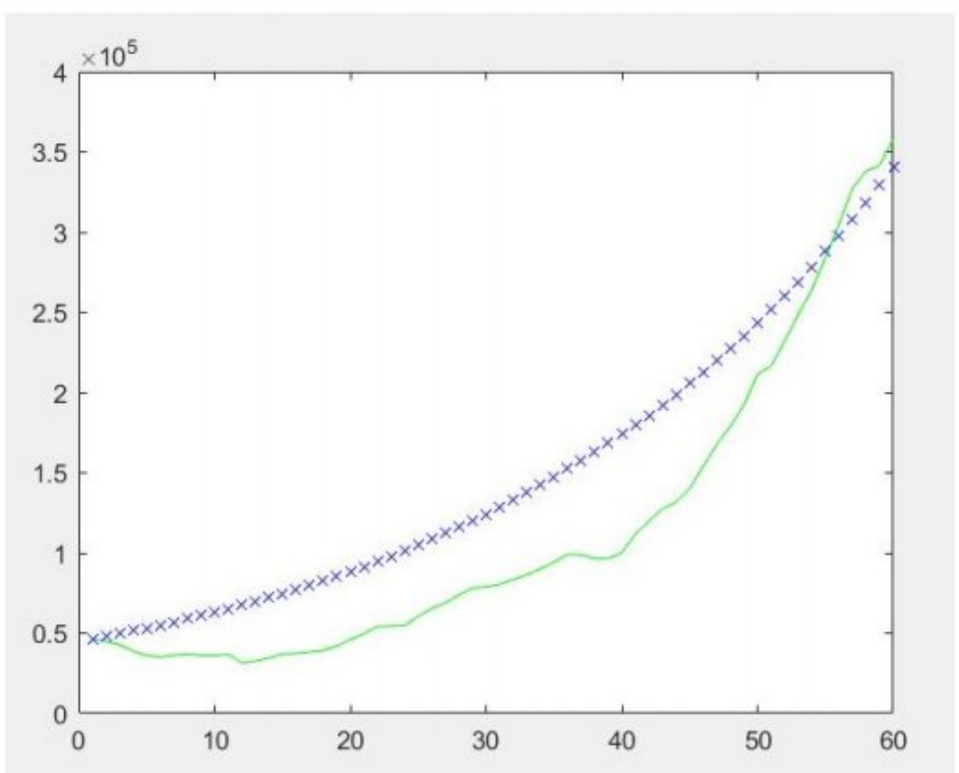
a	0.0830
b	0.0490
i	46,312
I_{sim}	46,312
N	123,144,223
S_{sim}	123,097,911
t	0
T	60

