Problem 1

Write and run a computer program to compute a Monte Carlo (including standard errors) of $\mathbb{E}((Y+Z)/(1+|Z|))$, where Y Exponential(3) and Z Normal(0,1) are independent.
Source code:

```
#Estimate E((Y+Z)/(1+|Z|)) where Y~Exp(3) and Z~normal(0,1) and are independent
fn = function(y,z) {(y+z)/(1+abs(z))}

#number of cases
M=10^6

ylist = rexp(M,3)
zlist = rnorm(M)
funclist = fn(ylist,zlist)
funcmean = mean(funclist)
funcse = sd(funclist)/sqrt(M)
cat('Estimate E((Y+Z)/(1+|Z|)) where Y~Exp(3) and Z~Normal(0,1)\n')
cat('M = ',M,'\n')
cat('estimate = ',funcmean,'\n')
cat('standard error = ',funcse,'\n')
```

Output of several runs:

```
## Estimate E((Y+Z)/(1+|Z|)) where Y~Exp(3) and Z~Normal(0,1)
## M =  1e+06
## estimate =  0.2054176
## standard error =  0.0004825309
## Estimate E((Y+Z)/(1+|Z|)) where Y~Exp(3) and Z~Normal(0,1)
## M =  1e+06
## estimate =  0.2056342
## standard error =  0.0004829007
## Estimate E((Y+Z)/(1+|Z|)) where Y~Exp(3) and Z~Normal(0,1)
## M =  1e+06
## estimate =  0.2047581
## standard error =  0.0004828433
## Estimate E((Y+Z)/(1+|Z|)) where Y~Exp(3) and Z~Normal(0,1)
## M =  1e+06
## estimate =  0.2045917
## standard error =  0.0004823873
## Estimate E((Y+Z)/(1+|Z|)) where Y~Exp(3) and Z~Normal(0,1)
## M =  1e+06
## estimate =  0.2043318
## standard error =  0.0004820518
```

The tabulated result is as follows:

Problem 1 continued on next page...

estimation	standard_error
0.2054176	0.0004825
0.2056342	0.0004829
0.2047581	0.0004828
0.2045917	0.0004824
0.2043318	0.0004821

The estimated value for $\mathbb{E}((Y+Z)/(1+|Z|))$ is 0.205 with standard error of approximately 4.8×10^{-4}. The results from the five runs are quite consistent. The number of samples taken for the estimation is set to be 10^6, as this number gives consistent results with reasonable computation power (runs within 1 second).

Problem 2

Re-write the integral

$$I := \int_1^\infty \left(\int_{-\infty}^\infty (1+x^2+\sin(x))^{-|y|^3-2}\, dy \right) dx$$

as some expected value, and then estimate I using Monte Carlo algorithm.

The interval of integration of x is from 1 to ∞, so we choose x to follow a shifted exponential distribution, i.e. $x-1$ follows a exponential(λ) distribution, and the value of λ is set to 1, by trial and error to minimize standard error. The interval of integration of y is $-\infty$ to ∞, so we can choose y to follow a Normal$(0,\sigma^2)$ distribution, and the value σ is also found to be 1 by methods of trial and error. Now, we can write the integral as

$$\begin{aligned}
I :&= \int_1^\infty \left(\int_{-\infty}^\infty (1+x^2+\sin(x))^{-|y|^3-2}\, dy \right) dx \\
&= \int_1^\infty \int_{-\infty}^\infty (1+x^2+\sin(x))^{-|y|^3-2}\sqrt{2\pi}e^{\frac{y^2}{2}} \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) dx \\
&= \int_1^\infty \int_{-\infty}^\infty (1+x^2+\sin(x))^{-|y|^3-2}\sqrt{2\pi}e^{\frac{y^2}{2}} \frac{1}{e^{(x-1)}} \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) \left(e^{-(x-1)} dx \right) \\
&= \int_0^\infty \int_{-\infty}^\infty (1+(x+1)^2+\sin(x+1))^{-|y|^3-2}\sqrt{2\pi}e^{\frac{y^2}{2}} \frac{1}{e^{x}} \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) \left(e^{-x} dx \right) \\
&= \mathbb{E}\left[\sqrt{2\pi}(X^2+2X+2+\sin(X+1))^{-|Y|^3-2} \exp\{\frac{Y^2}{2}+X\} \right],
\end{aligned}$$

where X follows a Exponential(1) distribution and Y follows a Normal$(0,1)$ distribution.
Source code:

```
#Estimate E[(sigma(2pi)^0.5\lambda)(X^2+2X+2+sin(X+1))^(-|Y|^3-2)exp(Y^2/2sigma^2+X)]
#where X~Exp(lambda) and Y~normal(0,sigma^2) and are independent
sigma = 1
lambda = 1
fn = function(x,y)
  {sigma/lambda*sqrt(2*pi)*(x^2+2*x+2+sin(x+1))^(-2-(abs(y))^3)*exp(y^2/2/sigma^2+x)}

#number of cases
M=10^6

xlist = rexp(M,1/lambda)
```

```
ylist = rnorm(M,0,sigma)
funclist = fn(xlist,ylist)
funcmean = mean(funclist)
funcse = sd(funclist)/sqrt(M)
cat('Using Monte Carlo Integration methods with \n')
cat('M = ',M,', sigma = ',sigma,', lambda = ',lambda,'\n')
cat('estimate = ',funcmean,'\n')
cat('standard error = ',funcse,'\n')
```

Output of several runs:

```
## Using Monte Carlo Integration methods with
## M =  1e+06 , sigma =  1 , lambda =  1
## estimate =  0.1393809
## standard error =  0.0001244963
## Using Monte Carlo Integration methods with
## M =  1e+06 , sigma =  1 , lambda =  1
## estimate =  0.1396676
## standard error =  0.0001337729
## Using Monte Carlo Integration methods with
## M =  1e+06 , sigma =  1 , lambda =  1
## estimate =  0.1395552
## standard error =  0.0001363247
## Using Monte Carlo Integration methods with
## M =  1e+06 , sigma =  1 , lambda =  1
## estimate =  0.1398393
## standard error =  0.0001878718
## Using Monte Carlo Integration methods with
## M =  1e+06 , sigma =  1 , lambda =  1
## estimate =  0.1395777
## standard error =  0.0001148437
```

The tabulated result is as follows:

estimation	standard_error
0.1393809	0.0001245
0.1396676	0.0001338
0.1395552	0.0001363
0.1398393	0.0001879
0.1395777	0.0001148

The integral is estimated to be 0.140 with standard error of approximately 0.000 with Monte Carlo methods. The results from the five runs are quite consistent. The number of samples taken for the estimation is set to be 10^6, as this number gives consistent results with reasonable computation power (runs within 1 second).

For the next four questions, let A, B, C, and D be the last four digits of your student number. Let $g : \mathbb{R}^5 \to [0, \infty)$ be the function defined by:

$$g(x_1, x_2, x_3, x_4, x_5) = (x_1 + A + 2)^{x_2+3} + (1 + \cos[(B + 3)x_3])\,(e^{(12-C)x_4}|x_4 - 3x_5|^{D+2} \prod_{i=1}^{5} \mathbb{1}_{0<x_i<2}$$

and let $\pi(x_1, x_2, x_3, x_4, x_5) = cg(x_1, x_2, x_3, x_4, x_5)$ be the corresponding five-dimensional probability density function, with unknown normalising constant c. Finally, let f be the uniform density on $[0, 2]^5$.

Problem 3

Identify the values of A, B, C, and D.

$$A = 7, B = 6, C = 6, D = 3$$

Problem 4

Write and run a computer program to estimate $\mathbb{E}_\pi[(X_1 - X_2)/(1 + X_3 + X_4X_5)]$by importance sampler with the above f. Discuss the extent to which this algorithm works well.
First write the expectation under the original distribution in terms of an expectation under f, and write $x = (x_1, x_2, x_3, x_4, x_5)$ below

$$\begin{aligned}
\mathbb{E}_\pi\left[\frac{X_1 - X_2}{1 + X_3 + X_4X_5}\right] &= \int \frac{x_1 - x_2}{1 + x_3 + x_4x_5}\pi(x)\,dx \\
&= \int \frac{\frac{x_1-x_2}{1+x_3+x_4x_5}\frac{g(x)}{f(x)}f(x)\,dx}{\frac{g(x)}{f(x)}f(x)\,dx} \\
&= \frac{\mathbb{E}_f[\frac{x_1-x_2}{1+x_3+x_4x_5}\frac{g(x)}{f(x)}]}{\mathbb{E}_f[\frac{g(x)}{f(x)}]} \\
&\approx \frac{\frac{1}{M}\sum_{i=1}^M\left(\frac{x_1^{(i)}-x_2^{(i)}}{1+x_3^{(i)}+x_4^{(i)}x_5^{(i)}}\frac{g(x^{(i)})}{f(x^{(i)})}\right)}{\frac{1}{M}\sum_{i=1}^M\left(\frac{g(x^{(i)})}{f(x^{(i)})}\right)} \\
&= \frac{\sum_{i=1}^M\left(\frac{x_1^{(i)}-x_2^{(i)}}{1+x_3^{(i)}+x_4^{(i)}x_5^{(i)}}\frac{g(x^{(i)})}{f(x^{(i)})}\right)}{\sum_{i=1}^M\left(\frac{g(x^{(i)})}{f(x^{(i)})}\right)}
\end{aligned}$$

Since for all points sampled under f, $f(x_1, x_2, x_3, x_4, x_5) = \frac{1}{32}$,

$$\mathbb{E}_\pi\left[\frac{X_1 - X_2}{1 + X_3 + X_4X_5}\right] = \frac{\sum_{i=1}^M\left(\frac{x_1^{(i)}-x_2^{(i)}}{1+x_3^{(i)}+x_4^{(i)}x_5^{(i)}}32g(x^{(i)})\right)}{\sum_{i=1}^M\left(32g(x^{(i)})\right)} = \frac{\sum_{i=1}^M\left(\frac{x_1^{(i)}-x_2^{(i)}}{1+x_3^{(i)}+x_4^{(i)}x_5^{(i)}}g(x^{(i)})\right)}{\sum_{i=1}^M\left(g(x^{(i)})\right)}$$

Notice that the importance sampling algorithm reduces to evaluating the expectation with integration, since f is constant.
Also, because we are using importance sampling, we can not get an estimated of the standard error from a single run. So we estimate the standard error using results from multiple (10) runs.

Source code:

```
#Estimate E_pi((X1-X2)/(1+X3+X4X5)) where pi=cg using importance sampling techniques
# with density f = uniform(0,2)^5

A = 7; B = 6; C = 6; D = 3

fn = function(x1,x2,x3,x4,x5) {(x1-x2)/(1+x3+x4*x5)}
g = function(x1,x2,x3,x4,x5)
```

```
  {(x1+A+2)^(x2+3)*(1+cos((B+3)*x3))*(exp((12-C)*x4))*abs(x4-3*x5)^(D+2)}

#number of runs
R=10
#number of cases
M=10^6

estlist=numeric(R)

cat('M = ',M,'\n')
for (i in 1:R) {
  #sample from unif(0,2)^5
  x1 = runif(M,0,2)
  x2 = runif(M,0,2)
  x3 = runif(M,0,2)
  x4 = runif(M,0,2)
  x5 = runif(M,0,2)

  fnlist = fn(x1,x2,x3,x4,x5)
  glist = g(x1,x2,x3,x4,x5)
  estlist[i] = sum(fnlist*glist)/sum(glist)
  cat('run = ',i,', estimate = ',estlist[i],'\n')
}

estmean = mean(estlist)
estse = sd(estlist)

cat('mean estimate = ',funcmean,'\n')
cat('estimate standard error = ',funcse,'\n')
```

Output of several runs:

```
## M =  1e+06
## run =  1 , estimate =  -0.08714028
## run =  2 , estimate =  -0.08637034
## run =  3 , estimate =  -0.08834511
## run =  4 , estimate =  -0.08894496
## run =  5 , estimate =  -0.08718104
## run =  6 , estimate =  -0.08830067
## run =  7 , estimate =  -0.08921454
## run =  8 , estimate =  -0.08562825
## run =  9 , estimate =  -0.08842424
## run =  10 , estimate =  -0.08817753
## mean estimate =  -0.08777848
## estimate standard error =  0.005801013
```

The tabulated result is as follows:

$$\begin{array}{ccccc} -0.0871403 & -0.0863703 & -0.0883451 & -0.088945 & -0.087181 \\ -0.0883007 & -0.0892145 & -0.0856282 & -0.0884242 & -0.0881775 \end{array}$$

Problem 4 continued on next page...

The mean of all estimates is -0.0877727 with standard error 0.0011529.
The algorithm works reasonably well because using importance sampling, we can easily pick samples from a much easier distribution. The result seems to be strong as well as the estimates across different runs are fairly consistent and the standard error is small.

