Stochastic Simulation Coursework

Code

Xinan Wang CID:01410777 (CID:01410777)

10/12/2019

Objective

In this coursework, I focused on simulating random samples from some probability density function $f_X(\cdot)$ where

$$f_X(x) \propto \frac{1}{(x-a)(b-x)} exp\{-\frac{1}{c}(d+log(\frac{x-a}{b-x}))^2\}, \text{ for } a<x<b$$

Based on my CID $1410777 = 17417 \cdot 3^4$, my corresponding parameters are $a=1,\ b=2,\ c=7 \text{ and } d=0$. I then test my scheme using diagnostic plots as well as statistical tests, in order to make sure the samples indeed followed the desired distribution and are mutally independent. Finally, I implemented Monte Carlo method to estimate the normalising constant in the given function, and compared it with hit-or-miss MC method.

1. Generating from $f_X(\cdot)$ using Rejection

We only know $f_X^*(x) = \frac{1}{(x-1)(2-x)} exp\{-\frac{1}{7}(log(\frac{x-1}{2-x}))^2\}, \text{ for } 1<x<2$, and I choose a piecewise envelope defined as

$$g_Y(y) = \begin{cases} k(10-4y),\ 1<y\leq\frac{3}{2} \\ k(4y-2),\ \frac{3}{2}<y\leq 2 \\ 0,\ \text{otherwise} \end{cases}$$

with corresponding cumulative density functions

$$G(y) = \begin{cases} 0,\ y\leq 1 \\ k(-2y^2+10y-8),\ 1<y\leq\frac{3}{2} \\ k(2y^2-2y+c),\ \frac{3}{2}<y\leq 2 \\ k(4+c),\ y>2 \end{cases}$$

We can see that the scaling factor k and constant c satisfy $\text{when } y=\frac{3}{2}, \frac{5}{2}k=\frac{1}{2}$ and $k(\frac{3}{2}+c)=\frac{1}{2}$, and so $k=\frac{1}{5},\ c=1$. Then we have to work out the value for M where,

$$M = \sup_{x\in(1,2)} \frac{f_X^*(x)}{g_Y(x)}$$

Let $h(x) = \frac{f_X^*(x)}{g_Y(x)}$, then use `optimize` function to compute

Hide

```
fstar <- function(x) {ifelse(x==1,0,ifelse(x==2,0,1/((x-1)*(2-x))*exp(-(1/7)*(log((x-1)/(2-x)))^
2)))} # Define the proportional pdf
g <- function(x) {ifelse(x<1.5,(1/5)*(10-4*x),(1/5)*(4*x-2))} # Define my envelope function
h <- function(x) fstar(x)/g(x) # Division to compute M
optim=optimize(h, c(1,2), lower = 1, upper = 1.5,maximum = TRUE,tol=0.001) # Make use of optimis
e function
M=optim[[2]] # Set M to be the maximum
optim # Display the maximal point
```

Hence $h(x)$ is maximised when $x \approx 1.04$, and thus M takes value ≈ 5.27,

Hide

```
knitr::opts_chunk$set(warning=FALSE, message=FALSE)
suppressMessages(library(tidyverse,verbose=FALSE,warn.conflicts=FALSE,quietly=TRUE))
suppressMessages(library(forecast,verbose=FALSE,warn.conflicts =FALSE,quietly=TRUE))

x=seq(1,2,l=1000)
fgx = data.frame(x=x, fx=fstar(x), Mgx=g(x)*M) # Get function values at a range of x's
mylabs=list(expression(f[X]^"*"*(x)),expression(Mg[Y](x))) # Labels
```

Simulation from $g_Y(\cdot)$:

We have $Y \sim g_Y(\cdot)$ and the corresponding cumulative density functions, thus

$$G_Y(y) = \begin{cases} \frac{1}{5}(-2y^2+10y-8), & 1 < y \leq \frac{3}{2} \\ \frac{1}{5}(2y^2-2y+1), & \frac{3}{2} < y \leq 2 \end{cases} = u \Rightarrow G_Y^{-1}(u) = \begin{cases} \frac{2-\sqrt{4-\frac{8}{5}(\frac{8}{5}+u)}}{\frac{4}{5}}, & 0 \leq u \leq \frac{1}{2} \\ \frac{\frac{2}{5}+\sqrt{\frac{4}{25}-\frac{8}{5}(\frac{1}{5}-u)}}{\frac{4}{5}}, & \frac{1}{2} < u \leq 1 \end{cases}$$

This result enables me to simulate from g_Y by inversion.
Below is my algorithm to simulate from $f_X(\cdot)$.

