**

Exercise day 1 to day 5

02443: **Stochastic  Simulation,** June 2015

S111503

Ran Wang

Contents

[1. Random number generator 4](#_Toc423452507)

[1.1 Abstract: 4](#_Toc423452508)

[1.2 What is stochasticity (randomness)? 4](#_Toc423452509)

[1.3 How to produce random sequences with a computer? 4](#_Toc423452510)

[1.4 How to check that a computer produces random numbers? 4](#_Toc423452511)

[1.4.1 Histogram 4](#_Toc423452512)

[1.4.2 Auto-correlation function 4](#_Toc423452513)

[1.4.3 Formal statistical test: Kolmogorov-Smirnov (KS) test 5](#_Toc423452514)

[1.5 Repeat the previous steps with parameters a, b and M of your choice. 6](#_Toc423452515)

[2. Non-uniform Random Numbers 8](#_Toc423452516)

[2.1 Inversion method 8](#_Toc423452517)

[2.2 Rejection method for beta distribution 8](#_Toc423452518)

[2.3 Rejection method for double exponential distribution 9](#_Toc423452519)

[2.4 Needle throwing 10](#_Toc423452520)

[2.5 Car selling with bid x following Poisson distribution 12](#_Toc423452521)

[**2.6** Bootstrap**:** Variance in the estimation of the median of a Poisson distribution 12](#_Toc423452522)

[2.7 Bivariate mean and variance 13](#_Toc423452523)

[3. Markov chain Monte Carlo methods 15](#_Toc423452524)

[3.1 General Metropolis Algorithm 15](#_Toc423452525)

[3.2 Metropolis algorithm with the Cauchy distribution C(0; 1) 16](#_Toc423452526)

[**3.3** Metropolis-Hastings algorithm with Gaussian proposal 17](#_Toc423452527)

[3.4 Rejection algorithm and HM simulating Beta 19](#_Toc423452528)

[Rejection algorithm: 19](#_Toc423452529)

[MH algorithm: 20](#_Toc423452530)

[3.5 Autocorrelation 20](#_Toc423452531)

[3.6 Simulate MVN using Gibbs algorithm 21](#_Toc423452532)

[4. Monte Carlo integration 22](#_Toc423452533)

[4.1 22](#_Toc423452534)

[4.2 23](#_Toc423452535)

[5. Monte Carlo Optimization methods 24](#_Toc423452536)

[5.1 24](#_Toc423452537)

[5.1.1 Uniform simulation algorithm 25](#_Toc423452538)

[Pros and Cons of uniform algorithm: 25](#_Toc423452539)

[5.1.2 None uniform algorithm-Rejection method 25](#_Toc423452540)

[5.1.3 Comparison with R inbuilt “optimize” function 26](#_Toc423452541)

[5.2 Simulated annealing method to maximize h on [-3; 3]^2 26](#_Toc423452542)

[Annealing method finding the maximum follows these steps: 27](#_Toc423452543)

[Properties of annealing method (as the lecture notes mentioned): 28](#_Toc423452544)

[Appendix 30](#_Toc423452545)

Exercise 01

# 1. Random number generator

## 1.1 Abstract:

This report will devote to clarify the definition and implementation of stochastics with help of uniform distribution, probability density function (pdf), independence, complexity theory and RNG (Random Number Generator). It follows by several approaches for validation of the random numbers.

## 1.2 What is stochasticity (randomness)?

It is something that cannot be predicted, not intrinsic to the phenomenon, and not operational.

## 1.3 How to produce random sequences with a computer?

Since the complexity of a sequence may grow with the length of the sequence, if we expect an infinite sequence, it might require an algorithm with infinite length to produce it. However, this is not feasible with a computer. One way to realize it is to use pseudo Random Number Generator. One can make use of erratic distribution of prime numbers, there is no formula for the n-th prime. The algorithm is the linear congruential generator (LCG): X_{n+1} = \left( a X_n + c \right)~~\bmod~~m

Where X is the sequence of pseudorandom number, and m, a, c are integer constants:

 m,\, 0<m  – the "[modulus](http://en.wikipedia.org/wiki/Modulo_operation)"

 a,\,0 < a < m – the "multiplier"

 c,\,0 \le c < m – the "increment"

 X_0,\,0 \le X_0 < m – the "seed" or "start value"

Properties:

Algorithm is deterministic, periodic, producing numbers on the discrete set {0, 1/m, (m-1)/m}, fast, and efficiency depends strongly on a, b and m.

## How to check that a computer produces random numbers?

### 1.4.1 Histogram

The histogram should be close to the density of a uniform distribution U([0; 1]). A uniform distribution, sometimes also known as a rectangular distribution, is a distribution that has constant probability. The density is f(x) = 1/(max-min)=1. In figure 1 upper left the histogram is very close to the red line that is on density 1.

### 1.4.2 Auto-correlation function

p(h) = c(h)=Var (X)

where c(h) = Cov(Xk ; Xk+h) = = E(X; Xk+h) - E(X) E(Xk+h)

Should be as small as possible for h > 0.

In figure 1 the upper right scatter plot and bottom left line plot (the later just connect all the scatters to generate lines) appears no trend, so the first impress suggests that they might be independent and random. are In figure 1 Bottom right shows the auto-correlation function plot, where there is only one significant correlation at lag 0, the rest spikes are inside the 5% significance limits, and it appears no patterns at all.

In this example h=2, Cov(X; Xk+2) = -0.0007267802 is extremely small, which follows that E(X; Xk+2) - E(X) E(Xk+2) =0, So the estimated random numbers are most likely independent.

### 1.4.3 Formal statistical test: Kolmogorov-Smirnov (KS) test

KS test offers a quantitative method to evaluate the equality of two distributions. A hypothesis test with H0: there is no significant difference between theoretical and empirical cumulative distribution function |F(x) - F\*(x)|, which means that xi independently draws from a uniform distribution U([0; 1]).

A small p-value means that we have to reject H0.

A big p-value means that we failed to reject H0.

In this example, when a = 16807, b = 0, m =2147483647, the p-value is 0.04975. so we infer that there is conflict between theoretical and empirical cumulative distribution.

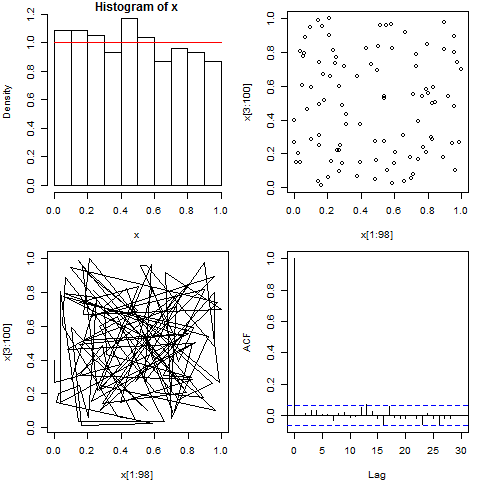


Figure 1 upper left: histogram with density; upper right: scatterplot; bottom left: random line; bottom right: auto correlation function. a = 16807, b = 0, m =2147483647

## 1.5 Repeat the previous steps with parameters a, b and M of your choice.

As the current p-value does not indicate the absence of conflict between theoretical and empirical cumulative distribution, I have to find a proper K, which is large enough to avoid discretization issues.