3.2.2 Functions

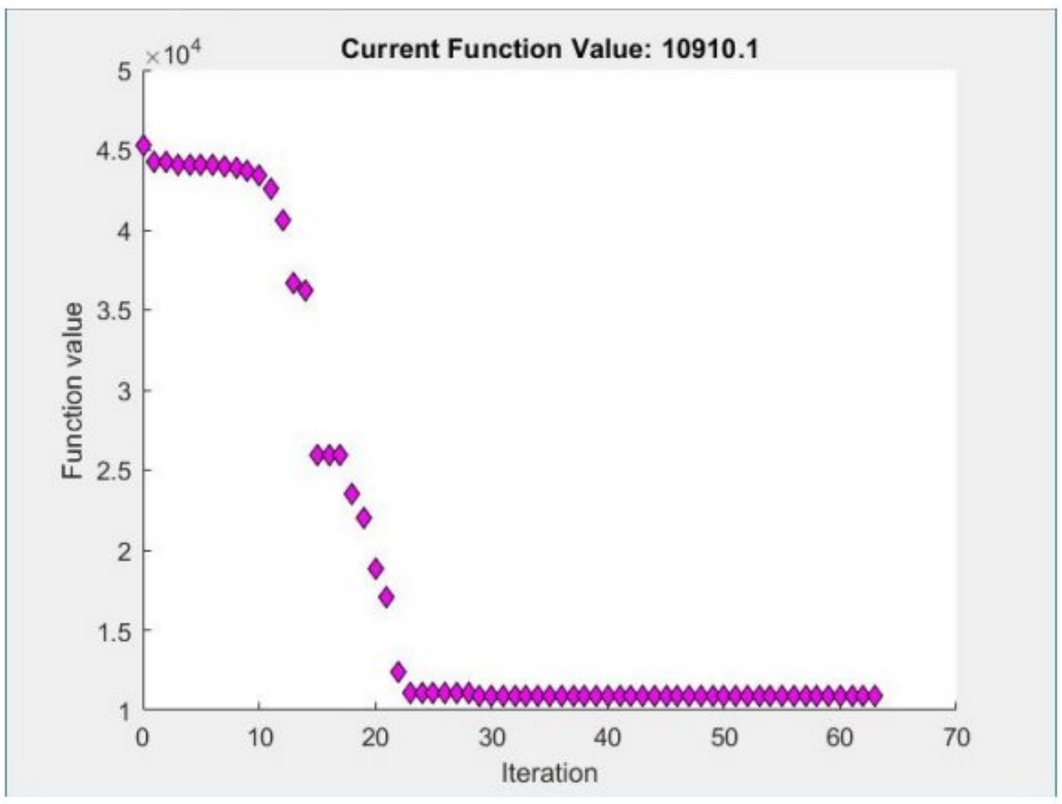
- `main.m`: Main function that calls $get_{data.m}$ to get the necessary data and performs the simulation by calling `simulation.m`.
- `get_data.m`: Takes in the data for the *S,I and R* values from the CSV file for Maharashtra for the specified duration of two months and computes the value of the population N.
- `ode_func.m`: Creates the differential equations for provided values of *a, b and T*.
- `estimate.m`: It uses the `ode45` MATLAB function to solve the differential equations that are fed in by `ode_func` and estimates the optimal values of a and b that are fed to `get_parameters.m`. It also computes the loss function f which has the difference between the actual S and I values from the dataset and the simulated S and I values that we get from our estimated *a and b*.
- `get_parameters.m`: Uses `fminsearch` and calls `estimate.m` which has an ODE solver function that acts on the DE provided and the estimated values of S and I and the loss function is computed to the tolerated value and it returns optimized values of a and b.
- `simulate.m`: Uses the *a and b* and *initial infected* values and at each timestep using Gillespie algorithm simulates the values for S and I and the final outcome after each day is recorded.

3.2.3 Results

The *x-axis* denotes the time passed in days and the *y-axis* shows the number of infected people, *I(t)*. The crossed output denotes the data from the dataset whereas the smooth curve shows the simulated I. We observe that the simulated curve is very smooth because of Gillespie algorithm.



Since the loss function reduces significantly after some iterations, we can see that `fminsearch` has worked as expected and our estimated values are inching closer to the actual values.



4 Simulation Analysis

As part of setting up the simulation experiment, one must decide what type of simulation to run. Simulations are usually distinguished as being one of two types: *terminating* or *non-terminating*. The difference between the two has to do with whether we are interested in the behavior of the system over a particular period of time or in the steady-state behavior of the system. It has nothing to do, necessarily, with whether the system itself terminates or is ongoing. The decision to perform a terminating or non-terminating simulation has less to do with the nature of the system than it does with the behavior of interest.

4.1 Simulation Analysis for Terminating Simulations

In terminating simulation, Parameters to be estimated are defined relative to specific initial and stopping conditions that are part of the model. Output performance measures generally depend on both the initial and stopping conditions.

Suppose that we make n independent replications of a terminating simulation each terminated by the event E. Here we make sure that for each replication, we use the same initial conditions, same terminating event, and separate random numbers.