Problem 5

Write and run a computer program to estimate $\mathbb{E}_\pi[(X_1 - X_2)/(1 + X_3 + X_4X_5)]$ by using a rejection sampler with the above f. Discuss the extent to which this algorithm works well.

First let $(x1, x2, x3, x4, x5)$ be denoted as x. In order to use the rejection sampler, we have to find a constant K such that $g(x) \leq Kf(x)$, so we need to find the maximum value of g in the box $(0, 2)^5$. The maximum value is found by using optim in R:

```
goptim = function(x) {
  x1=x[1]
  x2=x[2]
  x3=x[3]
  x4=x[4]
  x5=x[5]
  -(x1+A+2)^(x2+3)*(1+cos((B+3)*x3))*(exp((12-C)*x4))*abs(x4-3*x5)^(D+2)
}
optim(rep(1,5),goptim,NULL,method="L-BFGS-B",lower=rep(0,5),upper=rep(2,5))
```

Notice that optim finds the minimum instead of maximum value, so we have to define goptim as the negative of function $g(x)$. The result of running the code above gives:

```
## $par
## [1] 2 2 0 2 2
##
## $value
## [1] -5.368181e+13
##
## $counts
## function gradient
##        2        2
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: NORM OF PROJECTED GRADIENT <= PGTOL"
```

Since $f(x) = \frac{1}{32}$ is constant in $(0, 2)^5$, we need to set $K \geq 32 * \max(g)$ =1.718e+15, and since K does not need to be tight bound, we pick K =2e+15 for easier computation. Now we can proceed to estimate the expectation $\mathbb{E}_\pi[(X_1 - X_2)/(1 + X_3 + X_4X_5)]$ by using a rejection sampler with f. Note that we adjust the number of samples taken from f to 1e+7 this time because the low acceptance rate. We have to increase the number of samples in order to have enough samples for the π distribution.
Source code:

Problem 5 continued on next page...

```
#Estimate E_pi((X1-X2)/(1+X3+X4X5)) where pi=cg using rejection sampling techniques
# with density f = uniform(0,2)^5

A = 7; B = 6; C = 6; D = 3

fn = function(x1,x2,x3,x4,x5) {(x1-x2)/(1+x3+x4*x5)}
g = function(x1,x2,x3,x4,x5)
{(x1+A+2)^(x2+3)*(1+cos((B+3)*x3))*(exp((12-C)*x4))*abs(x4-3*x5)^(D+2)}
K = 2e+15

M = 1e+7

#sample from f
x1 = runif(M,0,2)
x2 = runif(M,0,2)
x3 = runif(M,0,2)
x4 = runif(M,0,2)
x5 = runif(M,0,2)

#generate r.v. for acceptance
u = runif(M)
acc = (u<=32*g(x1,x2,x3,x4,x5)/K)

fnlist = fn(x1[acc],x2[acc],x3[acc],x4[acc],x5[acc])
funcmean = mean(fnlist)
funcse = sd(funclist)/sqrt(sum(acc))
accrate = sum(acc)/M

cat('M = ',M,'\n')
cat('Number of samples accepted = ',sum(acc),', acceptance rate = ',accrate,'\n')
cat('Estimate = ',funcmean,'\n')
cat('Standard error = ',funcse,'\n')
```

Output of several runs:

```
## M =  1e+07
## Number of samples accepted =  6833 , acceptance rate =  0.0006833
## Estimate =  -0.08753893
## Standard error =  0.005837405
## M =  1e+07
## Number of samples accepted =  6794 , acceptance rate =  0.0006794
## Estimate =  -0.088423
## Standard error =  0.005854135
## M =  1e+07
## Number of samples accepted =  6885 , acceptance rate =  0.0006885
## Estimate =  -0.08980898
## Standard error =  0.005815319
## M =  1e+07
## Number of samples accepted =  6915 , acceptance rate =  0.0006915
```

```
## Estimate =  -0.08679519
## Standard error =  0.005802691
## M =  1e+07
## Number of samples accepted =  6973 , acceptance rate =  0.0006973
## Estimate =  -0.09099398
## Standard error =  0.005778508
```

The tabulated result is as follows:

estimation	standard_error
-0.0875389	0.0058374
-0.0884230	0.0058541
-0.0898090	0.0058153
-0.0867952	0.0058027
-0.0909940	0.0057785

The estimated value for $\mathbb{E}_\pi[(X_1 - X_2)/(1 + X_3 + X_4X_5)]$ is -0.089 with standard error of approximately 0.00582. The results from the five runs are quite consistent.
However, since the acceptance rate is so low (about 6.88×10^{-4}), the algorithm is not as effective as the importance sampler in Problem 4. The computation time is a lot longer and standard error of the result is also larger than the importance sampler in Problem 4.

Problem 6

Write and run a computer program to estimate $\mathbb{E}_\pi[(X_1 - X_2)/(1 + X_3 + X_4X_5)]$ using an MCMC algorithm of your choice, and obtain the best estimate you can. Include some discussion of accuracy, uncertainty, standard errors, etc.

For this problem, we will use a Random Walk Metropolis algorithm to estimate the expectation. The increments in the random walk follow a $N(0, \sigma^2 I)$ distribution, where we will set the parameter $\sigma = 0.2$ (by trial and error to minimize se).

Source code:

```
#Estimate E_pi((X1-X2)/(1+X3+X4X5)) where pi=cg using RWM algorithm
# with increments following a N(0,sigma^2*I) distribution

A = 7; B = 6; C = 6; D = 3

fn = function(x1,x2,x3,x4,x5) {(x1-x2)/(1+x3+x4*x5)}
g = function(x1,x2,x3,x4,x5) {
  if (x1<=0 | x1>=2 | x2<=0 | x2>=2 |x3<=0 | x3>=2 |
      x4<=0 | x4>=2 | x5<=0 | x5>=2)
    0
  else
    (x1+A+2)^(x2+3)*(1+cos((B+3)*x3))*(exp((12-C)*x4))*abs(x4-3*x5)^(D+2)
}

sigma = 0.2
```

```
M = 1e+6
B = 1e+5

#initialization
fnlist = numeric(M+B)
x1 = runif(1,0,2)
x2 = runif(1,0,2)
x3 = runif(1,0,2)
x4 = runif(1,0,2)
x5 = runif(1,0,2)
acc = 0

for (i in 1:(M+B)) {
  eps = rnorm(5,0,sigma)

  if (runif(1)<=g(x1+eps[1],x2+eps[2],x3+eps[3],x4+eps[4],x5+eps[5])/
        g(x1,x2,x3,x4,x5)) {
    if (i>B)
      acc=acc+1
    x1=x1+eps[1]
    x2=x2+eps[2]
    x3=x3+eps[3]
    x4=x4+eps[4]
    x5=x5+eps[5]
  }
  fnlist[i] = fn(x1,x2,x3,x4,x5)
}

funcmean = mean(fnlist[(B+1):(M+B)])
funciidse = sd(fnlist[(B+1):(M+B)])/sqrt(M)
acf_k = acf(fnlist[(B+1):(M+B)],lag.max = 1000,plot = FALSE)$acf
varfact = 2*sum(acf_k[1:min(which(acf_k<0.05))])-1
funcse = funciidse*sqrt(varfact)
accrate = acc/M

cat('B = ',B,', M = ',M,'\n')
cat('Number of samples accepted = ',acc,', acceptance rate = ',accrate,'\n')
cat('Estimate = ',funcmean,'\n')
cat('i.i.d. standard error = ',funciidse,'\n')
cat('varfact = ',varfact,'\n')
cat('Standard error = ',funcse,'\n')
```

Output of several runs:

```
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  173759 , acceptance rate =  0.173759
## Estimate =  -0.08263961
## i.i.d. standard error =  0.000135386
## varfact =  175.5155
```

```
## Standard error =  0.001793624
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  173366 , acceptance rate =  0.173366
## Estimate =  -0.08711674
## i.i.d. standard error =  0.0001361279
## varfact =  183.7932
## Standard error =  0.001845491
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  173740 , acceptance rate =  0.17374
## Estimate =  -0.08693576
## i.i.d. standard error =  0.0001343732
## varfact =  172.7299
## Standard error =  0.001766024
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  173184 , acceptance rate =  0.173184
## Estimate =  -0.08750525
## i.i.d. standard error =  0.0001351836
## varfact =  175.3872
## Standard error =  0.001790288
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  172432 , acceptance rate =  0.172432
## Estimate =  -0.08019379
## i.i.d. standard error =  0.0001348094
## varfact =  182.1287
## Standard error =  0.001819321
```

The tabulated result is as follows:

estimation	acc_rate	seiid	varfact	standard_error
-0.0826396	0.173759	0.0001354	182.1287	0.0017936
-0.0871167	0.173366	0.0001361	182.1287	0.0018455
-0.0869358	0.173740	0.0001344	182.1287	0.0017660
-0.0875052	0.173184	0.0001352	182.1287	0.0017903
-0.0801938	0.172432	0.0001348	182.1287	0.0018193

From the table above, we can first notice that the estimated result is consistent with the previous samplers, but the range of the estimation is slightly larger here using the MCMC sampler, while the standard error is smaller in this case. Also, we see that the acceptance rate is about 17%, and is close the the optimal 23%. To verify that the MCMC sampler works effectively, we plot the value of the estimated function after the burn-in stage

```
fnlist6 = fnlist
plot(fnlist6[(B+1):(B+M)],type='l')
```

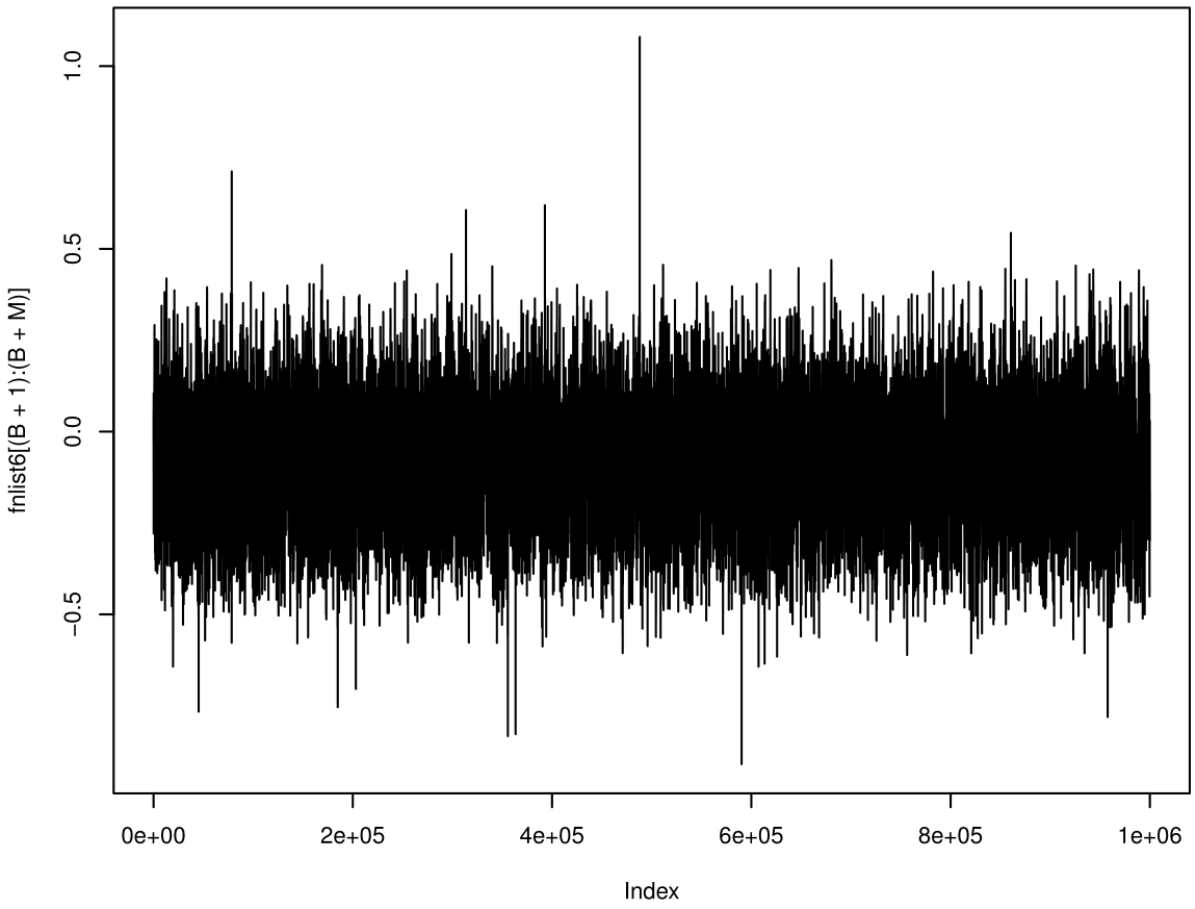


Figure 1: function values of MCMC algorithm after burning-in

From the figure above, we can see that the sample is reasonably well represented (at least the value of the function itself varies significantly around its mean value), indicating a good (accurate) approximation by the MCMC algorithm.