When I choose a = 31; b =1, m =3189, the RNG generates a sequence that passes all the aforementioned randomness checks (see figure 2), and p-value = 0.1922. Particularly, the histogram is continuous meaning no discretization, which is because the m is large enough to produce at most 3189 different random values between 0 and 1. On the other hand, if I chose a very small m, say m=10, the RNG will produce at most 10 different random values between 0 and 1, thus discretization will occur.

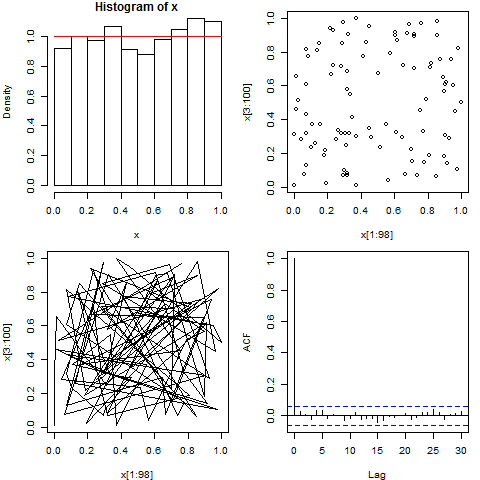


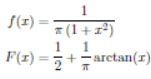
Figure 2 upper left: histogram with density; upper right: scatterplot; bottom left: random line; bottom right: auto correlation function. a = 31; b =1, m =3189.

Exercise 02

# 2. Non-uniform Random Numbers

## 2.1 Inversion method

The Cauchy(0,1) probability density distribution and cumulative distribution functions are given by:



Since the cdf of the target distribution is easy to invert, we have the invert function:



We first generate n=1000 random numbers uniformly in [0,1] (u=runif(n,0,1), then we plug this u into the invert function. The samples from the empirical distribution generated by the invert function actually simulate the target distribution (see figure 3). This method requires having F easily to invert numerically, and it does not generalize to higher dimensions.

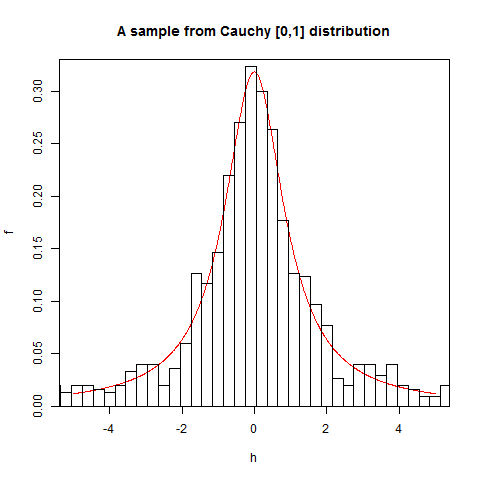


Figure 3 Histogram of y together with theoretical pdf C[0,1]

## 2.2 Rejection method for beta distribution

The idea of rejection method is that we simulate an auxiliary distribution g(x) (we know how to simulate) to actually mimic the target distribution that we do not know how to simulate.

We define the set *Sg ϵ* R2 as:

*Sg* = {(*x,y*)*ϵ*R2 *| xϵ*[0*,*1] *and yϵ*[0*,C* \* *g*(*x*)]}

Where C is a constant and it should be chosen such that *C\*g(x) >f(x),* which ensures that the enlarged auxiliary distribution can fully cover the target distribution, in this case, the target distribution is *beta (2,5).*

We first generate 1000 uniformly distributed random numbers in *Sg,*  then scale it and throw away those ones for which *y > f(x)* (see figure 4). The good thing about rejection method is that it can be generalized easily in higher dimensions.

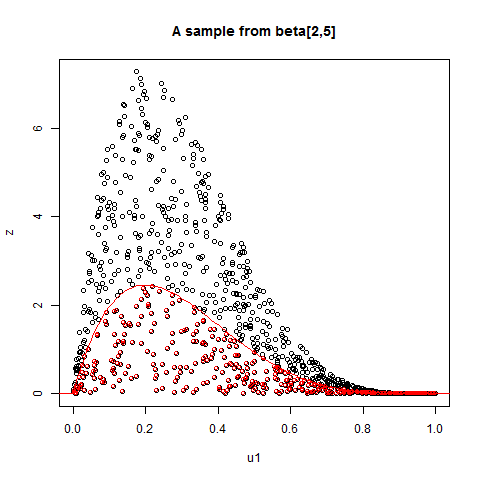
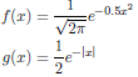
****

Figure 4 A sample from beta distribution. The majorated auxiliary distribution comprises of all the circles, and the target distribution only comprises of red circles.

## 2.3 Rejection method for double exponential distribution

The auxiliary distribution g(x) is given as a function of x that follows N(0,1) . We will use g(x) to mimic the target distribution f(x) which we don’t know. They two functions look like:

We again use C=2 to scale g(x) making sure the majorated distribution will fully cover the target distribution, then we throw away the data points (black circles) that are greater than g(x) (see figure 5).



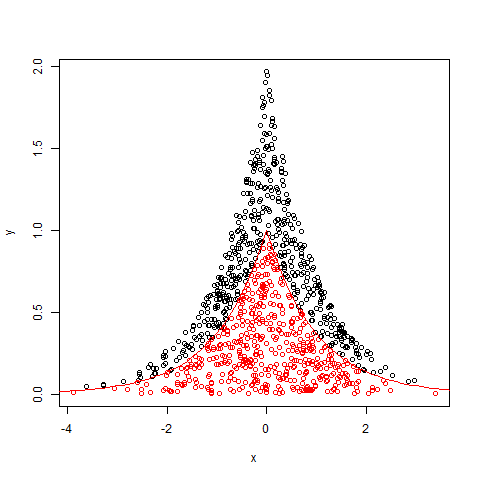


Figure 5 f(x) (red circles) together with Cg(x) (red and black circles)

## 2.4 Needle throwing

Each time we throw the needle, its head first falls into a plank at a point (x1,y1), and its tail falls either on a boarder of the plank (intersect) or totally inside a plank (not intersect). There are three variables in the beginning, the needle’s head landing position (x1,y1) and its angle (a) with the x coordinate. We will simulate the these variables following uniform distribution U(0,1) for position, U(0,360°) for angle. When the three variables are in place, we can calculate the needle’s tail landing position (x2,y2):

x2 = x1 + l\*cos(a)

y2 = y1 = l\*sin(a)

We assume that the plank is so long in the vertical direction that the needle cannot intersect on its boarder in the vertical direction, but only in the horizontal direction. So the judgement for intersection becomes quite clear, which we only need to count on the needle tail’s x coordinate, if that is inside the boarders, in this case it is (0,1).