Hide

```
library(ggplot2)
p <- ggplot(fgx) # Plot to compare f* and M*g
p + geom_line(aes(x,fx,colour="fx"))+
  geom_line(aes(x,Mgx,colour="fy"))+
  labs(y="pdf", title=expression("Comparison of "~f[X]^"*"* (x) ~"and "~Mg[X](x)))+
  scale_colour_manual("", values=c("fx"="red","fy"="blue"), labels=mylabs)
```

Rejection Algorithm:

1. Simulate $U_1 = u_1 \sim U(0,1)$.
2. Use rejection algorithm to generate from g_Y using inversion.
3. Set

$$y = G_Y^{-1}(u_1) = \begin{cases} \frac{2-\sqrt{4-\frac{8}{5}(\frac{8}{5}+u_1)}}{\frac{4}{5}}, & 0 \leq u_1 \leq \frac{1}{2} \\ \frac{\frac{2}{5}+\sqrt{\frac{4}{25}-\frac{8}{5}(\frac{1}{5}-u_1)}}{\frac{4}{5}}, & \frac{1}{2} < u_1 \leq 1 \end{cases}$$

.

4. Simulate $U_2 = u_2 \sim U(0,1)$
5. If $u_2 \leq \frac{f_x^*(y)}{Mg_Y(y)}$, set $X = y$, then $X \sim f_X(\cdot)$.
6. Otherwise GOTO 1.

Hide

```
utoy <- function(u){
  ifelse((u>0)&(u<=0.5),(2-sqrt(4-(8/5)*(8/5+u)))/(4/5),ifelse((u>0.5)&(u<=1),
         (2/5+sqrt(4/25-(8/5)*(1/5-u)))/(4/5),0))
}


rhn <- function(n){
  # simulated values in x
  x <- vector()
  # estimated acceptance probabilities in p
  p <- vector()
  len_x = 0
  mult = 1.1 # 1/approximate acceptance probability

  while(len_x <= n){
    n_to_gen = max((n-len_x)*mult,10) # number to generate, not less than 10
    u1 = runif(n_to_gen)
    u2 = runif(n_to_gen)
    y=utoy(u1)
    Mgy=g(y)*M
    cond <- u2<(fstar(y)/Mgy)
    p <- c(p, sum(cond)/n_to_gen) # keep track of estimated acceptance prob
    x <- c(x, c(y[cond]))# concatenate accepted values
    len_x <- length(x)
  }
  return(list(x=x[1:n], p=p))
}
```

Hide

```
library(ggplot2)

n=50000

ptm <- proc.time() # Start timing
x=rhn(n) # Generate n=50000 number of samples
proc.time() - ptm # Compute time taken

x_rhn <- data.frame(x=x$x) # Extract x_rhn as data
p <- x$p # Extract acceptance rate vector
p # Display
```

The computational time taken by my scheme is approximately 0.1 second when simulating 50,000 samples. This indicates that the code is efficient. In addition, the first element of the p vector demonstrates that in the first 55,000 samples of X simulated, nearly 90% of them are accepted, which is indeed a high acceptance rate.

2. Verification of my scheme

In this section I justified that the samples indeed follow $f_X(\cdot)$ independently, by means of diagnostic plots and statistical tests.

Diagnostic Plots

Below are two diagnostics plots I produced to demonstrate that the data generated are distributed as independent with pdf $f_X(x)$. I firstly plot a histogram with a fitting pdf.

Hide

```
# Histogram
mylabs=list(expression(f[X]^"*"* (x)))
cols = c("fy"="blue")
x=x_rhn$x
x_plot = cbind(x_rhn, fstarx = 1/((x-1)*(2-x))*exp(-(log((x-1)/(2-x)))^2/7)/(0.9*M)) #add an app
roximate scale factor to the unscaled pdf
p <- ggplot(x_plot)
 p+ labs(y="density",title="Histogram and f*(x)")+
  geom_histogram(aes(x,y=..density..),breaks=seq(1, 2, l=30))+
  geom_line(aes(x,fstarx,colour="fy"))+
   scale_colour_manual(name="",values=cols, labels=mylabs)
```

It can be seen from the histogram that the data closely follows the required theoretical distribution. Next, I further proved the independence with the help of lag plot.