Problem 7

Consider an independence sampler algorithm on $\chi = (1, \infty)$, where $\pi(x) = 5x^{-6}$ and $q(x) = rx^{-r-1}$ for some choice of $r > 0$, with identity functional $h(x) = x$.
(a) For what value of r will the algorithm provide i.i.d. samples?
The algorithm provide i.i.d. samples for $r = 5$, because when $r = 5$, $\pi(x) = q(x)$. This will eliminate any rejections in the independence sampler, so the sample will be i.i.d.
(b) For what values of r will the sampler be geometrically ergodic?
Independence sampler is geometrically ergodic if and only if there is $\delta > 0$ such that $q(x) \geq \delta\pi(x)$ for π-a.e.

$x \in \chi$. With $\chi = (1, \infty)$, we need to be able to pick $\delta > 0$ so that

$$\sup_{x \geq 1} \frac{\pi(x)}{q(x)} =\leq \frac{1}{\delta}$$

This means that for any finite $x > 1$, we require

$$\frac{pi(x)}{q(x)} = \frac{5x^{-6}}{rx^{-r-1}} = \frac{5}{r} x^{r-5} < \infty$$

So we require $r - 5 \leq 0$, i.e. $r \leq 5$. Also since $q(x) = rx^{-r-1}$ is a valid density function, we require $r > 0$. Therefore, $r \in (0, 5]$.
(c) For $r = 1/20$, find a number n such that $D(x, n) < 0.01$ for all $x \in \chi$.
For $r = 1/20$,

$$\frac{\pi(x)}{q(x)} = \frac{5}{\frac{1}{20}} x^{\frac{1}{20}-5} = 100x^{-0.99}$$

Therefore, for $x \in \chi = (1, \infty)$,

$$\frac{\pi(x)}{q(x)} \leq 100$$

so we can pick $\delta = \frac{1}{100}$. Now, as

$$D(x, n) \leq (1 - \delta)^n$$

we can simply choose n such that $(1 - \delta)^n < 0.01$, so $n > \log(0.01)/\log(0.99) = 458.21$. Therefore, $n = 459$ will suffice.
(d) Write and run a computer program to estimate $\mathbb{E}_\pi(h)$ with this algorithm in the two cases $r = 1/20$ and $r = 10$, each with $M = 10^5$ and $B = 10^4$. Estimate the corresponding standard errors by two different methods: (i) using "varfact", and (ii) from repeated independent runs.
First, in order to sample from $q(x)$, we notice that if U Uniform(0,1),

$$P((1 - U)^{-\frac{1}{r}} \leq x) = P(1 - U \geq x^{-r}) = P(U \leq 1 - x^{-r}) = P(U \leq F(x)) = F(x)$$

Therefore, $X =_d (1 - U)^{-\frac{1}{r}}$.
Then we can implement the independence sampler with the following code:

```
#Estimate E_pi(x) where pi=5x^-6 independence sampler algorithm
# with proposal distribution q(x) = r x^{-r-1}

h = function(x) {x}
g = function(x) {5*x^(-6)}
q = function(x,r) {r*x^(-r-1)}

runs = 10

M = 1e+5
B = 1e+4

funcmean = numeric(runs)
funciidse = numeric(runs)
varfact = numeric(runs)
funcse = numeric(runs)
```

```
accrate = numeric(runs)

cat('B = ',B,', M = ',M,'\n')
for (i in 1:runs) {
  acc = 0
  fnlist = numeric(M+B)
  x = (1-runif(1))^(-1/r)
  for (j in 1:(B+M)) {
    y = (1-runif(1))^(-1/r)
    if (runif(1)<=g(y)*q(x,r)/g(x)/q(y,r)) {
      acc = acc+1
      x = y
    }
    fnlist[j] = h(x)
  }

  funcmean[i] = mean(fnlist[(B+1):(M+B)])
  funciidse[i] = sd(fnlist[(B+1):(M+B)])/sqrt(M)
  acf_k = acf(fnlist[(B+1):(M+B)],lag.max = 1000,plot = FALSE)$acf
  varfact[i] = 2*sum(acf_k)-1
  funcse[i] = funciidse[i]*sqrt(varfact[i])
  accrate[i] = acc/M

  cat('Number of samples accepted = ',acc,', acceptance rate = ',accrate[i],'\n')
  cat('Estimate = ',funcmean[i],'\n')
  cat('i.i.d. standard error = ',funciidse[i],'\n')
  cat('varfact = ',varfact[i],'\n')
  cat('Standard error = ',funcse[i],'\n')
}
meanse <- data.frame(funcmean,funciidse,varfact,funcse,accrate)
estse = sd(funcmean)
cat('Standard error calculated from multiple runs = ',estse,'\n')
```

with $r = 1/20$, the output of running the program is printed below:

```
## B =  10000 , M =  1e+05
## Number of samples accepted =  2093 , acceptance rate =  0.02093
## Estimate =  1.238635
## i.i.d. standard error =  0.0009258633
## varfact =  114.0854
## Standard error =  0.009889218
## Number of samples accepted =  2180 , acceptance rate =  0.0218
## Estimate =  1.237005
## i.i.d. standard error =  0.0009949706
## varfact =  72.25659
## Standard error =  0.008457635
## Number of samples accepted =  2203 , acceptance rate =  0.02203
## Estimate =  1.243761
## i.i.d. standard error =  0.0009788405
```

```
## varfact =  91.95337
## Standard error =  0.009386328
## Number of samples accepted =  2205 , acceptance rate =  0.02205
## Estimate =  1.26803
## i.i.d. standard error =  0.001196367
## varfact =  114.8249
## Standard error =  0.01281984
## Number of samples accepted =  2239 , acceptance rate =  0.02239
## Estimate =  1.26214
## i.i.d. standard error =  0.001043855
## varfact =  132.7526
## Standard error =  0.01202712
## Number of samples accepted =  2237 , acceptance rate =  0.02237
## Estimate =  1.240768
## i.i.d. standard error =  0.0009611
## varfact =  65.51433
## Standard error =  0.007779232
## Number of samples accepted =  2167 , acceptance rate =  0.02167
## Estimate =  1.259207
## i.i.d. standard error =  0.001016062
## varfact =  108.19
## Standard error =  0.01056851
## Number of samples accepted =  2188 , acceptance rate =  0.02188
## Estimate =  1.250411
## i.i.d. standard error =  0.001161109
## varfact =  89.67467
## Standard error =  0.01099532
## Number of samples accepted =  2165 , acceptance rate =  0.02165
## Estimate =  1.25477
## i.i.d. standard error =  0.001067317
## varfact =  103.8663
## Standard error =  0.01087754
## Number of samples accepted =  2212 , acceptance rate =  0.02212
## Estimate =  1.252788
## i.i.d. standard error =  0.001087062
## varfact =  108.7818
## Standard error =  0.01133789
## Standard error calculated from multiple runs =  0.01056452
```

with $r = 10$, the output of running the program is printed below:

```
## B =  10000 , M =  1e+05
## Number of samples accepted =  73929 , acceptance rate =  0.73929
## Estimate =  1.24135
## i.i.d. standard error =  0.0009011254
## varfact =  46.05885
## Standard error =  0.006115638
## Number of samples accepted =  74109 , acceptance rate =  0.74109
## Estimate =  1.227835
```

```
## i.i.d. standard error =  0.0007831439
## varfact =  20.71352
## Standard error =  0.003564253
## Number of samples accepted =  73333 , acceptance rate =  0.73333
## Estimate =  1.245432
## i.i.d. standard error =  0.000920215
## varfact =  101.153
## Standard error =  0.009255046
## Number of samples accepted =  74465 , acceptance rate =  0.74465
## Estimate =  1.233405
## i.i.d. standard error =  0.0008525449
## varfact =  42.70032
## Standard error =  0.005570996
## Number of samples accepted =  73682 , acceptance rate =  0.73682
## Estimate =  1.24653
## i.i.d. standard error =  0.0009628246
## varfact =  124.5792
## Standard error =  0.01074657
## Number of samples accepted =  73657 , acceptance rate =  0.73657
## Estimate =  1.239802
## i.i.d. standard error =  0.0009025768
## varfact =  124.087
## Standard error =  0.0100542
## Number of samples accepted =  73517 , acceptance rate =  0.73517
## Estimate =  1.244985
## i.i.d. standard error =  0.0009591974
## varfact =  177.6196
## Standard error =  0.01278361
## Number of samples accepted =  73592 , acceptance rate =  0.73592
## Estimate =  1.246485
## i.i.d. standard error =  0.0009477594
## varfact =  51.65175
## Standard error =  0.006811466
## Number of samples accepted =  73218 , acceptance rate =  0.73218
## Estimate =  1.247764
## i.i.d. standard error =  0.0009336237
## varfact =  37.46121
## Standard error =  0.005714297
## Number of samples accepted =  73434 , acceptance rate =  0.73434
## Estimate =  1.24259
## i.i.d. standard error =  0.0008906098
## varfact =  60.98202
## Standard error =  0.00695486
## Standard error calculated from multiple runs =  0.00643962
```

For ease of reading the result, we tabulate the result in the following two tables

funcmean	funciidse	varfact	funcse	accrate
1.238635	0.0009259	114.08541	0.0098892	0.02093
1.237005	0.0009950	72.25659	0.0084576	0.02180
1.243761	0.0009788	91.95337	0.0093863	0.02203
1.268030	0.0011964	114.82491	0.0128198	0.02205
1.262140	0.0010439	132.75263	0.0120271	0.02239
1.240768	0.0009611	65.51433	0.0077792	0.02237
1.259207	0.0010161	108.18999	0.0105685	0.02167
1.250411	0.0011611	89.67467	0.0109953	0.02188
1.254770	0.0010673	103.86630	0.0108775	0.02165
1.252789	0.0010871	108.78176	0.0113379	0.02212

for r=1/20

mean of estimate = 1.2507516
mean of estimate standard error = 0.0104139
standard deviation calculated from multiple runs = 0.0015607
mean of acceptance rate = 0.021889

funcmean	funciidse	varfact	funcse	accrate
1.241350	0.0009011	46.05885	0.0061156	0.73929
1.227835	0.0007831	20.71352	0.0035643	0.74109
1.245432	0.0009202	101.15297	0.0092550	0.73333
1.233405	0.0008525	42.70032	0.0055710	0.74465
1.246530	0.0009628	124.57925	0.0107466	0.73682
1.239802	0.0009026	124.08702	0.0100542	0.73657
1.244985	0.0009592	177.61965	0.0127836	0.73517
1.246485	0.0009478	51.65175	0.0068115	0.73592
1.247764	0.0009336	37.46121	0.0057143	0.73218
1.242590	0.0008906	60.98202	0.0069549	0.73434

for r=10

mean of estimate = 1.2416177
mean of estimate standard error = 0.0077571
standard deviation calculated from multiple runs = 0.002838
mean of acceptance rate = 0.736936

(e) First notice that the standard error calculated from varfact and i.i.d standard error is slightly larger than the standard error calculated from multiple runs in both cases. Also notice that the two estimates are quite close to each other, and the standar errors are also quite similar in the cases where $r = 1/20$ and $r = 10$. This result is quite surprising because the acceptance rate for the two cases are completely different. For $r = 1/20$, the acceptance rate is just over 0.02, where as the acceptance rate is about 0.73 for the $r = 10$ case.

Also, for $r = 1/20$ the sampler is geometrially ergodic and should be more reliable, at least to the extend that we know the Central Limit Theorem will hold for sure.

Finally, we can analytically calculate the mean and standard deviation

$$\mathbb{E}_\pi h = \int_1^\infty 5x^{-5}dx = -\frac{5}{4}x^{-4}\Big|_1^\infty = \frac{5}{4}$$
$$\mathbb{E}_\pi h^2 = \int_1^\infty 5x^{-4}dx = -\frac{5}{3}x^{-3}\Big|_1^\infty = \frac{5}{3}$$
$$\sigma = \sqrt{\frac{5}{3} - \left(\frac{5}{4}\right)^2} = 0.3227$$
$$\frac{\sigma}{\sqrt{M}} \approx 0.001$$

Both 95% confidence intervals contain the exacty value.