I made the throwing in a way that I throw 10 times as a batch, and I made 1000 batches. Figure 6 visualize 9 batches with a probability. Finally, I have an average probability 0.635.

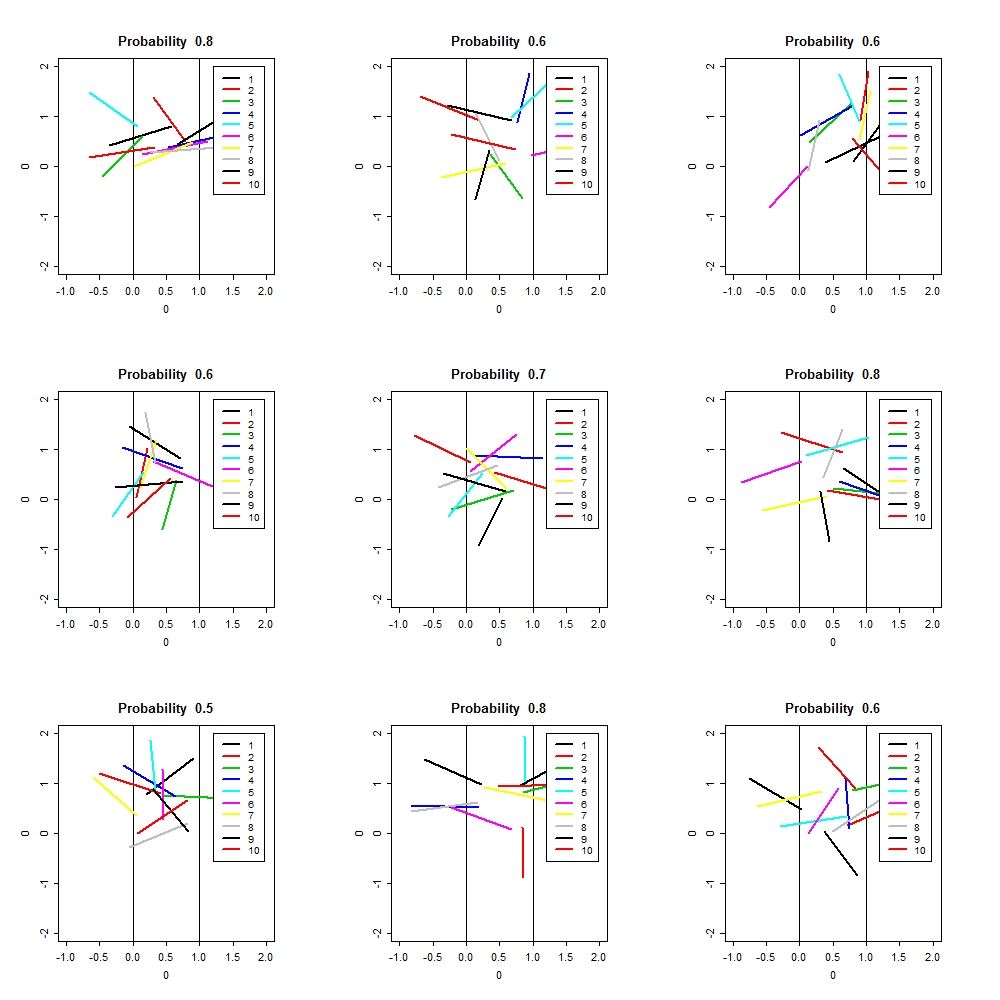
****

Figure 6 9 batches out of 1000 and the corresponding probabilities

## 2.5 Car selling with bid x following Poisson distribution

We simulate 10000 bids Xi by using Poisson(10) distribution for the car. For the conditional case, we count the order of bids Xi that are greater than 12. The first several Poisson distribution look like:

Xi = 8 10 8 9 9 7 10 10 11 6 11 12 10 10 14 9 12 7 14 5 9

The order of the bids that are bigger than 1:

ki = 15 19 23 25

So the waiting time is ki+1 - ki

di = 4 4 2

For the unconditional case, we are given the assumption that Xuns are i.i.d Poisson(10). So we set up the threshold for X such that we count the orders when X> Xun .

Finally, we have the waiting time 4.84 for conditional case and 2.17 for unconditional case.

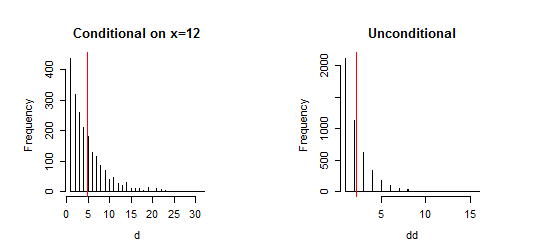
****

Figure 7 Histogram of waiting times for the cases: conditionally on Xi =12 (left) and unconditionally Xuns are i.i.d Poisson(10). (right)

## **2.6** Bootstrap**:** Variance in the estimation of the median of a Poisson distribution

We can estimate the accuracy of an estimator of the median by using bootstrap. Firstly, we generate n=200 random Poisson distribution data points. In order to estimate the variance of the median, we need to repeat the random generation of Poisson distribution p=1000 times. Therefore, we get a dataset (n x p). For theoretical median, we will calculate 1000 medians for each of the repetition; and for bootstrap median, we use the R inbuilt function “sample” to sample 200 data points from the 2000 random Poisson distribution data points with replacement. Because sampling 200 data points from a pool with 200 data points does not make sense. Then we calculate 1000 medians for each repetition.

We noticed that as p increases (law of large numbers), the variance of bootstrap median 0.1107 tend to be closer to the theoretical variance of median 0.1039 (see figure 8).

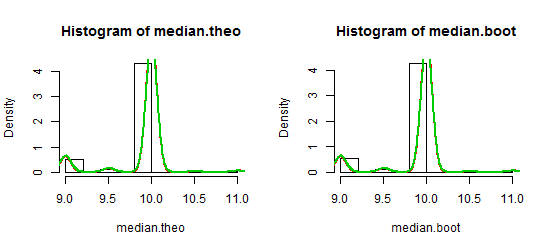


Figure 8 Histograms of medians when p is 1000

## 2.7 Bivariate mean and variance

Bivariate normal distribution is a kind of multivariate normal distribution (X = (X1,X2,…,Xn)T) with n-dimension equal to 2. Each of the component Xi of the follows monovariate normal distribution N(0,1).

1. So we simulate samples of n=50000 for both X = (X1,X2), which has a covariance matrix [1 0*.*7; 0*.*7 1]
2. Based on LU factorization, we can simulate MVN random vectors, i.e. Y=L\*X, where L is the

lower triangular matrix.