Let X_j be an output random variable defined over the j^{th} replication, for j = 1,2,. . . ,n ; it is assumed that the X_j s are comparable for different replications. Then the X_j s are IID random variables.

Now if we want to obtain a point estimate and confidence interval for the mean μ = E (X) , where X is a random variable defined on a replication as described above. Then we get that X(n) is an unbiased point estimator for , and an approximate 100(1 - α) percent confidence interval for is given by

$$X(n) \pm t_{n-1,1-\alpha/2}\sqrt{S^2(n)/n}$$

Here, $t_{n-1,1-\alpha/2}\sqrt{S^2(n)/n}$ is the *half-width.*

4.2 Simulation Analysis for Non-Terminating Simulations

A non-terminal/steady-state simulation is a simulation where the simulation cannot be stopped by the occurrence of some event. Theoretically, the simulation can go on forever with no change in statistical behaviour.

Let $Y_1, Y_2, ..., Y_n$ be the output of a stochastic process of a non-terminating simulation. The mean μ of the steady state needs to be estimated for simulation analysis and is defined by

$$\mu = \lim_{i \to \infty} E(Y_i)$$

However, for small values of i, $E(Y_i) \neq \mu$ since the initial random variables will not be representative of the steady state mean. This would result in $\overline{Y}$ being a biased estimator. This is known as the problem of initial transient. To tackle this, generally the initial observations are not used and only the observations after a certain number, say l are used.

4.2.1 The Replication/Deletion Approach

Due to the issue mentioned previously, the initial l observations are not used for estimation. The value of l can either be decided by using judgement or Welchs procedure. Make n simulations of length m each, $m >> l$ of the experiment. Y_{ki} is the i^{th} random variable from the k^{th} simulation where i=1,2,. . . ,m and k=1,2,. . . ,n. Define

$$X_k = \sum_{i=l+1}^{m} \frac{Y_{ki}}{m-l}, k = 1, 2, ..., n$$

Since i=l+1, X_k's are IID with $E(X_k) \approx \mu$

Hence, now we can get $\overline{X}(n) = \sum_{k=1}^{n} \frac{X_k}{n}$ as the point estimate and

$$\overline{X}(n) \pm t_{n-1,1-\frac{\alpha}{2}} \sqrt{\frac{S^2(n)}{n}}$$

as the CI for confidence level α.

5 Variance Reduction Techniques

We have seen that the number of draws that we have to generate in a simulator is related to the precision we would like to reach. And variance reduction methods will be such that we will be able to reach the same precision with a lower number of draws. Or, equivalently, we will reach a higher precision with the same number of draws.

There are two techniques that are most prominent.

- Antithetic Draws
- Control Variates

5.1 Antithetic Draws

Let us consider the mean of two identically distributed random variables U_1 and U_2. If U_1 and U_2 are independent, then $Var(\frac{U_1+U_2}{2}) = \frac{Var(U_1)+Var(U_2)}{4}$, In general,

$$Var(\frac{U_1+U_2}{2}) = \frac{Var(U_1)+Var(U_2)+2Cov(U_1,U_2)}{4}$$

Thus the variance of $\frac{U_1+U_2}{2}$ is smaller if U_1 and U_2 are negatively correlated. This fact leads us to consider negatively correlated variables as a possible method for reducing variance.

Suppose that $X_1, ..., X_n$ are simulated via the inverse transform method. For each m, generate $U_j \sim U(0,1)$ and compute $X_j = F_X^{-1}(U_j)$. Note that since both U and 1-U $\sim$ U(0,1), $Y_j = g(F_X^{-1}(1-U_1^{(j)}, ...)$ has the same distribution as $Y_j' = g(F_X^{-1}(U_1^{(j)}), ...)$

For Y_j and Y'_j to be negative correlated, g must be an *n-variate monotone function* i.e. $g(x_1, ..., x_n) \leq (\geq) g(y_1, ..., y_n)$ when $(x_1, ..., x_n) \leq (\geq)(y_1, ...y_n)$.