Problem 8

Let $\chi = \mathbb{R}$, and let $\pi(x) = cg(x)$, where $g(x) = e^{-|x|/10}(1 + \cos(x)\sin(x))$, and let $h(x) = x + x^2$. With appropriate choise of M, B, σ, and starting distribution $\mathcal{L}(X_0)$, estimate $\mathbb{E}_\pi(h)$ in each of the two different ways: (a) With a usual random-walk Metropolis algorithm for π, with the usual proposal distribution Y_n $N(X_{n-1}, \sigma^2)$. For this problem, we choose again M to be 10e+6 and B to be 10e+5, and we choose σ to be 53 so that the acceptance rate is around the optimal value 0.234. The RWM algorithm is implemented using the following code:

```
#Estimate E_pi(X+X^2) where pi=cg using RWM algorithm where
# g(x) = e^(-|x|/10)*(1+cos(x)sin(x^3))
# with increments following a N(0,sigma^2) distribution

fn = function(x) {x+x^2}
g = function(x) {exp(-abs(x)/10)*(1+cos(x)*sin(x^3))}

sigma = 53
M = 1e+6
B = 1e+5

#initialization
fnlist = numeric(M+B)
xlist = numeric(M+B)
x = rnorm(1,0,sigma)
acc = 0

for (i in 1:(M+B)) {
  eps = rnorm(1,0,sigma)

  if (runif(1)<=g(x+eps)/g(x)) {
    if (i>B)
      acc=acc+1
    x = x+eps
  }
  xlist[i] = x
  fnlist[i] = fn(x)
```

```
}

funcmean = mean(fnlist[(B+1):(M+B)])
funciidse = sd(fnlist[(B+1):(M+B)])/sqrt(M)
acf_k = acf(fnlist[(B+1):(M+B)],lag.max = 1000,plot = FALSE)$acf
varfact = 2*sum(acf_k[1:min(which(acf_k<0.05))])-1
funcse = funciidse*sqrt(varfact)
accrate = acc/M

cat('B = ',B,', M = ',M,'\n')
cat('Number of samples accepted = ',acc,', acceptance rate = ',accrate,'\n')
cat('Estimate = ',funcmean,'\n')
cat('i.i.d. standard error = ',funciidse,'\n')
cat('varfact = ',varfact,'\n')
cat('Standard error = ',funcse,'\n')
```

Output of several runs:

```
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  238066 , acceptance rate =  0.238066
## Estimate =  200.6944
## i.i.d. standard error =  0.4448528
## varfact =  6.844363
## Standard error =  1.163812
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  237394 , acceptance rate =  0.237394
## Estimate =  199.571
## i.i.d. standard error =  0.4484531
## varfact =  6.913195
## Standard error =  1.179116
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  238054 , acceptance rate =  0.238054
## Estimate =  200.1975
## i.i.d. standard error =  0.4433383
## varfact =  6.77882
## Standard error =  1.154283
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  238477 , acceptance rate =  0.238477
## Estimate =  200.3298
## i.i.d. standard error =  0.4522812
## varfact =  7.017603
## Standard error =  1.198127
## B =  1e+05 , M =  1e+06
## Number of samples accepted =  237175 , acceptance rate =  0.237175
## Estimate =  198.6471
## i.i.d. standard error =  0.4413461
## varfact =  6.776058
## Standard error =  1.148862
```

The tabulated result is as follows:

estimation	acc_rate	seiid	varfact	standard_error
200.6944	0.238066	0.4448528	6.776058	1.163812
199.5710	0.237394	0.4484531	6.776058	1.179116
200.1975	0.238054	0.4433383	6.776058	1.154283
200.3298	0.238477	0.4522812	6.776058	1.198127
198.6471	0.237175	0.4413461	6.776058	1.148862

We notice that the varfact values are quite small, indicating that the samples are not strongly correlated. To verify the effectiveness of the sampler, here we plot the value of x after the burn-in stage

```
xlist8a = xlist
plot(xlist8a[(B+1):(B+M)],type='l')
```

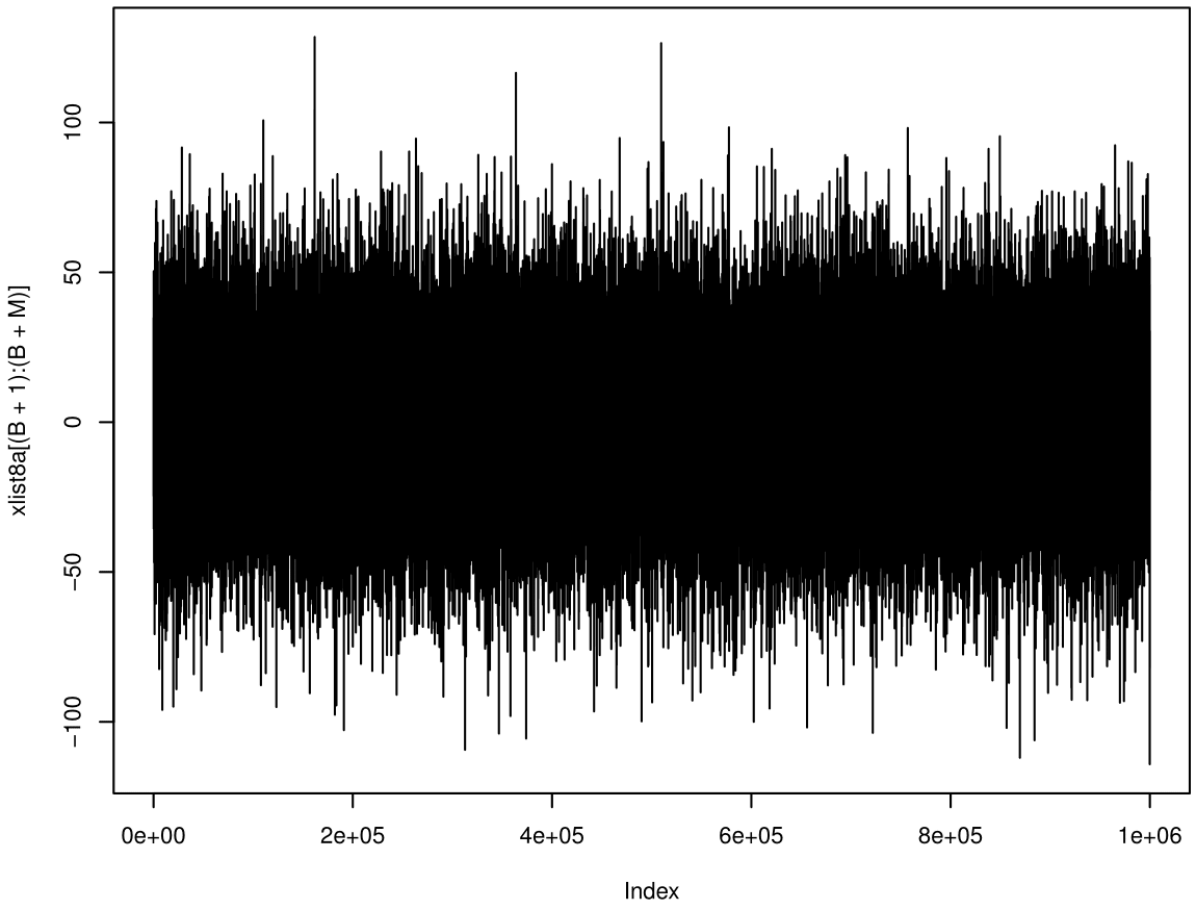


Figure 2: x values of RWM algorithm after burning-in

From the figure above, we can see that the sample is reasonably well represented because of the high variability and low correlation, indicating a good (accurate) approximation by the RWM algorithm.

(b) With a Langevin (Metropolis-Hastings) algorithm with proposals $Y_n\ N(X_{n-1}+\frac{1}{2}\sigma^2 g'(X_{n-1})/g(X_{n-1}), \sigma^2)$. First we calculate analycically $g'(x)$:

$$\begin{aligned}\frac{dg}{dx} &= \begin{cases} \frac{d}{dx}\left(e^{-x/10}(1+\cos(x)\sin(x^3))\right) & \text{for } x>0 \\ \frac{d}{dx}\left(e^{x/10}(1+\cos(x)\sin(x^3))\right) & \text{for } x<0 \end{cases} \\ &= \begin{cases} -\frac{1}{10}e^{-x/10}(1+\cos(x)\sin(x^3)) + e^{-x/10}(3x^2\cos(x)\cos(x^3)-\sin(x)\sin(x^3)) & \text{for } x>0 \\ \frac{1}{10}e^{x/10}(1+\cos(x)\sin(x^3)) + e^{x/10}(3x^2\cos(x)\cos(x^3)-\sin(x)\sin(x^3)) & \text{for } x<0 \end{cases}\end{aligned}$$

With the closed form of $g'(x)$ calculated, we can implement the Langevin algorithm with the following code

```
#Estimate E_pi(X+X^2) where pi=cg using Metropolis-Hasting algorithm where
# g(x) = e^(-|x|/10)*(1+cos(x)sin(x^3))
# with proposal N(X_(n-1)+1/2sigma^2g'(X_(n-1)),sigma^2)

sigma = 0.1
M =1e5
B = 2e4

fn = function(x) {x+x^2}
g = function(x) {exp(-abs(x)/10)*(1+cos(x)*sin(x^3))}
gp = function(x) {
  if (x>=0)
    -exp(-x/10)/10*(1+cos(x)*sin(x^3))+exp(-x/10)*(3*x^2*cos(x)*cos(x^3)-sin(x)*sin(x^3))
  else
    exp(x/10)/10*(1+cos(x)*sin(x^3))+exp(x/10)*(3*x^2*cos(x)*cos(x^3)-sin(x)*sin(x^3))
}
qq = function(x,y) {
  exp(-(y-x-sigma^2*gp(x)/g(x)/2)^2/sigma^2/2)
}

#initialization
fnlist = numeric(M+B)
xlist = numeric(M+B)
x = rnorm(1,0,sigma)
acc = 0

for (i in 1:(M+B)) {

  m = 1/2*sigma^2*gp(x)/g(x)
  y = rnorm(1,x+m,sigma)

  if (runif(1)<=g(y)*qq(y,x)/g(x)/qq(x,y)) {
    if (i>B)
      acc=acc+1
    x = y
  }
  xlist[i] = x
```

```
  fnlist[i] = fn(x)
}

funcmean = mean(fnlist[(B+1):(M+B)])
funciidse = sd(fnlist[(B+1):(M+B)])/sqrt(M)
acf_k = acf(fnlist[(B+1):(M+B)],lag.max = 1000,plot = FALSE)$acf
varfact = 2*sum(acf_k)-1
funcse = funciidse*sqrt(varfact)
accrate = acc/M

cat('B = ',B,', M = ',M,'\n')
cat('Number of samples accepted = ',acc,', acceptance rate = ',accrate,'\n')
cat('Estimate = ',funcmean,'\n')
cat('i.i.d. standard error = ',funciidse,'\n')
cat('varfact = ',varfact,'\n')
cat('Standard error = ',funcse,'\n')
```

Output of several runs:

```
## B =  20000 , M =  1e+05
## Number of samples accepted =  72892 , acceptance rate =  0.72892
## Estimate =  16.37651
## i.i.d. standard error =  0.07014971
## varfact =  1875.972
## Standard error =  3.038359
## B =  20000 , M =  1e+05
## Number of samples accepted =  66806 , acceptance rate =  0.66806
## Estimate =  51.28206
## i.i.d. standard error =  0.2649013
## varfact =  1958.284
## Standard error =  11.72255
## B =  20000 , M =  1e+05
## Number of samples accepted =  55956 , acceptance rate =  0.55956
## Estimate =  23.24639
## i.i.d. standard error =  0.06920837
## varfact =  1798.478
## Standard error =  2.935021
## B =  20000 , M =  1e+05
## Number of samples accepted =  64279 , acceptance rate =  0.64279
## Estimate =  18.73548
## i.i.d. standard error =  0.06656209
## varfact =  1821.751
## Standard error =  2.841001
## B =  20000 , M =  1e+05
## Number of samples accepted =  76744 , acceptance rate =  0.76744
## Estimate =  8.867096
## i.i.d. standard error =  0.03519465
## varfact =  1715.128
## Standard error =  1.457555
```

The tabulated result is as follows:

estimation	acc_rate	seiid	varfact_r	standard_error
16.376511	0.72892	0.0701497	1875.972	3.038359
51.282063	0.66806	0.2649013	1958.284	11.722547
23.246395	0.55956	0.0692084	1798.478	2.935021
18.735479	0.64279	0.0665621	1821.751	2.841001
8.867096	0.76744	0.0351946	1715.128	1.457555

We notice that the varfact values are quite small, indicating that the samples are not strongly correlated. To verify the effectiveness of the sampler, here we plot the value of x after the burn-in stage

```
xlist8b = xlist
plot(xlist8b[(B+1):(B+M)],type='l')
```