1. We check that var(Y) is a matrix close to the original covariance matrix and compare the scatter plots of both samples x and y (see figure 9).

Var(Y) =



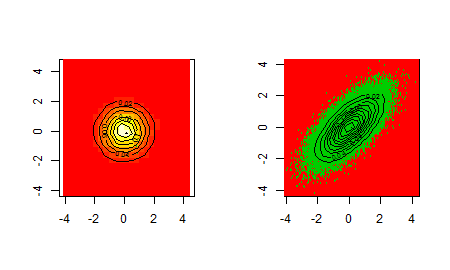


Figure 9 Scatter plot of sampled variables X (left panel) and Y (right panel) both of them of size n=50000

Now we have to extract a sample of a random variable such that Y2|Y1 =0.5. We code it in R by saying that sample Y2 when 0.49<= Y1 <=0.51. Figure 10 shows that the empirical distribution (green curve) is very close to the target distribution (red curve). The mean and variance of this sample are:

*mean*(Y2) = 0*.*37; *var*(Y2) = 0*.*54

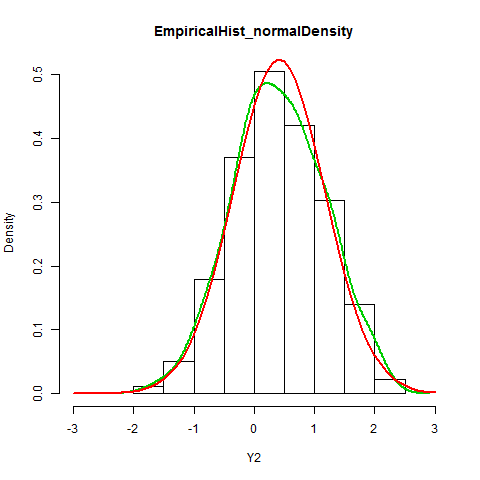


Figure 10 Normalized histogram of Y2 compared with: f(Y2| Y1=0.5) (green curve) and N(mean(Y2),sqrt(var(Y2)))

Exercise 03

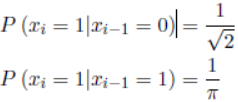
# 3. Markov chain Monte Carlo methods

## 3.1 General Metropolis Algorithm

Markov chain is a sequence of random values such that the distribution of the future depends on the past only through the present. It is difficult to give an explicit example of random sequence that does not have a Markov structure.

We want to simulate a Markov chain X={*xi*} whose entries has two states in the set S={0*,*1}.

We first initialize the Initialize x1 at random with p(x1 = 0) = p(x1 = 1) = ½ and consider the transition kernel:



A sequence with n=5000 elements are simulated, and they distributed either at 1 or 0 (see figure 11)

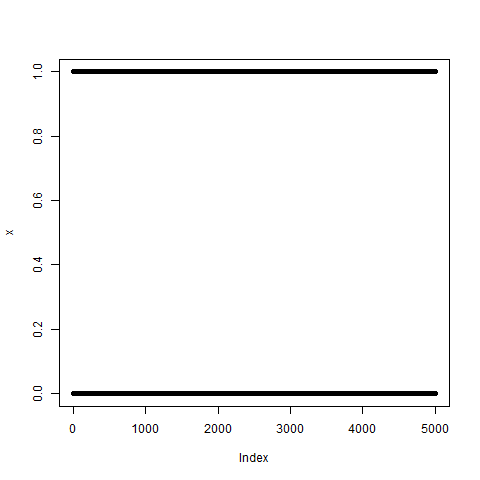


Figure 11 Markov chain sequence with n=1000 entries in S={0,1}.

## 3.2 Metropolis algorithm with the Cauchy distribution C(0; 1)

Taking the metropolis algorithm given in the lecture notes, we construct a Markov chain

of n=10000 elements targeting f: Cauchy(0,1) and using q: N(0,1) as proposal distribution. We decide the value of the next state by looking at the ratio R, which is the probability of the occurrence of the transition.



Pros of general metropolis:

1. Works for discrete and continuous distribution
2. Works in any dimension
3. Easy to implement

Cons of general metropolis:

1. If auxiliary q is chosen to be equal to the target f, R=1, this is optimal be useless.
2. If q is poorly chosen w.r.t f, R might be numerically unstable or low (Reject) most of the time.

The figure 12 implies that the N(0,1) might not be the ideal proposal for target Cauchy(0,1), the reasons being:

1. The scatter plot does not give a clear convergent trend; it seems that there is high density among the range from -2 to 2.
2. It shows that high density on the tails of the histogram, making the distribution different from the target C(0,1).

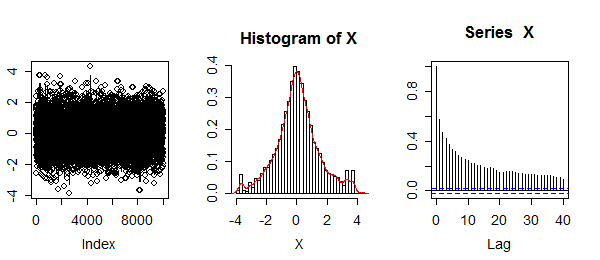


Figure 5 Left: complete Markov chain. Middle: normalized histogram corresponding. Right: autocorrelation plot

## **3.3** Metropolis-Hastings algorithm with Gaussian proposal

Metropolis-Hastings (M-H) algorithm is identical to the original Metropolis one but using

a proposal distribution that depends on the current state of the chain. We choose

N(x[i], sigma=1) as auxiliary pdf and repeat the process we performed in previous exercise.

1. Construct a chain of n=10000 elements using the new proposal.
2. Plotting chain elements (see figure 12 left)
3. Constructing the corresponding histogram and autocorrelation (see figure 12 middle)

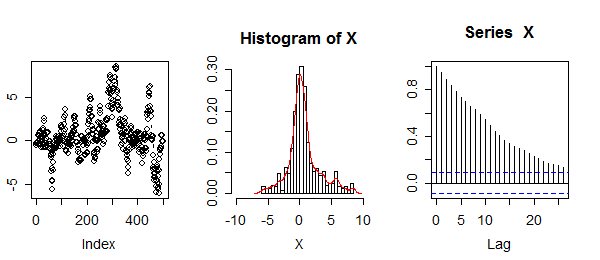


Figure 6 sigma= 1. Left:complete Markov chain; Middle: normalized histogram corresponding; Right: autocorrelation plot

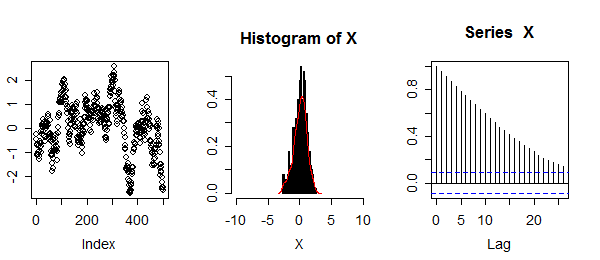


Figure 14 sigma= 0.3. Left: complete Markov chain; Middle: normalized histogram corresponding; Right: autocorrelation plot