Hide

```
n1=500
x1=rhn(n1) # Generate a small number of samples to verify
x_rhn1 <- data.frame(x=x1$x)

gglagplot(x_rhn1$x, lags=4, do.lines=FALSE)+ # Lag plot
  labs(title=expression("Lag Plots of X"))
```

The lag scatter plots of the samples show a random scatter, which further supports that the data are generated independently from $f_X(\cdot)$.

Statistical Tests

Firstly I use a hypothesis test of confidence level 95% to test if the data is randomly sampled from $f_X(\cdot)$.

Suppose I want to compare the frequencies of the first decimal place of the data, denoted by $O_0, \cdots, O_9$, and calculate the theoretical frequencies $E_0, \cdots, E_9$ by numerical integration of the function at subintervals of width 0.1 respectively. Define te statistic

$$S = \sum_{i=0}^{9} \frac{(O_i - E_i)^2}{E_i}$$

which by theory follows a central χ^2_9 distribution. We have null hypothesis

$$H_0 : S \sim \chi^2_9$$

and reject H_0 if the test statistic is sufficiently large, and the confidence region is calculated by quantile function of chi-square distribution in $\mathtt{R}$.

Hide

```
x_1d <- trunc((10*x_rhn)%%10) # Calculate the first decimal place for all sample data
x_n <- ftable(x_1d) # Actual frequency
lower=seq(1,1.9,by=0.1) # Define lower bounds for integrals
integral <- vector()
for (i in lower) {
  s=integrate(fstar,i,(i+0.1))$value # Compute the theoretical frequency of each digit when gene
rating n samples
  integral <- c(integral,s)
}
E <- n*integral/sum(integral) # Vector of theoretical data
s=sum((x_n-E)^2/E) # Test statistic
s<qchisq(0.95,df=9) # Examine if the test statistic falls into the one tail confidence interval
```

The result does not lead to rejecting H_0 so the samples indeed follow $f_X(\cdot)$. I then use time series method to calculate the sample autocorrelation sequence and thus plot the correlogram. Define the test statistic as

$$\hat{\rho_k} = \frac{\sum_{i=1}^{n-k}(X_i - \bar{X})(X_{i+k} - \bar{X})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}, \text{ and } E(\hat{\rho_k}) \approx -\frac{1}{n}, \ Var(\hat{\rho_k}) \approx \frac{1}{n}$$

under the assumption that the samples X_i's are i.i.d. distributed. Therefore, an approximate 95% confidence interval is given by $-\frac{1}{n} \pm \frac{1.96}{\sqrt{n}}$, and this will appear on the correlogram with the autocorrelation values.

Hide

```
ggAcf(x_rhn)+
  labs(title="Autocovariance sequence of generated Data")
```

The auto-correlation plot shows that correlation of almost all number of lags are sufficiently small and fall into the confidence region, and there is no sign of significant correlations at any lag. Hence, this also verifies the properness of my scheme.

3. Estimate normalising constant

In this final section, I implemented crude Monte Carlo algorithm and hit-or-miss Monte Carlo algorithm to estimate the normalising constant of the density function. Specifically, I was trying to estimate

$$I = \int_1^2 f_X^*(x)dx$$

and the normalising constant is just $\frac{1}{I}$.

Note that I repeat each of my schemes for 10 times and calculate variance of the estimates, in order to test stability of the estimators.

I begin with considering $f_X^*(x) = \phi(x)f_U(x)$, where $\phi(x) = f_X^*(x)$ and $f_U(x) = 1$ for $1 < x < 2$. This is equivalent to considering the integral as $E_{f_U}(X)$. Thus I used the estimator

$$\hat{I} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{(X_i-1)(2-X_i)} exp\{-\frac{1}{7}(log(\frac{X_i-1}{2-X_i}))^2\},\ X_i \overset{i.i.d}{\sim} U(1,2)$$

Hide

```
n_MC <- 50000
theta <- 1/integrate(fstar,1,2)$value # True value for the parameter
theta_MC <- vector()
ptm <- proc.time()
for (i in 1:10){
u <- runif(n_MC) # Generate U(0,1) samples
u <- u+1 # Transform into U(1,2)
I_MC <- sum(fstar(u))/n_MC # Estimator of I
theta_MC <- c(theta_MC,1/I_MC) # Append 1/I to the vector of estimates
}
proc.time()-ptm # Processing time
sum((theta_MC-theta)^2)/10 # Variance
```

The scheme runs fast and efficiently, with a small variance, so accuracy and stability are both satisfied.