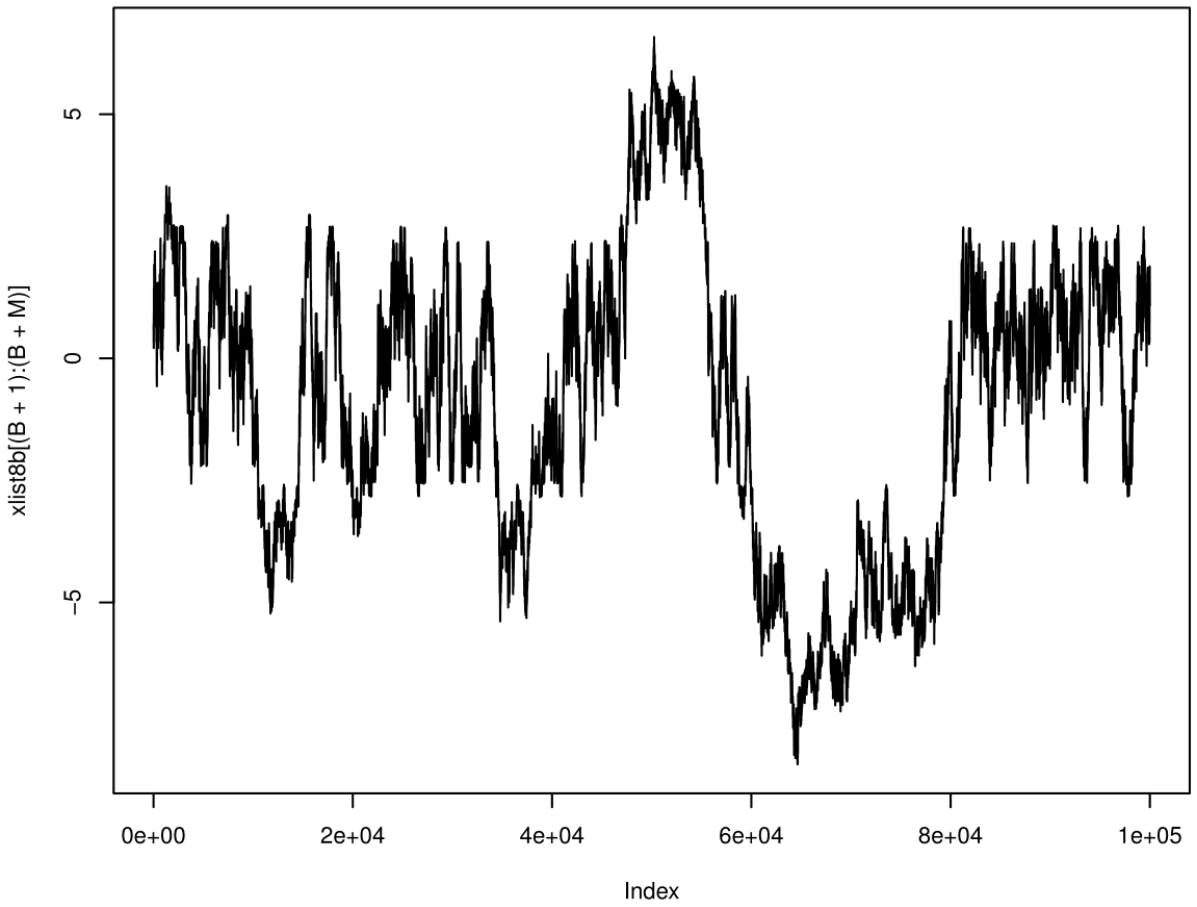


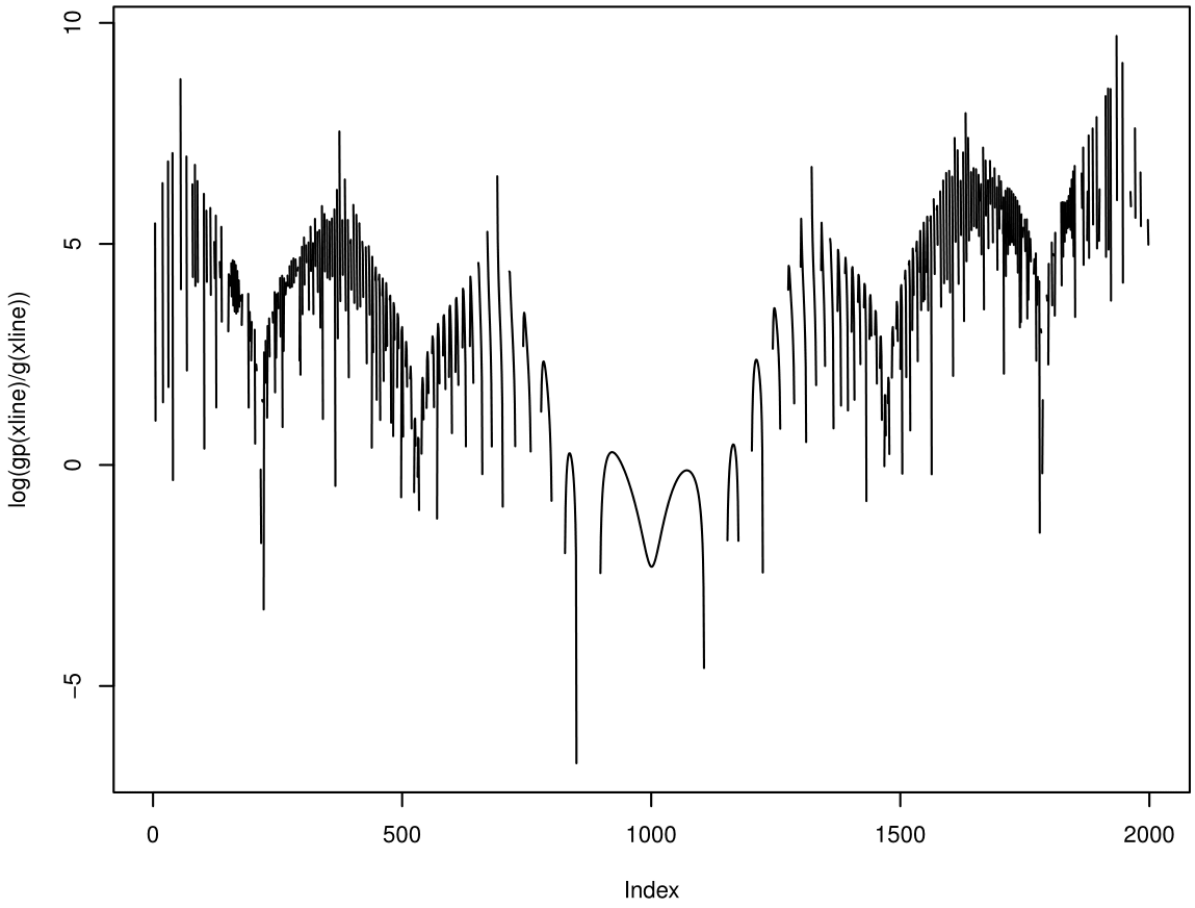
Figure 3: x values of Langevin algorithm after burning-in

Notice that the estimation is by no means consistent, and the varfact values are huge, making the standar error large too.

(c) Compare the two algorithms and discuss which one is "better".
In this case, the random-walk Metropolis algorithm clearly outperforms the Langevin algorithm as it produces more reasonable estimates to the integral. The Langevin algorithm is designed to work well for well behaved functions. In particular, if the function $g'(x)/g(x)$ is smooth, then the Langevin algorithm is going to outperform RWM algorithm. However, if we plot $g'(x)/g(x)$, we notice that it varies significantly even on a log-scaled plot

```
xline = seq(-10,10,0.01)
plot(log(gp(xline)/g(xline)),type='l')
```

```
## Warning in if (x >= 0) -exp(-x/10)/10 * (1 + cos(x) * sin(x^3)) +
exp(-x/10) * : the condition has length > 1 and only the first element will
be used
## Warning in log(gp(xline)/g(xline)): NaNs produced
```



Since to the function $g'(x)/g(x)$ is unstable, a discrete approximation of this function will fail. As a result, the RWM algorithm outperforms the Langevin algorithm significantly.

Problem 9

Consider the standard variance component model described in lecture, with $K = 6$, $J_i = 5$ and $\{Y_{ij}\}$ the famous "dyestuff" date, with prior values $a_1 = a_2 = a_3 = b_1 = b_2 = b_3 = 100$. Estimate the posterior mean of W/V, in each of the three ways:
(a) With a random-walk Metropolis algorithm. Since we are dealing with likelihood functions, we need to compute things in log-scale. The ratio of distributions becomes

$$\log\left(\frac{g(Y)}{g(X)}\right) = \log g(Y) - \log g(X)$$

The initial θ_i values are reasonably assumed to follow a normal distribution centered at sample mean of Y_{ij} with V, W equal to the sample variance of Y_{ij}.
The RWM algorithm is implemented as follows:

```
a=100
K=6
J=5

B=2e+6
M=5e+5
sigma = 7

Ydye = t( matrix(
  c(1545, 1440, 1440, 1520, 1580,
    1540, 1555, 1490, 1560, 1495,
    1595, 1550, 1605, 1510, 1560,
    1445, 1440, 1595, 1465, 1545,
    1595, 1630, 1515, 1635, 1625,
    1520, 1455, 1450, 1480, 1445), nrow=5) )

logg = function(m,V,W,theta,Ydye) {
  g = -a/V + log(max(c(V,1e-5)))*(-a-1) + (-a/W)
  g = g + log(max(c(W,1e-5)))*(-a-1) - (m-a)^2/a/2
  g = g + log(max(c(V,1e-5)))*(-K/2) + log(max(c(W,1e-5)))*(-K*J/2)
  g = g - sum((theta-m)^2)/V/2 - sum((Ydye-theta)^2)/W/2
  return (g)
}

m = rnorm(1,a,a)
V = var(c(Ydye))
W = var(c(Ydye))
theta = rnorm(K,mean(Ydye),sqrt(V))
acc = 0
fnlist = numeric(B+M)

for (i in 1:(B+M)) {
  eps_m = rnorm(1,0,sigma)
  eps_V = rnorm(1,0,sigma)
  eps_W = rnorm(1,0,sigma)
  eps_theta = rnorm(K,0,sigma)
```

```
  dg = logg(m+eps_m,V+eps_V,W+eps_W,theta+eps_theta,Ydye) -
    logg(m,V,W,theta,Ydye)
  if (runif(1)<=exp(dg)) {
    if (i>B)
      acc = acc + 1
    m = m+eps_m
    V = V+eps_V
    W = W+eps_W
    theta = theta+eps_theta
  }
  fnlist[i] = W/V
}
funcmean = mean(fnlist[(B+1):(M+B)])
funciidse = sd(fnlist[(B+1):(M+B)])/sqrt(M)
acf_k = acf(fnlist[(B+1):(M+B)],lag.max = 1000,plot = FALSE)$acf
varfact = 2*sum(acf_k)-1
funcse = funciidse*sqrt(varfact)
accrate = acc/M

cat('B = ',B,', M = ',M,'\n')
cat('Number of samples accepted = ',acc,', acceptance rate = ',accrate,'\n')
cat('Estimate = ',funcmean,'\n')
cat('i.i.d. standard error = ',funciidse,'\n')
cat('varfact = ',varfact,'\n')
cat('Standard error = ',funcse,'\n')
```

Output of several runs:

```
## B =  2e+06 , M =  5e+05
## Number of samples accepted =  127952 , acceptance rate =  0.255904
## Estimate =  0.006394276
## i.i.d. standard error =  8.691605e-07
## varfact =  222.6512
## Standard error =  1.296918e-05
## B =  2e+06 , M =  5e+05
## Number of samples accepted =  127690 , acceptance rate =  0.25538
## Estimate =  0.005920185
## i.i.d. standard error =  8.150501e-07
## varfact =  259.4071
## Standard error =  1.312729e-05
## B =  2e+06 , M =  5e+05
## Number of samples accepted =  127896 , acceptance rate =  0.255792
## Estimate =  0.006361079
## i.i.d. standard error =  8.695236e-07
## varfact =  242.357
## Standard error =  1.353659e-05
## B =  2e+06 , M =  5e+05
## Number of samples accepted =  127489 , acceptance rate =  0.254978
## Estimate =  0.006515764
```

```
## i.i.d. standard error =  9.252959e-07
## varfact =  382.6875
## Standard error =  1.810101e-05
## B =  2e+06 , M =  5e+05
## Number of samples accepted =  128078 , acceptance rate =  0.256156
## Estimate =  0.005825482
## i.i.d. standard error =  8.563688e-07
## varfact =  477.4472
## Standard error =  1.871214e-05
```