Proposition 1 *If $(X_1,. . . ,X_n)$ are independent, and f and g are increasing functions, then* ***E[f(X)g(X)] $\geq$ E[f(X)]E[g(X)]***

Proof 2 *Assume that f and g are increasing functions. The proof is by induction on n.*
Suppose n = 1. Then, (f(x) f(y))(g(x) g(y)) ≥ 0 for all x,y ∈ R.
Hence

$$E[(f(X)f(Y))(g(X)g(Y))] \geq 0$$
$$E[f(X)g(X)] + E[f(Y)g(Y)] \geq E[f(X)g(Y)] + E[f(Y)g(X)].$$

Here X and Y are iid, thus

$$\begin{aligned} 2E[f(X)g(X)] &= E[f(X)g(X)] + E[f(Y)g(Y)] \\ &\geq E[f(X)g(Y)] + E[f(Y)g(X)] \\ &= 2E[f(X)]E[g(X)]. \end{aligned}$$

The statement is true for n=1. Assume that it is true for $X \in R^{n-1}$. Applying the induction hypothesis,

$$\begin{aligned} E[f(X)g(X)][X_n = x_n] &\geq E[f(X_1, ..., X_{n-1}, x_n)]E[g(X_1, ..., X_{n-1}, x_n] \\ &\geq E[f((X)|X_n = x_n)]E[g((X)|X_n = x_n)] \end{aligned}$$

Which can be re-written as $E[f(X)g(X)|X_n] \geq E[f(X)|X_n]E[g(X)|X_n]$.
Now E[f(X) — X_n] and E[g(X) — X_n] are each increasing functions of X_n. So applying the result for n = 1 and taking the expected values of both sides,

$$\begin{aligned} E[f(X)g(X)|X_n] &\geq E[E[f(X)|X_n]E[g(X)|X_n] \\ &\geq E[f(X)]E[g(X)]. \end{aligned}$$

Corollary 1 *If $g=g(X_1,..,X_n)$ is monotone, then $Y=g(F^{-1}{}_X(U_1),\ldots,F^{-1}{}_X(U_n))$ and $Y'=g(F^{-1}{}_X(1\text{-}U_1),\ldots,F^{-1}{}_X(1\text{-}U_n))$ are negatively correlated.*

The antithetic variable approach is easy to apply. If m MC replicates are required, generate $\frac{m}{2}$ replicates $Y_j = g(F_X^{-1}(U_1^{(j)}, ..., F_X^{-1}(U_n^{(j)}))$ and the remaining $\frac{m}{2}$ replicates $Y_j' = g(F_X^{-1}(1 - U_1^{(j)}), ..., F_X^{-1}(1 - U_n^{(j)})$ where $U_i^{(j)}$ are *iid* U(0,1) variables and *i=1,...n* and *j=1,...,m/2*. Then the arithmetic estimator is:

$$\begin{aligned} \hat{\theta} &= \frac{1}{m}[Y_1 + Y_1' + Y_2 + Y_2' + ... + Y_{m/2} + Y_{m/2}'] \\ &= \frac{2}{m}\sum_{j=1}^{m/2}(\frac{Y_j + Y_j'}{2}) \end{aligned}$$

Thus, only $\frac{nm}{2}$ uniform variates are needed instead of nm, and the required variance of the MC estimator is reduced by using antithetic variables.

5.2 Control Variates

We are trying to calculate the empirical mean of the distribution from which we are sampling. We use the simulation to estimate $\Theta = \mathrm{E}[\mathrm{X}]$, where X is an output of the simulation. Let Y be another output of the simulation which is similar to X, such that we know $\mathrm{E}[\mathrm{Y}] = \mu$. We will consider another random variable which will also be an unbiased estimator of X but its variance would be less. Let that variable be Z.