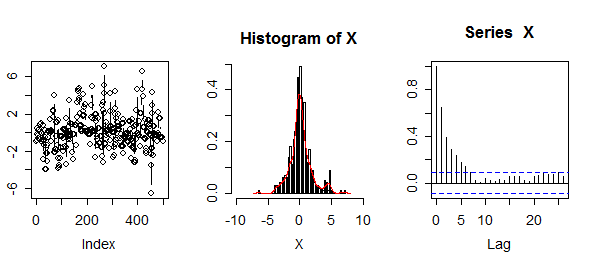


Figure 15 sigma= 3. Left: complete Markov chain; Middle: normalized histogram corresponding; Right: autocorrelation plot

Figure 13 shows that the new proposal N(x[i], sigma=1) produces less rejections than in ordinary metropolis. Notice that the points have a continuous appearance, meaning that the next state of the chain is close to the current one in contrast with the jumps observed from ordinary metropolis method. This is due to the new proposal’s dependence on current entry.Thus the autocorrelation is stronger than for the ordinary metropolis.

We expect the decrease of sigmain the proposal could produce a more continuous look for the points and consequently higher level of autocorrelation (see figure 14).

On the other hand, increasing the value of sigma would result in candidates being further away from the current value, producing jumping effect in scatter plot and less autocorrelation (see figure 15).

## 3.4 Rejection algorithm and HM simulating Beta

### Rejection algorithm:

We use auxiliary distribution U[0,1] and follow the procedure detailed in the rejection method form the Second Exercise. Notice that beta (1/2,1/2) diverges for x=0 and x=1, so we must be careful with the scaling parameter C. Figure 16 shows some results of the outcome sequence.

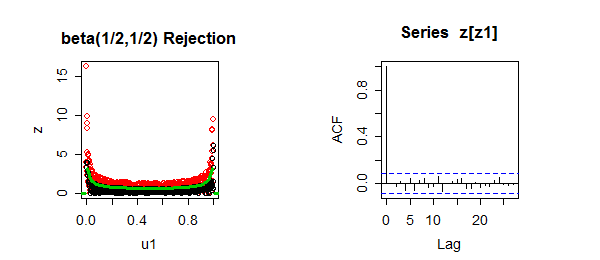


Figure 16 Left panel: Accepted (black) and rejected (red) points. Green line is the target distribution f. Right panel: autocorrelation null.

### MH algorithm:

We use sigma=1 for the proposal. Figure 17 shows the beta distribution and autocorrelation.

Except computation cost, both algorithms produce beta distribution compatible with the target distribution, but only the rejection method produces an uncorrelated sequence.

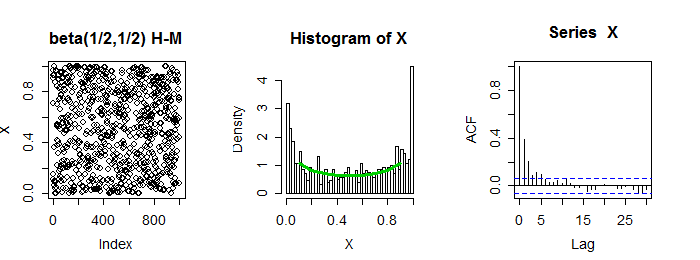


Figure 17 Left panel: beta distribution. Middle panel: normalized histogram, Green line is the target distribution f. Right panel: weak autocorrelation

## 3.5 Autocorrelation

The autocorrelation corresponding to the MCMC constructed in previous exercises show

correlations in the entries. **Burn-in** refers to discarding an initial portion of a Markov chain sample so that the effect of initial values on the posterior inference is minimized. After the burn-in period, the elements of the sequence follow the target distribution but they are not independent.  We see from the previous exercise when sigma is relatively small, the autocorrelation is high, and high sample autocorrelation can result in biased Monte Carlo standard errors. Usually, we can **thin** the Markov chain in order to reduce sample autocorrelations.

In exercise 4 we notice that Metropolis-Hastings algorithm produces a sequence less correlated when a big value of sigma is chosen for the proposal. Therefore, the corresponding thinned chain is obtained less costly.

## 3.6 Simulate MVN using Gibbs algorithm

Bivariate normal with covariance matrix sigma [1 *p* ; *p* 1] and the conditional distribution Y|X=x ~ *N*(*p*x,1-*p*2) are given, applying p = 0.5, n=1000 to simulate the sequences (X,Y).

The covariance matrix [1.0686, 0.5594, 0.5594, 0.9975] is close to the desired matrix sigma and the scatter plot of the sampled points (See figure 18) looks compatible with a bivariate normal. Moreover, the cross-correlation looks not bad, only a few spikes around lag 0 are high. The histograms appears typical normal distributions centered at 0 with corresponding covariance.

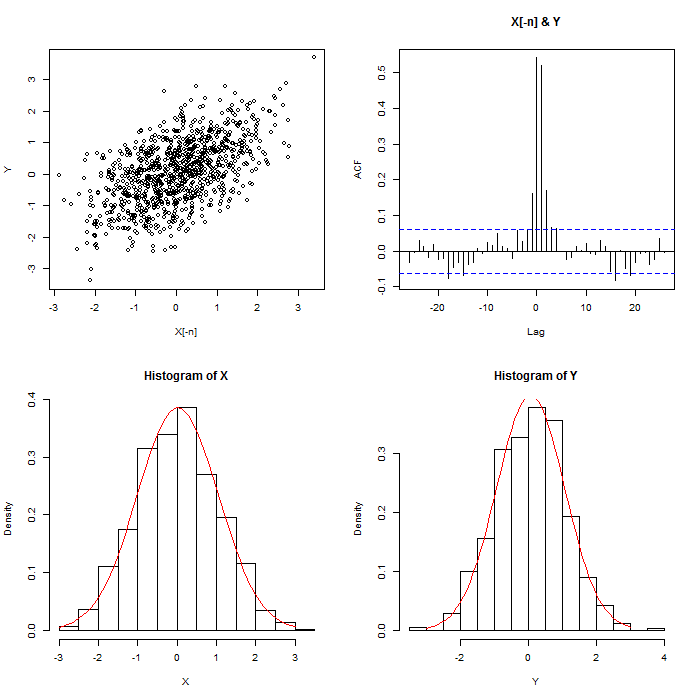


Figure 7 Upper left: Scatter plot of covariance matrix; upper right: cross correlation. bottom histograms for X and Y.