The tabulated result is as follows:

estimation	acc_rate	seiid	varfact_r	standard_error
0.0063943	0.255904	9e-07	222.6512	1.30e-05
0.0059202	0.255380	8e-07	259.4071	1.31e-05
0.0063611	0.255792	9e-07	242.3570	1.35e-05
0.0065158	0.254978	9e-07	382.6875	1.81e-05
0.0058255	0.256156	9e-07	477.4472	1.87e-05

Problem 9 continued on next page...

```
plot(fnlist[(B+1):(B+M)],type='l')
```

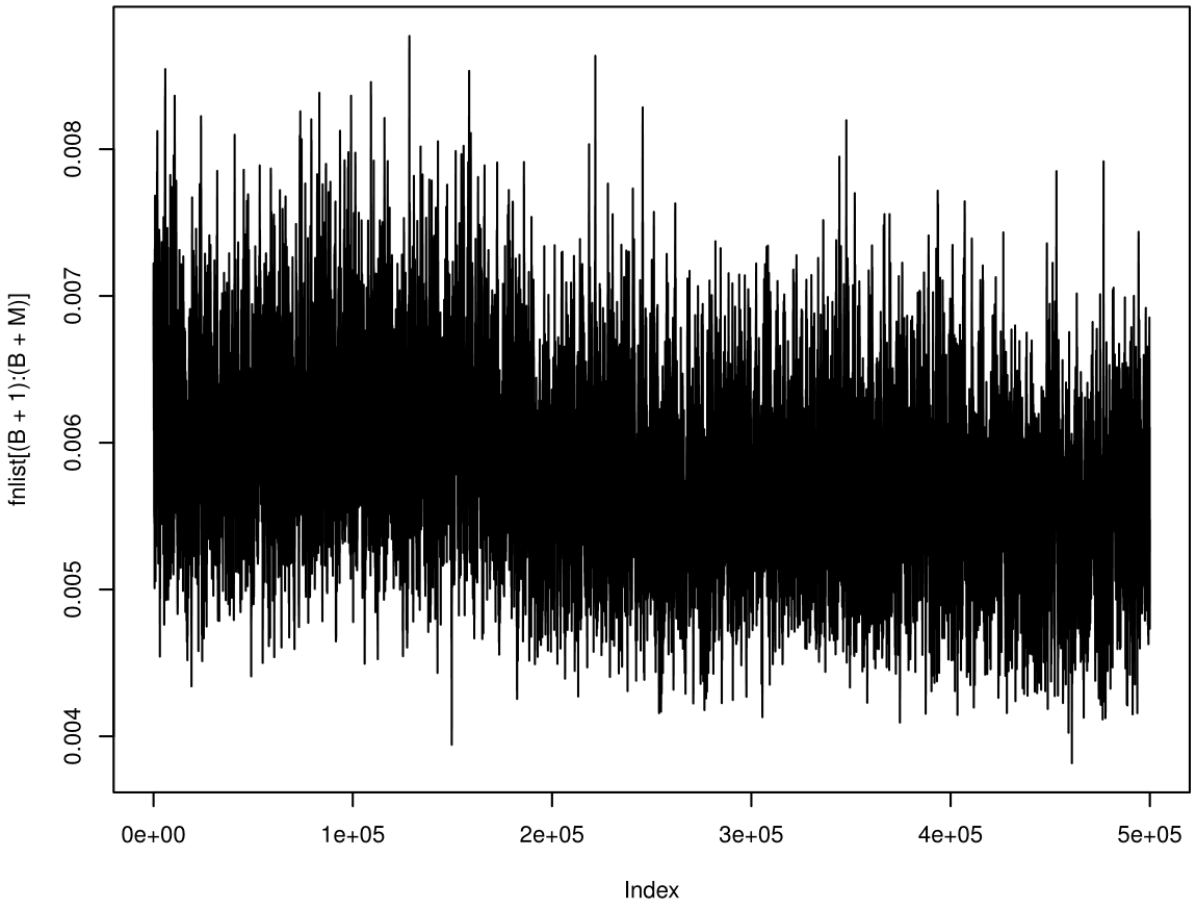


Figure 4: function values of Random-Walk Metropolis algorithm after burning-in

(b) With a Metropolis-within-Gibbs algorithm. Using a Metropolis-within-Gibbs algorithem, we can safely increase σ. Source code:

```
a=100
K=6
J=5

B=1e+6
M=5e+5
sigma = 40

Ydye = t( matrix(
  c(1545, 1440, 1440, 1520, 1580,
    1540, 1555, 1490, 1560, 1495,
    1595, 1550, 1605, 1510, 1560,
```

```
    1445, 1440, 1595, 1465, 1545,
    1595, 1630, 1515, 1635, 1625,
    1520, 1455, 1450, 1480, 1445), nrow=5) )

logg = function(m,V,W,theta,Ydye) {
  g = -a/V + log(max(c(V,1e-5)))*(-a-1) + (-a/W)
  g = g + log(max(c(W,1e-5)))*(-a-1) - (m-a)^2/a/2
  g = g + log(max(c(V,1e-5)))*(-K/2) + log(max(c(W,1e-5)))*(-K*J/2)
  g = g - sum((theta-m)^2)/V/2 - sum((Ydye-theta)^2)/W/2
  return (g)
}

m = rnorm(1,a,sigma)
V = var(c(Ydye))
W = var(c(Ydye))
theta = rnorm(K,mean(Ydye),sqrt(V))
acc = 0
fnlist = numeric(B+M)

for (i in 1:(B+M)) {
  eps_m=0
  eps_V=0
  eps_W=0
  eps_theta = rep(0,K)
  ci = i %% (3+K)
  if (ci==0)
    eps_m = rnorm(1,0,sigma)
  else if (ci==1)
    eps_V = rnorm(1,0,sigma)
  else if (ci==2)
    eps_W = rnorm(1,0,sigma)
  else
    eps_theta[ci-2] = rnorm(1,0,sigma)

  dg = logg(m+eps_m,V+eps_V,W+eps_W,theta+eps_theta,Ydye) -
    logg(m,V,W,theta,Ydye)
  if (runif(1)<=exp(dg)) {
    if (i>B)
    acc = acc + 1
    m = m+eps_m
    V = V+eps_V
    W = W+eps_W
    theta = theta+eps_theta
  }
  fnlist[i] = W/V
}
funcmean = mean(fnlist[(B+1):(M+B)])
funciidse = sd(fnlist[(B+1):(M+B)])/sqrt(M)
acf_k = acf(fnlist[(B+1):(M+B)],lag.max = 1000,plot = FALSE)$acf
```

```
varfact = 2*sum(acf_k)-1
funcse = funciidse*sqrt(varfact)
accrate = acc/M

cat('B = ',B,', M = ',M,'\n')
cat('Number of samples accepted = ',acc,', acceptance rate = ',accrate,'\n')
cat('Estimate = ',funcmean,'\n')
cat('i.i.d. standard error = ',funciidse,'\n')
cat('varfact = ',varfact,'\n')
cat('Standard error = ',funcse,'\n')
```

Output of several runs:

```
## B =  1e+06 , M =  5e+05
## Number of samples accepted =  177717 , acceptance rate =  0.355434
## Estimate =  0.005133218
## i.i.d. standard error =  7.382176e-07
## varfact =  298.7258
## Standard error =  1.275912e-05
## B =  1e+06 , M =  5e+05
## Number of samples accepted =  177155 , acceptance rate =  0.35431
## Estimate =  0.004777328
## i.i.d. standard error =  7.062485e-07
## varfact =  357.6378
## Standard error =  1.335609e-05
## B =  1e+06 , M =  5e+05
## Number of samples accepted =  177056 , acceptance rate =  0.354112
## Estimate =  0.004697013
## i.i.d. standard error =  6.735418e-07
## varfact =  305.6105
## Standard error =  1.177467e-05
## B =  1e+06 , M =  5e+05
## Number of samples accepted =  177643 , acceptance rate =  0.355286
## Estimate =  0.004896768
## i.i.d. standard error =  8.344297e-07
## varfact =  779.2573
## Standard error =  2.329326e-05
## B =  1e+06 , M =  5e+05
## Number of samples accepted =  177505 , acceptance rate =  0.35501
## Estimate =  0.004181851
## i.i.d. standard error =  6.293576e-07
## varfact =  411.9449
## Standard error =  1.277371e-05
```

The tabulated result is as follows:

Problem 9 continued on next page...

estimation	acc_rate	seiid	varfact_r	standard_error
0.0051332	0.355434	7e-07	298.7258	1.28e-05
0.0047773	0.354310	7e-07	357.6378	1.34e-05
0.0046970	0.354112	7e-07	305.6105	1.18e-05
0.0048968	0.355286	8e-07	779.2573	2.33e-05
0.0041819	0.355010	6e-07	411.9449	1.28e-05

```
plot(fnlist[(B+1):(B+M)],type='l')
```

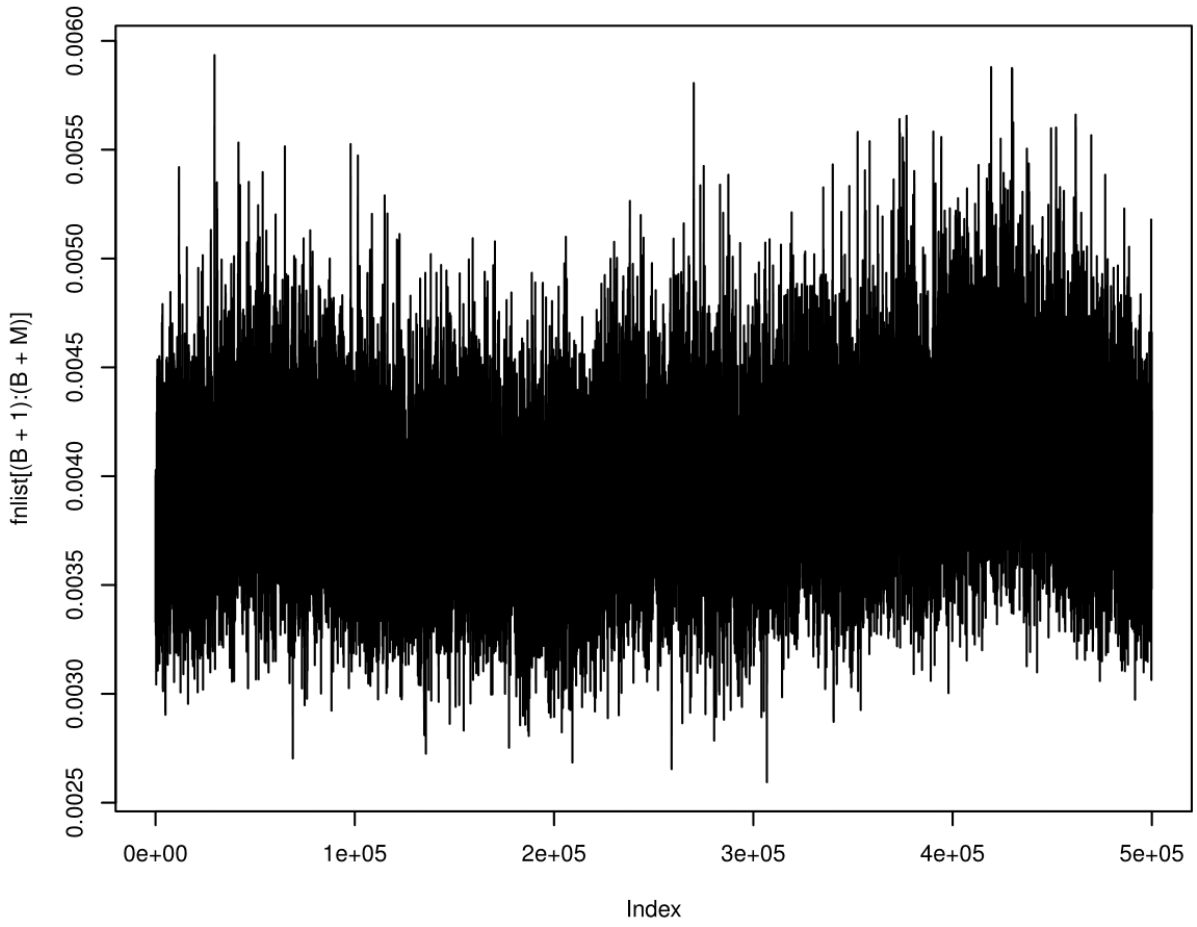


Figure 5: function values of Metropolis-within-Gibbs algorithm after burning-in

(c) With a Gibbs sampler.
For Gibbs sampler, we need to calculate the conditional distributions in closed form.