$$Z = X + c(Y - \mu)$$

Now as E[Y]=μ, also $E[Z] = E[X]$ Variance of Z can be calculated as:

$$Var(Z) = Var(X + cY) = VarX + c^2Var(Y) + 2cCov(X, Y)$$

Now we will find c such that Var(Z) comes out to be minimum. First derivative of Var(Z) with respect to c is $2cVar(Y) + 2Cov(X,Y)$ And, at a minimum, the derivative must be zero. We denote the value at which this derivative is zero by c*:

$$c^* = \frac{-Cov(X,Y)}{Var(Y)}$$

To verify that it is indeed a minimum, we calculate the second derivative $2Var(Y)$ which is positive. Hence c^* minimizes Var(Z) and we define Z^* corresponding to c*:

$$Z^* = X - \frac{Cov(X,Y)}{Var(Y)}(Y - \mu)$$

Variance of Z* becomes:

$$Var(Z^*) = Var(X) - \frac{Cov(X,Y)^2}{Var(Y)} \leq Var(X)$$

In the worst case, Var(Z*) is equal to Var(X) when X and Y are independent. But, of course, we find a Y which is correlated with X, in order to reduce the variance. Hence, control variates guarantee that the variance of the new random variable will always be less or equal to the previous one.

In practice, Cov(X,Y) and Var(Y) are not usually known. We can use their sample estimates.

$$\widehat{Cov}(X,Y) = \frac{1}{n-1}\sum_{r=1}^{R}(X_r - \overline{X})(Y_r - \overline{Y})$$

$$\widehat{Var}(Y) = \frac{1}{n-1}\sum_{r=1}^{R}(Y_r - \overline{Y})^2$$

Still, these quantites are quite cumbersome to calculate, there is actually a better way to compute them. We regress X on Y.

$$X = aY + b + \epsilon$$

where a is the coefficient of Y in the linear regression, and b is the intecept. And the error term, as always in linear regression, is a normal distribution with mean 0 and variance σ^2. The least square estimators of a and b are:

$$\hat{a} = \frac{\widehat{Cov}(X,Y)}{\widehat{Var}(Y)} = \frac{\sum_{r=1}^{R}(X_r - \overline{X})(Y_r - \overline{Y})}{\sum_{r=1}^{R}(Y_r - \overline{Y})^2}$$

$$\hat{b} = \overline{X} - \hat{a}\overline{Y}$$

And we can clearly observe that $c^* = -\hat{a}$

So, there is a relationship between the results of this linear regression, and the variance reduction method that we have seen. Moreover, if we calculate the prediction obtained by the linear model at the value μ of Y:

$$\begin{aligned}\hat{b} + \hat{a}\mu &= \overline{X} - \hat{a}\overline{Y} + \hat{a}\mu \\ &= \overline{X} - \hat{a}(\overline{Y} - \mu) \\ &= \overline{X} + c^*(\overline{Y} - \mu) \\ &= \hat{\theta}\end{aligned}$$

This is exactly the control variate estimate of Θ.

In summary, $\widehat{\Theta}$ can be obtained using the linear model and evaluating X at μ. Therefore, the procedure is really simple: we run the simulator, and collect the draws from X and Y. Then, regress X on Y and obtain $\hat{a}$ and $\hat{b}$ which are the coefficient and the intercept of the linear model. And then, we calculate the quantity above to obtain $\widehat{\Theta}$. Hence this is a way to exploit the correlation between two quantities in the simulator. One that we don't know, and one that we know.