Exercise 04

# 4. Monte Carlo integration

# 4.1

The plot of this function on [0,1] see figure 19:

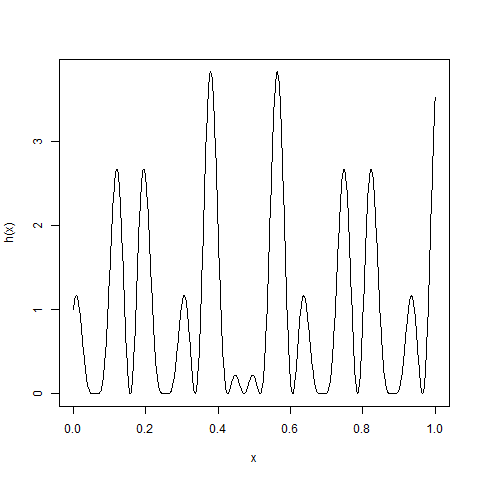
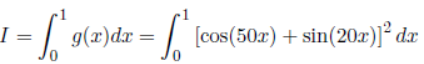


Figure 8 h(x) in the interval [0,1].

The function h(x) is given, we will consider its integral:



We can estimate this integral with stochastic simulations sampling from a uniform distribution.

This results very easy because f(x)=*I*[0*,*1](x) is exactly the unit in the integration domain (i.e. f(x)=1 and h(x)=g(x)). Therefore, we estimate I as:



where X={*xi*} is sampled from U[0,1]. Using n=105 numbers we obtain , which is close to 0.9652 yielded by the R function integrate.

# 4.2

Firstly, we know that the unit ball *Bd* is inside the hyper-cube *Cd,* which can be shown as *Bd* ϵ *Cd*  for all *d*. The sequence x=(*x*1,...,*xd*) is given as *Cd* ,such that |*xi*| *<* 1. We know how to simulate points uniformly distributed in the hyper-cube. *Bd*  is given from the same sequence such that sqrt(sum(*x*1^2) )<=1, which impose a condition on the hyper-cube, making sure that the sequence is inside the unit ball (see figure 20). The steps are:

1. Define dimension d{2,3,10}, observation n=10000, matrix [n x p], area *Cd* =2^p.
2. Formulate the function I. *Bd* (x) with the condition sqrt(sum(*x*1^2) )<=1.
3. Simulate sequences following uniform distribution.
4. Counts (*inlier*) how many elements are in the ball (meet the condition).
5. Volume estimate is area Cd multiply the ratio *inlier* / n.
6. Use 100 iteration, the variance estimate is sum (*Vd* -*E [**Vd])^2 \* 1/n^2*

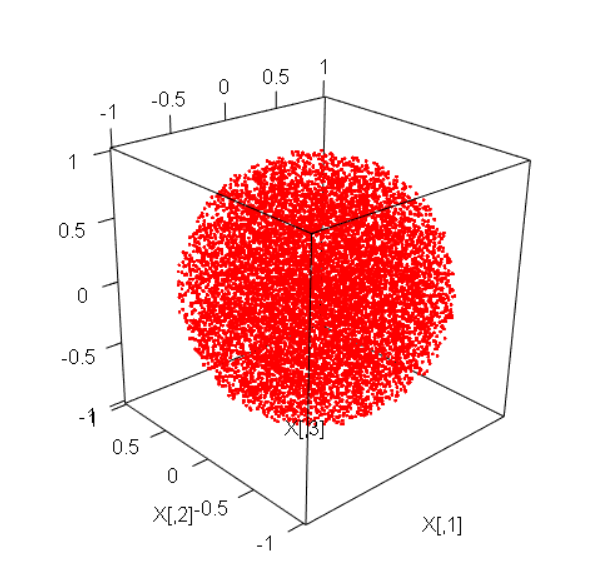
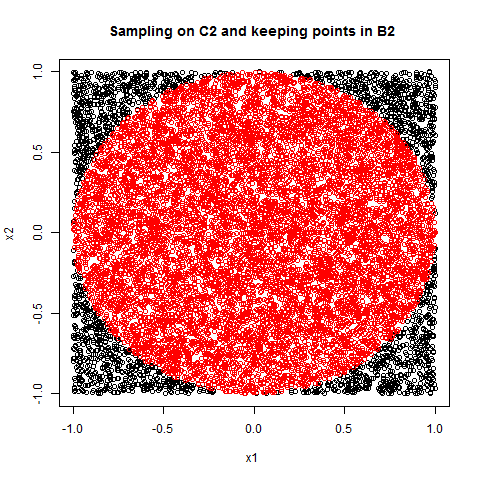


Figure 20 10000 points uniformly distributed in C2, red ones lie inside B2. Left: d=2, right d=3

Volume:

|  |  |  |
| --- | --- | --- |
| dimension | *Vd true* | *E [Vd] inter100* |
| 2 | 3.14159 | 3.139 |
| 3 | 3.14159 | 3.139 |
| 10 | 2.55016 | 2.562 |

Variance:

|  |  |
| --- | --- |
| dimension | *Var(Vd)* inter100 |
| 2 | 3.2635e-10 |
| 3 | 1.7505e-09 |
| 10 | 2.6453e-07 |

Exercise 05

# 5. Monte Carlo Optimization methods

## 5.1

Stochastic simulations can work as an optimization method in many field like network optimization, pricing, parameter estimation. Consider a multimodal distribution function *h(x)* defined on X (*X = Rp*), Consider we want to find a x0 that yields the maximum value of *h(x)*, *x0 = Arg**maxX h(x).* in the exercise the function is given:

*h*(*x*) = 3\**d*(*x,* − 2*,*0*.*2) + 6\**d*(*x,*1*,*0*.*5) + *d*(*x,*3*,*0*.*3)

The function is positive and have three peaks that comprise of one global maximizer at x=2, and two local maximizers at x=1, and x=3.

### 5.1.1 Uniform simulation algorithm

According to the algorithm from the lecture notes, We first sample n=100 numbers from uniform distribution on [-5,5], and check which x yields the greatest value for *h(x)*. We found

*Xmax*=1.9914 that is very close to the true maximum. Figure 21 left shows *h(x)* and the uniform sample as scatters at bottom, and a vertical line indicates the maximum.

### Pros and Cons of uniform algorithm:

1. It is easy to implement and general.
2. It is in a way blind to *h(x)* that it simulates samples far away from the maximum, resulting in computational costly.

So a better approach would be using a none uniform algorithm, such that it sample only the x that are close to maximum of *h(x)*. could spare computation,

### 5.1.2 None uniform algorithm-Rejection method

Rejection method can be one of the none uniform method. Alternatively, we can also use Metropolis, but considering computational cost, we’d better use Rejection, as Metropolis needs to run for-loop in R code.

The rejection method requires the function *h(x)* to be normalized, simply divided *h(x)* by its integral 10, which yields *hh(x)*. Then follow these steps:

1. Use sample size 1000, and sample uniform distribution for both x=U(-4,4) and y=U(0,1).
2. Plug xi into *hh(x)* and extract only those xi that are smaller than *hh(x)* function curve.

Rejection method produces a maximum 1.9967, and the samples are concentrated to the maximums (see figure 21 right ).