Firstly, the joint distribution is given by

$$\begin{aligned}
f(V,W,\mu,\theta,Y) =& Ce^{-b_1/V}V^{-a_1-1}e^{-b_2/W}W^{-a_2-1}e^{-(\mu-a3)^2/2b_3}V^{-K/2}W^{-\frac{1}{2}KJ}\times \\
& \exp\left[-\sum_{i=1}^{K}(\theta_i-\mu)^2/2V-\sum_{i=1}^{K}\sum_{j=1}^{J}(Y_{ij}-\theta_i)^2/2W\right] \\
=& Ce^{-a/V}V^{-a-1}e^{-a/W}W^{-a-1}e^{-(\mu-a)^2/2a}V^{-K/2}W^{-\frac{1}{2}KJ}\times \\
& \exp\left[-\sum_{i=1}^{K}(\theta_i-\mu)^2/2V-\sum_{i=1}^{K}\sum_{j=1}^{J}(Y_{ij}-\theta_i)^2/2W\right]
\end{aligned}$$

where C is a normalizing constant, and $a = a_1 = a_2 = a_3 = b_1 = b_2 = b_3$.
Now, we can write the following conditional distributions:

$$\begin{aligned}
f(\mu|V,W,\theta) &= C\exp\left[-\frac{(\mu-a)^2}{2a}-\sum_{i=1}^{K}\frac{(\theta_i-\mu)^2}{2V}\right] \\
&= C\exp\left[-\mu^2\left(\frac{1}{2a}+\frac{K}{2V}\right)+\mu\left(1+\sum_{i=1}^{K}\frac{\theta_i}{V}\right)\right]
\end{aligned}$$

If we let $\mu|*$ to follow a $N(m,v)$ distribution for some m and v, and match the coefficients to get

$$\begin{aligned}
v &= \left(\frac{1}{2a}+\frac{K}{2V}\right)^{-1}\Big/2 = \left(\frac{1}{a}+\frac{K}{V}\right)^{-1} \\
m &= \left(1+\sum_{i=1}^{K}\frac{\theta_i}{V}\right)\left(\frac{1}{a}+\frac{K}{V}\right)^{-1}
\end{aligned}$$

Therefore, we have

$$(\mu|V,W,\theta) \sim N\left(\left(1+\sum_{i=1}^{K}\frac{\theta_i}{V}\right)\left(\frac{1}{a}+\frac{K}{V}\right)^{-1},\left(\frac{1}{a}+\frac{K}{V}\right)^{-1}\right)$$

$$\begin{aligned}
f(V|\mu,W,\theta) &= Ce^{-a/V}V^{-a-1}V^{-K/2}\exp\left(-\sum_{i=1}^{K}\frac{(\theta_i-\mu)^2}{2V}\right) \\
&= CV^{-a-1-K/2}\exp\left(-\frac{1}{V}\left[a+\sum_{i=1}^{K}\frac{(\theta_i-\mu)^2}{2}\right]\right)
\end{aligned}$$

Notice that this is the pdf of a Inverse Gamma distribution with parameters $\alpha = a + K/2$ and $\beta = a + \sum_{i=1}^{K}(\theta_i-\mu)^2/2$, i.e. we have

$$(V|\mu,W,\theta) \sim IG\left(a+\frac{K}{2},a+\sum_{i=1}^{K}\frac{(\theta_i-\mu)^2}{2}\right)$$

Similarly,

$$\begin{aligned}
f(W|\mu,V,\theta) &= Ce^{-a/W}W^{-a-1}W^{-KJ/2}\exp\left(-\frac{1}{W}\sum_{i=1}^{K}\sum_{j=1}^{J}\frac{(Y_{ij}-\theta_i)^2}{2}\right) \\
&= CW^{-a-1-KJ/2}\exp\left(-\frac{1}{W}\left[a+\sum_{i=1}^{K}\sum_{j=1}^{J}\frac{(Y_{ij}-\theta_i)^2}{2}\right]\right)
\end{aligned}$$

Problem 9 continued on next page...

Again, this is the pdf of Inverse Gamma distribution with parameters $\alpha = a + KJ/2$ and $\beta = a + \sum_{i=1}^{K}\sum_{j=1}^{J}(Y_{ij} - \theta_i)^2/2$. i.e.

$$(W|\mu, V, \theta) \sim IG\left(a + KJ/2, a + \sum_{i=1}^{K}\sum_{j=1}^{J}\frac{(Y_{ij} - \theta_i)^2}{2}\right)$$

Lastly,

$$\begin{aligned} f(\theta_i|\mu, V, W) &= C\exp\left(-\frac{(\theta_i - \mu)^2}{2V} - \sum_{j=1}^{J}\frac{(Y_{ij} - \theta_i)^2}{2W}\right) \\ &= C\exp\left(-\theta_i^2\left(\frac{1}{2V} + \frac{J}{2W}\right) + \theta_i\left(\frac{\mu}{V} + \sum_{j=1}^{J}\frac{Y_{ij}}{W}\right)\right) \end{aligned}$$

If we let $\mu|*$ to follow a $N(m, v)$ distribution for some m and v, and match the coefficients to get

$$v = \left(\frac{1}{2V} + \frac{J}{2W}\right)^{-1}\Big/2 = \left(\frac{1}{V} + \frac{J}{W}\right)^{-1}$$

$$m = \left(\frac{\mu}{V} + \sum_{j=1}^{J}\frac{Y_{ij}}{W}\right)\left(\frac{1}{V} + \frac{J}{W}\right)^{-1}$$

i.e. we have

$$(\theta_i|\mu, V, W) \sim N\left(\left(\frac{\mu}{V} + \sum_{j=1}^{J}\frac{Y_{ij}}{W}\right)\left(\frac{1}{V} + \frac{J}{W}\right)^{-1}, \left(\frac{1}{V} + \frac{J}{W}\right)^{-1}\right)$$

Finally, using the closed form conditional distribution above, we can implement the Gibbs sampler as follows:

```
a=100
K=6
J=5

B=1e+5
M=5e+5
sigma = 20

Ydye = t( matrix(
  c(1545, 1440, 1440, 1520, 1580,
    1540, 1555, 1490, 1560, 1495,
    1595, 1550, 1605, 1510, 1560,
    1445, 1440, 1595, 1465, 1545,
    1595, 1630, 1515, 1635, 1625,
    1520, 1455, 1450, 1480, 1445), nrow=5) )

logg = function(m,V,W,theta,Ydye) {
  g = -a/V + log(max(c(V,1e-5)))*(-a-1) + (-a/W)
  g = g + log(max(c(W,1e-5)))*(-a-1) - (m-a)^2/a/2
  g = g + log(max(c(V,1e-5)))*(-K/2) + log(max(c(W,1e-5)))*(-K*J/2)
  g = g - sum((theta-m)^2)/V/2 - sum((Ydye-theta)^2)/W/2
  return (g)
```

```
}

m = rnorm(1,a,sigma)
V = var(c(Ydye))
W = var(c(Ydye))
theta = rnorm(K,mean(Ydye),sqrt(V))
acc = 0
fnlist = numeric(B+M)

for (i in 1:(B+M)) {
  eps_m=0
  eps_V=0
  eps_W=0
  eps_theta = rep(0,K)
  ci = i %% (3+K)
  if (ci==0)
    m = rnorm(1,1+sum(theta)/V/(1/a+K/V),(1/a + K/V)^(-0.5))
  else if (ci==1)
    V = 1/rgamma(1,a+K/2,a+sum((theta-m)^2)/2)
  else if (ci==2)
    W = 1/rgamma(1,a+K*J/2, a+sum((Ydye-theta)^2)/2)
  else
    theta[ci-2] = rnorm(1,(m/V+sum(Ydye[ci-2,])/W)/(1/V+J/W),
                        (1/V+J/W)^(-0.5))

  fnlist[i] = W/V
}
funcmean = mean(fnlist[(B+1):(M+B)])
funciidse = sd(fnlist[(B+1):(M+B)])/sqrt(M)
acf_k = acf(fnlist[(B+1):(M+B)],lag.max = 1000,plot = FALSE)$acf
varfact = 2*sum(acf_k)-1
funcse = funciidse*sqrt(varfact)
accrate = acc/M

cat('B = ',B,', M = ',M,'\n')
cat('Number of samples accepted = ',acc,', acceptance rate = ',accrate,'\n')
cat('Estimate = ',funcmean,'\n')
cat('i.i.d. standard error = ',funciidse,'\n')
cat('varfact = ',varfact,'\n')
cat('Standard error = ',funcse,'\n')
```

Output of several runs:

```
## B =  1e+05 , M =  5e+05
## Number of samples accepted =  0 , acceptance rate =  0
## Estimate =  0.003992738
## i.i.d. standard error =  7.8035e-07
## varfact =  11.25486
## Standard error =  2.617939e-06
```

```
## B =  1e+05 , M =  5e+05
## Number of samples accepted =  0 , acceptance rate =  0
## Estimate =  0.00399849
## i.i.d. standard error =  7.796216e-07
## varfact =  10.92722
## Standard error =  2.577144e-06
## B =  1e+05 , M =  5e+05
## Number of samples accepted =  0 , acceptance rate =  0
## Estimate =  0.003996011
## i.i.d. standard error =  7.800165e-07
## varfact =  8.801791
## Standard error =  2.314138e-06
## B =  1e+05 , M =  5e+05
## Number of samples accepted =  0 , acceptance rate =  0
## Estimate =  0.003992633
## i.i.d. standard error =  7.772488e-07
## varfact =  10.07245
## Standard error =  2.466764e-06
## B =  1e+05 , M =  5e+05
## Number of samples accepted =  0 , acceptance rate =  0
## Estimate =  0.003994966
## i.i.d. standard error =  7.799505e-07
## varfact =  10.16798
## Standard error =  2.487049e-06
```

The tabulated result is as follows:

estimation	seiid	varfact_r	standard_error
0.0039927	8e-07	11.254861	2.6e-06
0.0039985	8e-07	10.927216	2.6e-06
0.0039960	8e-07	8.801791	2.3e-06
0.0039926	8e-07	10.072450	2.5e-06
0.0039950	8e-07	10.167975	2.5e-06