I then tuned my scheme, by noticing $f_X^*(\cdot)$ being symmetric about $x = \frac{3}{2}$. Therefore

$$I = \int_1^{\frac{3}{2}} 2f_X^*(x)dx$$

. I considered $2f_X^*(x) = \psi(x)f_{U'}(x)$, where $\psi(x) = f_X^*(x)$ and $f_{U'}(x) = 2$ for $1 < x < \frac{3}{2}$. This is equivalent to considering the integral as $E_{f_{U'}}(X)$. Thus I used the estimator

$$\hat{I} = \frac{1}{n}\sum_{j=1}^{n} \frac{1}{(X_j-1)(2-X_j)} exp\{-\frac{1}{7}(log(\frac{X_j-1}{2-X_j}))^2\},\ X_j \overset{i.i.d}{\sim} U(1,\frac{3}{2})$$

Hide

```
theta_MC1 <- vector()
ptm <- proc.time()
for (i in 1:10){
u <- runif(n_MC)
u <- 1+u/2 # Transform into U(1,3/2)
I_MC1 <- sum(fstar(u))/n_MC
theta_MC1 <- c(theta_MC1,1/I_MC1)
}
proc.time()-ptm
sum((theta_MC1-theta)^2)/10
```

By comparison this estimator has a smaller variance. In addition, I used antithetic variates so that I can further decrease the variance of the estimator. This can be achieved by taking $v_i = \frac{5}{2} - u_i$ into consideration for all uniforms $U(1, \frac{3}{2})$ generated.

Hide

```
theta_MC2 <- vector()
ptm <- proc.time()
for (i in 1:10){
u <- runif(n_MC)
u <- 1+u/2 # Transform into U(1,3/2)
v <- 5/2-u # Compute corresponding v=5/2-u to reduce variance
uv <- c(u,v)
I_MC2 <- sum(fstar(uv))/(2*n_MC)
theta_MC2 <- c(theta_MC2,1/I_MC2)
}
proc.time()-ptm
sum((theta_MC2-theta)^2)/10
```

And finally below is how I plugged hit-or-miss Monte Carlo algorithm for the estimation, and calculated the variance of the estimator for ten times of estimation.

Hide

```
n_HMMC=50000
theta_HMMC <- vector()
ptm <- proc.time()
for (i in 1:10){
u = runif(n_HMMC)
v = runif(n_HMMC) # Generate n uniforms for x and y directions respectively
u <- u+1 # Transform to (1,2)
fmax <- optimize(fstar, c(1,2), lower = 1, upper = 1.5,maximum = TRUE,tol=0.001)[[2]]
v <- v*fmax #Transform to (0,fmax)
cond <- v<fstar(u) # Number falls below the curve
theta_HMMC <- c(theta_HMMC,1/(fmax*(2-1)*sum(cond)/n_HMMC))
}
proc.time() - ptm
sum((theta_HMMC-theta)^2)/10
```

It can be seen that hit-or-miss MC takes more time while producing the largest variance, compared with the three crude MC algorithms mentioned above.

In fact, when I tried to reduce variance using substitution, I managed to analytically solved the integral.

$$I = \int_1^2 \frac{1}{(x-1)(2-x)} exp\{-\frac{1}{7}(log(\frac{x-1}{2-x}))^2\}dx$$

Substitute $\frac{x-1}{2-x} = u$, and then $x = 2 - \frac{1}{u+1},\ dx = \frac{1}{(u+1)^2}du$, and

$$I = \int_0^{+\infty} \frac{(u+1)^2}{u} exp\{-\frac{1}{7}log^2(u)\}\frac{1}{(u+1)^2}du = \int_0^{+\infty} \frac{1}{u} exp\{-\frac{1}{7}log^2(u)\}du$$

Using subtitution again by setting $t = log(u)$, then $dt = \frac{1}{u}du$. Therefore

$$I = \int_{-\infty}^{+\infty} exp(-\frac{1}{7}t^2)dt = \sqrt{7\pi}$$

Does $\frac{1}{\sqrt{7\pi}}$ count as an unbiased estimator for the normalising constant with variance 0?