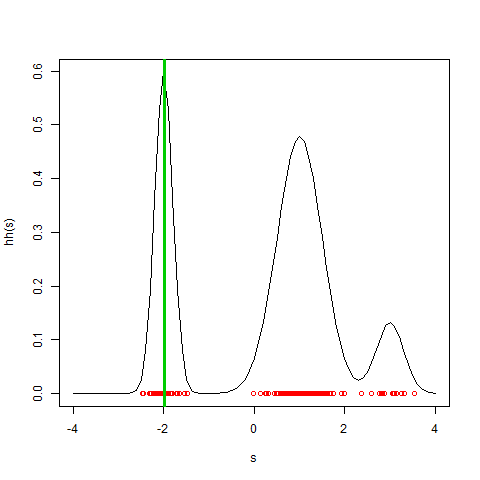
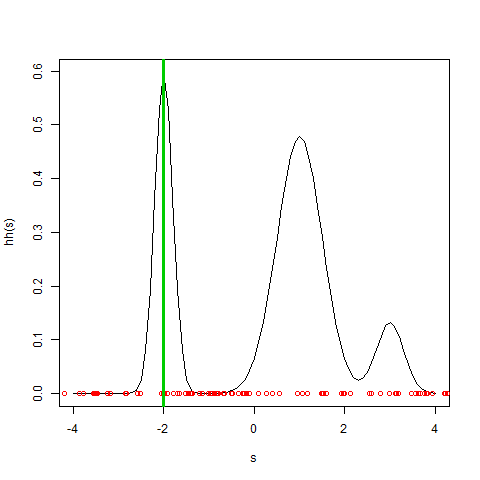
****

Figure 21 Objective function h(x) with uniform distribution U[-5,5] (left) and 0.1\*h(x) (right).

### 5.1.3 Comparison with R inbuilt “optimize” function

Using R inbuilt optimize in the interval [-4,4] yields a maximum 1.000007, which is one of the local maximum, which provides a prove that this R function could sometimes trap in a local maximum.

### 5.2 Simulated annealing method to maximize h on [-3; 3]^2

A sequential method that mixes features of the stochastic gradient method and of the tempered density simulation method.

In this exercise a 2D normal distribution function (P=(*x*,*y*) ϵ R2) is given:

*h*(*x,y*) =3*d*(*x,*0*,*0*.*5)*d*(*y,*0*,*0*.*5) + *d*(*x,* − 1*,*0*.*5)*G*(*y,*1*,*0*.*3) +*d*(*x,*1*,*0*.*5)*d*(*y,*1*,*0*.*3)

This function is positive and gradient is steep as the two variables are close to 0. P=(0,0) is the global maximizer of h that present other 2 local maximizers at P=(-1,1) and P=(1,1). Figure 22 shows the contour plot. Light color indicates high function value (center of contour), whereas dark color indicates low function value (rim).

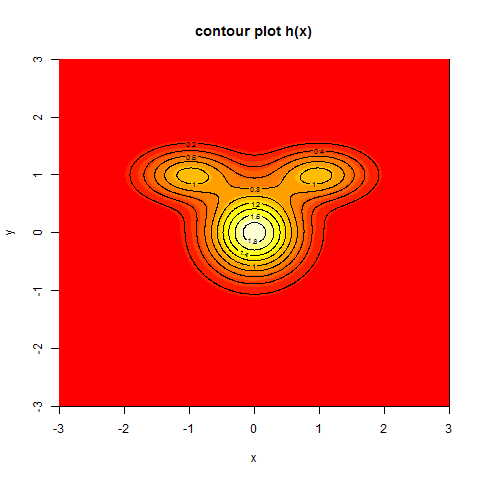


Figure 22 Contour plot of the objective function h(x).

### Annealing method finding the maximum follows these steps:

1. Choose a deterministic sequence *Ti* = C/Log(i) for *i* >= 1.
2. Choose two auxiliary distributions, one is uniform U(-3,3)2, another is a bivariate normal N(0,I2).
3. Initialize point P0[1,] from the uniform distribution,.
4. Iteration start with sampling a very small number ξ from bivariate normal N, then propose a candidate value P\* =Pi + ξ.
5. Impose a recursive process such that If is out of [-3; 3]^2, we do the sampling of ξ again and produce a P\* again.
6. Function’s difference when both variables changes a little bit: ∆h(Px,Py)= h(P\*x,P\*y) - h(Px,Py).
7. Compute the exponential of the slope and compare it with 1 to get the minimum:
8. ρ =min (exp ( (∆h/Ti), 1)
9. Accept the transition from Pi to P\* with probability ρ of a binomial random distribution.

Figure 23 shows that the algorithm starts from an initial point that could be anywhere on the 2D plot, and then at the second iteration it quickly jumps to the track and evolves to the maximum. Finally, the solution properly settled on the area that is very close the maximum of function h(x,y).

### Properties of annealing method (as the lecture notes mentioned):

1. Random walk algorithm, as we assign a ξ each iteration.
2. Moves increasing h are always accepted, the green line in general approaches the maximum.
3. Moves decreasing h are sometimes accepted, the green line can sometimes goes farther away from the maximum.
4. Moves decreasing h less likely accepted as i increases.
5. Rationale: for large i, one should be close to the solution. The green points indicates the last iteration.
6. Requires choosing (Ti )i>=1: often Ti = C/ log i
7. Requires choosing and tuning g: often N(0; Ip)

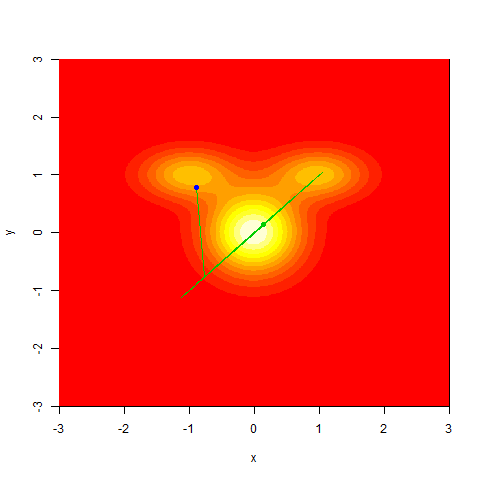


Figure 23 How annealing method finds the maximum. blue is initial point, green is the end point.