Problem 9 continued on next page...

```
plot(fnlist[(B+1):(B+M)],type='l')
```

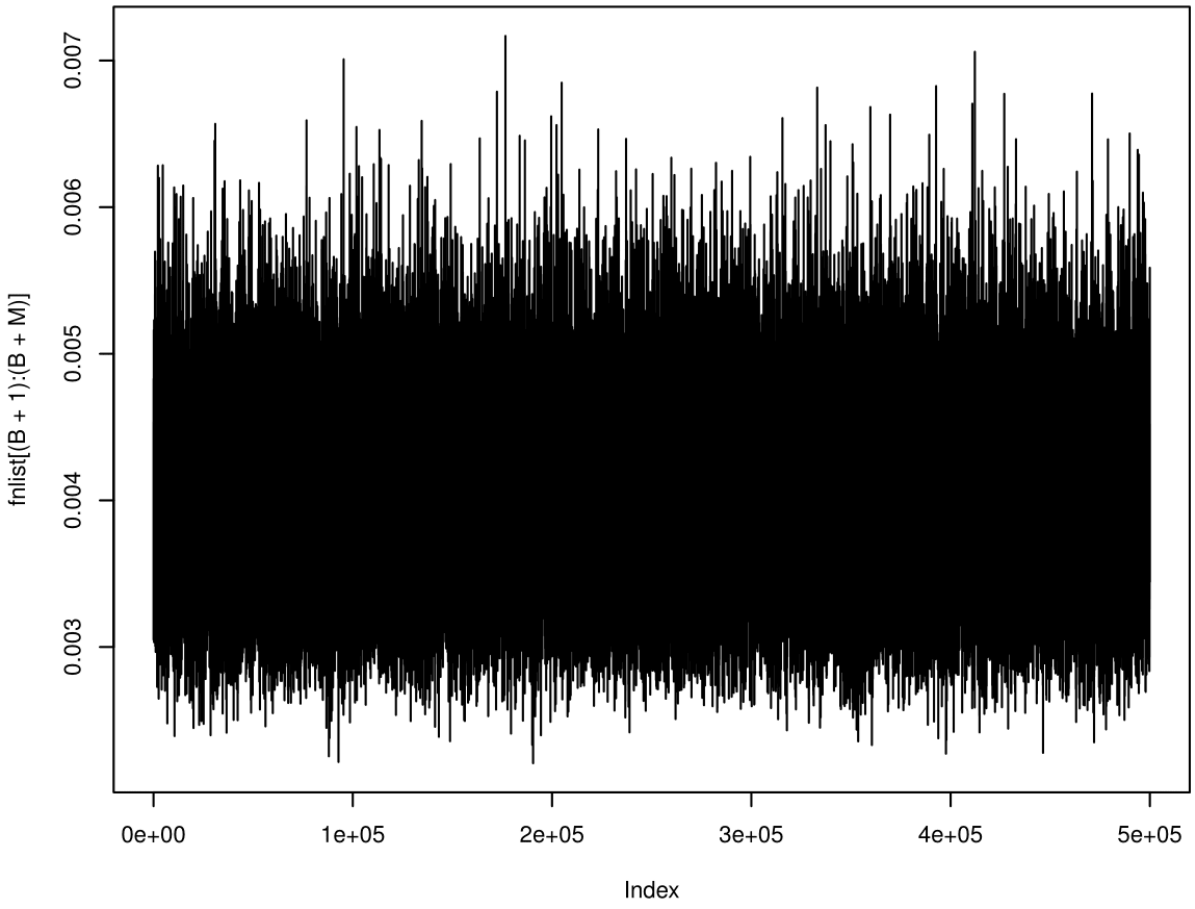


Figure 6: function values of Gibbs sampler after burning-in

(d) In terms of performance, the Gibbs sampler performs the best among the three as it requires the least amount of burning-in. The random-walk Metropolis requires 2e+6 steps just to burn-in to consistently generates non-trending function values (The function estimates tend to decrease over time if not burnt-in properly). This is quite a waste of computation power. The Metropolis-within-Gibbs requires similar number of steps to burn-in. On the contrary, the Gibbs sampler algorithm requires less than 1e+5 cycles to burn-in, which is magnitudes less than the required burning-in cycles for the other two algorithms.
The Gibbs sampler algorithm also performs better in terms of standard error since they produce low varfact values, meaning that the samples generated from the Gibbs sampler algorithm are close to i.i.d. samples.
However, the Gibbs sampler requires us to compute the conditional distribution in closed form, which may be very tedious and often impossible. The other two algorithms do not require analytic computation (choosing conjugate priors), hence they are much more easily implemented.
The Metropolis-within-Gibbs algorithm converges faster when comparing with the random-walk Metropolis algorithm. This is because the RWM proposes points that differs in all dimensions and will cause a higher chance of rejection.
Despite all these disadvantages, there are some advantages to the Random-Walk Metropolis algorithm. For

example, if the distribution is defined on two regions that are disconnected in more than one dimension, then the Gibbs-like algorithms will not be able to travel from one region to the other.

Problem 10

Term Structure of Interest Rates

According to definition on INVESTOPEDIA, "the term structure of interest rates is the relationship between interest rates or bond yields and different terms or maturities. The term structure of interest rates is also known as a yield curve and it plays central role in an economy. The term structure reflects expectations of market participants about future changes in interest rates and their assessment of monetory policy conditions"
In mathematical finance, the term structure of interest rates is often modeled using stochastic methods, which can pose a number of difficult problems for estimation. One of the most crucial problems is that the parameters enter the state space in a highly nonlinear fasion. However, solving these non-linear systems are often quite rewarding as we can derive bonds and their derivatives prices from the models, yet, most of the time, we can only use MCMC methods to help solving the system.

Vasicek's Model With Jumps

The Vasicek's model assumes that the instantaneous spot rate is a Gaussian diffusion:

$$dr_t = \kappa_r(\theta_r - r_t)dt + \sigma_r dW_t^r$$
$$dr_t = (a_r + b_r r_t)dt + \sigma_r dW_t^r$$

where W_t^r is a Brownian motion under the objective measure, P, and we will work with the (a_r, b_r) parameterization of the drift.
Baz and Das (1996) consider an extension of Vasicek's model to incorporate jumps in the short rate:

$$dr_t = (a_r - b_r r_{t-})dt + \sigma_r dW_t^r + d\left(\sum_{j=1}^{N_t} \xi_{\tau_j}\right)$$

where we assume that N_t is a Poisson process with constant intensity λ and the jumps sizes are i.i.d. normal $\xi_{\tau_j} \sim N(\mu_J, \sigma_J^2)$. To value bonds under this model, we assume that the market price of interest rate risk is $\lambda_t = \lambda_r\sqrt{r_t}$ and if we assume the diffusive risk premium is constant, that N_t^Q is a Poisson process under Q with constant intensity λ^Q and that the jump sizes are normally distributed under Q, $\xi_{\tau_j}^Q \sim N\left(\mu_j^Q, (\sigma_J^2)^Q\right)$, the evolution of the spot rate under Q is then

$$dr_t = \kappa_r(\theta_r^Q - r_t)dt + \sigma_r dW_t^r(Q) + d\left(\sum_{j=1}^{N_t^Q} \xi_{\tau_j}^Q\right)$$

where $W_t^r(Q)$ is a standard Brownian motion under the equivalent martingale measure Q. Then under this model, Using Feyman Kac's theorem, the price a non-defaultable zero coupon bond is given by

$$P(r_t, \tau) = E_t^Q\left[e^{-\int_t^{t+\tau} r_s ds}\right]$$

Problem 10 continued on next page...

Monte Carlo Simulation and Integration Implementation

In this mini-project, we assume the following parameter values

$$\begin{aligned}\tau &= 10\\ \theta_r^Q &= 0.04\\ r_t &= 0.015\\ \kappa_r &= 0.6\\ \sigma_r &= 0.03\\ \lambda^Q &= 2\\ \mu_J^Q &= 0\\ (\sigma_J^2)^Q &= 0.01\end{aligned}$$

For small h, at any instance t, the distribution of r_{t+h} given r_t follows

$$f(r_{t+h}|r_t) \sim N(r_t + \kappa_r(\theta_r^Q - r_t)h, (\sigma_r^2)^Q\sqrt{h} + Bernuolli(\lambda^Q h) * N\left(\mu_J^Q, (\sigma_J^2)^Q\right)$$

Using the distribution above, we simulate $M = 2000$ paths and use Monte Carlo integration techiniques to calculate the price of zero-coupon bond. Ten runs are executed to investigate the consistency of the estimate. Source code:

```
#calculate the price of zero coupon bonds with Vasicek
#with jump models

#financial parameters (all adjusted for risk neutral measure)
T = 10
theta = 0.04
cr = 0.015
kappa = 0.6
sigma = 0.03
lambda = 2
sigmaj = 0.01
muj = 0

#time interval set to be 0.01 to make 100 intervals per year
h=0.01
M = 2000
num = round(T/h)
vallist = rep(0,M)
plotX = matrix(nrow = 20, ncol = num+1)
for (j in 1:M) {
  X = rep(0,num+1)
  X[1] = cr
  for (i in 1:num) {
    if (runif(1)<lambda*h) {jump = rnorm(1,muj,sigmaj)}
    else jump = 0
    X[i+1] = rnorm(1,X[i]+kappa*(theta-X[i])*h,sigma*sqrt(h))+jump
    if (j<=20) plotX[j,] = X
```

```
  }
  vallist[j] = exp(-sum(X)*h)
}
price = mean(vallist)
se = sd(vallist)/sqrt(M)
cat('The estimated zero-coupon bond price is ',price,'\n')
cat('The standard error is ',se,'\n')
```

Output of several runs:

```
## The estimated zero-coupon bond price is  0.7043499
## The standard error is  0.002424588
## The estimated zero-coupon bond price is  0.706554
## The standard error is  0.002378813
## The estimated zero-coupon bond price is  0.7110355
## The standard error is  0.002479945
## The estimated zero-coupon bond price is  0.7031318
## The standard error is  0.002333143
## The estimated zero-coupon bond price is  0.7018515
## The standard error is  0.002379074
## The estimated zero-coupon bond price is  0.7057709
## The standard error is  0.002414406
## The estimated zero-coupon bond price is  0.7024047
## The standard error is  0.002393151
## The estimated zero-coupon bond price is  0.7087772
## The standard error is  0.002428283
## The estimated zero-coupon bond price is  0.7051398
## The standard error is  0.002356732
## The estimated zero-coupon bond price is  0.7117877
## The standard error is  0.00238904
```

The tabulated result is as follows:

estprice	estse
0.7043499	0.0024246
0.7065540	0.0023788
0.7110355	0.0024799
0.7031318	0.0023331
0.7018515	0.0023791
0.7057709	0.0024144
0.7024047	0.0023932
0.7087772	0.0024283
0.7051398	0.0023567
0.7117877	0.0023890

From the table above, we can see that the estimated price is fairly consistent with a mean of 0.7060803. The standard error of the price is also relatively small, the mean of the estimated standard error is 0.0023977. The standard error calculated from multiple runs of executing the program is 0.0034751. The small standar error of the estimated price means our MC estimation is reasonably accurate.

There are several advantages to this approach.
First of all, since there are multiple sources of stochasticity in our model (namely, the diffusive brownian motion, the number of jumps and the size jumps), solving the system analytically is (almost) impossible. In order to find a reasonable bond price under this model, we have to use MC simulation method.
Also, since we have simulated paths of realizations of the spot rate, we can also find out how risky the bond is. That is, if we invest the same amount in the bank account and follow the simulated spot rate paths, how much more or less we are going to make in comparison to investing in the zero-coupon bond.
Last but not least, we can look at some sample paths of the spot rate curve to see if our model assumption is valid. The following figure outlines 20 sample paths from our simulation

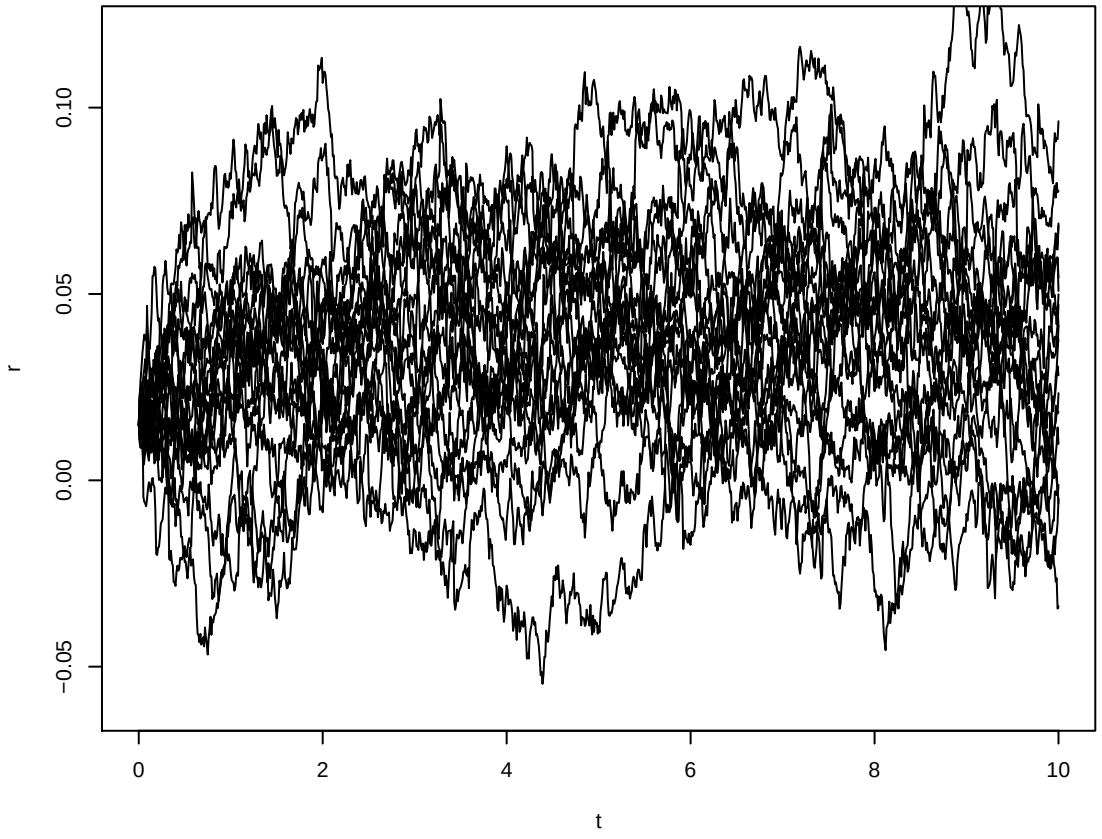


Figure 7: 20 simulated paths of the spot rate under risk neutral measure

From the figure above, we observe that the simulated paths follow typical strucure of actual observed spot rate curves. This validates to some extent that our model assumption is valid.

Further Research

The original plan was actually to work on more advanced Monte Carlo method to estimate zero-coupon bond prices under this model assumption. However, the paper I was reading on this subject was quite intense, and I did not fully understand the methods described in the paper (and marking midterm papers for other courses is taking way longer than I have expected). In the paper, the financial parameters are not assumed to

be known in advance, and the only given data in the implementation is the actual observed bond prices with different maturity. The spot rates and the parameters are assumed to follow a complicated joint distribution, so metroplis (or other sampling algorithms) have to be used to obtain the term structure curves. Here is the link to the referenced paper (`https://www0.gsb.columbia.edu/mygsb/faculty/research/pubfiles/564/MCMC.pdf`)