6 Sources and References

1. Statistical Simulation - Stackexchange
2. Simulation, Fifth Edition by Sheldon M. Ross
3. Weng et al, "Using Simulation and Data Envelopment Analysis in optimal healthcare efficiency allocations" Proceedings of the 2011 Winter Simulation Conference
4. M. Law, "Statistical analysis of simulation output data: The practical state of the art," 2015 Winter Simulation Conference (WSC), 2015, pp. 1810-1819, doi: 10.1109/WSC.2015.7408297
5. Alexopoulos and A. F. Seila, "Output data analysis for simulations," Proceeding of the 2001 Winter Simulation Conference (Cat. No.01CH37304), 2001, pp. 115-122 vol.1, doi: 10.1109/WSC.2001.977252.
6. Martin Haugh(2017), Simulation Efficiency and an Introduction to Variance Reduction Methods, Columbia University
7. The SIR Model for Spread of Disease - MAA
8. MCMC, Monte Carlo Methods, Markov Chains - Wikipedia
9. MCMC Sampling for Dummies
10. Markov Chains - Brilliant, Stationary Distributions -Brilliant

7 Appendix

7.1 main.m

```
get_data;
[f, fminval] = get_parameters();
simulation;
```

7.2 get_data.m

```
function [] = get_data()

    global S I R
    global T N i
    opts = detectImportOptions('maharashtra_SIR.csv');
    preview('maharashtra_SIR.csv',opts)
    opts.SelectedVariableNames = {'Susceptible','Infected','Removed'};
    SIR = readmatrix('maharashtra_SIR.csv',opts);

     S = SIR(330:389,1);
     I = SIR(330:389,2);
     R = SIR(330:389,3);

     T = 389 - 330 + 1;

     N = S(1) + I(1) + R(1);

     i = I(1);

end
```

7.3 get_parameters.m

```
function [f, fminval] = get_parameters()

    global S I R T
    global a b

    params0 = rand(2,1);

    max_iters = 200;

    options = optimset('Display','off','MaxIter',max_iters, 'MaxFunEvals',max_iters,'PlotFcns','op

    [params, fminval,f] = fminsearch(@estimate, params0, options);
    warning('on')

    a = params(1);
    b = params(2);
end
```

7.4 estimate.m

```
function f = estimate(params)

    global S I T

    try
            warning('off')
        [tsol,sol] = ode45(@(t,T) ode_func(t,T,params(1),params(2)),0:T-1, [S(1),I(1)]);
            warning('on')
        catch
            f=NaN;
            warning('on')
            return
    end

    f =  (norm((S(1:T) - sol(:,1))) + norm((I(1:T) - sol(:,2))))/T;

end
```

7.5 ode_func.m

```
function dy = ode_func(t,var,a,b)

global N

dy=zeros(2,1);

dy(1)=-(a/N)*var(1)*var(2);
dy(2)=(a/N)*var(1)*var(2)-b*var(2);

end
```

7.6 simulation.m

```
function [] = simulation()

    global N a b
    global i T I S

    t = zeros(1,1);
    S_sim = (N - i) * ones(1,1);
    I_sim = i * ones(1,1);

    while((t(1) < T) && (I_sim(1) ~= 0))
        u = rand(2, 1);
        l = (a * S_sim(1) * I_sim(1)) / N + b * I_sim(1);
        t = [ t(1)- log(u(1)) / l; t];
        if u(2) <= a * S_sim * I_sim(1) /(N * l)
            if S_sim(1) ~= 0
                S_sim = [S_sim(1) - 1; S_sim];
                I_sim = [I_sim(1) + 1; I_sim];
            else
                S_sim = [S_sim(1); S_sim];
                I_sim = [I_sim(1); I_sim];
            end
        else
            S_sim = [S_sim(1); S_sim];
            I_sim = [I_sim(1)-1; I_sim];
        end
    end
    t = [T; t];
    S_sim = [S_sim(1); S_sim];
    I_sim = [I_sim(1); I_sim];

    S_sim2 = zeros(T,1);
    I_sim2 = zeros(T,1);
    t1 = ceil(t);
    for x=1:T
        S_sim2(x) = S_sim(find(t1==x,1,'last'));
        I_sim2(x) = I_sim(find(t1==x,1,'last'));
    end
    %S_sim2 = S_sim2.';

    figure();
    plot(1:T,I,'-g');
    hold on;
    plot(1:T,I_sim2,'xb','DisplayName','Gillespie Algorithm vs Actual Infectives');

end
```