# Appendix

**Exercise 01**

rm(list=ls())

## Linear congruential method

a = 16807 ; b =0 ; m =2147483647

n = 1000 # nb of iterations

x = numeric(length=n)

x[1] = 3

for(i in 2:n)

{

x[i] = (a\*x[i-1] + b) %% M

}

x = x/m

png('hist.png')

par(mfrow=c(2,2))

par(mar=c(4,4,1,1))

hist(x,prob=TRUE,freq=F)

u = seq(0,1,0.01)

lines(u,dunif(u),col=2) # density f(x) = 1/(max-min) punif(u)=u; dunif(u)=1

# h=2

plot(x[1:98],x[3:100])

plot(x[1:98],x[3:100],type='l')

#plot(x,type='l')

acf(x)

dev.off()

cov(x[1:98],x[3:100]) #[1] -0.0007267802

ks.test(x,"punif",alternative="two.sided")

# punif:Uniform Distribution function. (cumulative distribution function)

# a large p-value indicates accept H0: no difference between simulated RandomNumbers and theoretical

x1<-runif(n=100,min=0,max=1)

hist(x1,prob=TRUE,freq=F)

u = seq(0,1,0.01)

lines(u,dunif(u),col=2)

**Exercise 02**

**##Question 1 ####**

**rm(list=ls())**

**n = 1:1000**

**h = seq(-5,5,0.01)**

**f = 1/(pi \* (1+h^2)) ##pdf**

**png('Cauchy.png')**

**plot(h,f,type='l',col=2,main="A sample from Cauchy [0,1] distribution") # target**

**u = runif(n)**

**x = tan(pi\*u - pi/2) ## F cdf Empirical**

**hist(x,breaks=seq(min(x),max(x)+1,0.3),add=TRUE,prob=TRUE)**

**dev.off()**

**##Question 2 ####**

**n = 1000;**

**u1 = runif(n); #X**

**u2 = runif(n); # majorating density**

**#C <- max(dbeta(h,2,5,ncp=0)) + 0.1**

**C=3;**

**z = u2\*C\*dbeta(u1,2,5,ncp=0) # Y = Ug(X)**

**h = seq(-.1,1.1,0.01)**

**png('RejectionMethod.png')**

**plot(u1,z,main="A sample from beta[2,5]")**

**lines(h,dbeta(h,2,5,ncp=0),type='l',ylim=c(0,7),col=2,cex=1.9)**

**accept <- z <= dbeta(u1,2,5,ncp=0)**

**points(u1[accept],z[accept],col=2,cex=.8)**

**dev.off()**

**## Question 3####**

**n=1000; x = rnorm(n,0,1); u=runif(n); C=2;**

**g = dexp(abs(x));**

**#a = 1; g = a/2\*exp(-a\*abs(x));**

**y =u \* C\*g;**

**h = seq(-5,5,.1)**

**png('DoubleExponentialDistribution.png');**

**plot(x,y)**

**lines(h,dexp(abs(h)),col=2)**

**i = y<=g;**

**points(x[i],y[i],col=2);**

**dev.off**

**##Question 4 #####**

**n=10;l=1;w=1; iter=9;**

**#test = matrix(nrow=n,ncol=iter,data=NA);**

**test =numeric(n)**

**png('IntersectionProb.png',height= 1000, width=1000)**

**par(mfrow=c(3,3),pty = "m",mex=.7,pin=c(3,3),cex=0.9)**

**# for(i in 1:9){**

**# plot(x,y,xlim=c(-1,2),ylim=c(-2,2),type='n');**

**# abline(v=0);abline(v=1)**

**# }**

**prob = rep(0,iter)**

**for(j in 1:iter){**

**plot(x,y,xlim=c(-1,2),ylim=c(-2,2),type='n')**

**abline(v=0);abline(v=1)**

**for(i in 1:n){**

**end = runif(2,0,w);**

**a = runif(1,0,2\*pi);**

**head = numeric(2)**

**head[1] = end[1] + cos(a)**

**head[2] = end[2] + sin(a)**

**x = c(end[1],head[1])**

**y = c(end[2],head[2])**

**lines(x,y,type='l',col=i, xlim=c(-1,2),ylim=c(-2,2),lwd=2)**

**legend(1.2,2,1:n,lwd=rep(2.5,n),col=1:n,cex=.9)**

**test[i]= head[1]>l || head[1]<0;**

**}**

**prob[j] = mean(test);**

**#title(prob[j])**

**title(main = paste("Probability ", prob[j]),cex=1.2)**

**}**

**dev.off()**

**prob1 = mean(prob)**

**##Question 5 #####**

**N=10000; lambda = 10;**

**X = rpois(N,lambda);**

**X1 = rpois(N,lambda);**

**# k = X>12;**

**# X[y]**

**k <- which(X>12)**

**d<- diff(k) #waiting time**

**m1=mean(d)**

**kk <- which(X>X1)**

**dd <-diff(kk)**

**m2=mean(dd)**

**#unconditional**

**png('poission\_conditional\_unconditional.png',height=250,width=550)**

**par(mfrow=c(1,2),pty = "s",mex=.8,pin=c(2,2),cex=.9)**

**hist(d,breaks=seq(min(d),max(d),0.01),main="Conditional on x=12") # most waiting time<=5**

**abline(v=m1,col=2);**

**hist(dd,breaks=seq(min(dd),max(dd),0.01),main="Unconditional")**

**abline(v=m2,col=2)**

**dev.off()**

**prob = lambda/N;**

**#k1 = X<=12;**

**k1 <- which(X<12);**

**dge <- dgeom(k, prob, log = FALSE) # probability that the kth trial (out of k trials) is the first success**

**# prob that the first success requires k number of trials**

**dge <- (1-prob)^(k-1) \* prob**

**png('Geom.png')**

**plot(dge,type = "o", main = "The Geometric Distribution")**

**hist(dge,breaks=20,prob=TRUE)**

**dev.off**

**##Question 6 #####**

**n=200; lambda = 10; p = lambda/n; nn=2000;**

**X1 = rpois(n,lambda);mean(X1); X2 = rpois(nn,lambda);mean(X2);**

**median(X1); median(X2)**

**B <- 2000**

**X1.theo <- X1.boot <- matrix(nrow=n,ncol=B)**

**for(b in 1:B)**

**{**

**noise=rnorm(n,0,1/sqrt(n))**

**X1.theo[,b] <- rpois(n,lambda);**

**X1.boot[,b] <- sample(x=X2,size=n,replace=TRUE)**

**}**

**median.theo <- apply(X1.theo,MARGIN=2,FUN=median)**

**median.boot <- apply(X1.boot,MARGIN=2,FUN=median)**

**png("median.png",height=250,width=550)**

**par(mfrow=c(1,2))**

**hist(median.theo,prob=TRUE)**

**lines(density(median.theo),col=2,lwd=2);**

**lines(density(median.boot),lwd=2,col=3);**

**hist(median.boot,prob=TRUE)**

**lines(density(median.theo),col=2,lwd=2);**

**lines(density(median.boot),lwd=2,col=3);**

**dev.off()**

**var(median.theo)**

**var(median.boot)**

**lambda/n # truth**

**##Question 7 #####**

**#standardize Gaussian: N(0,1), cov() = 0.7**

**sd =1 ; rho = .7**

**Sigma = matrix(nr=2,nc=2,data=c(sd,rho,rho,sd))**

**Sigma**

**## Choleski factorisation**

**U = chol(Sigma) ; L = t(U)**

**L**

**L %\*% U ## just checking**

**# simulation**

**n = 50000**

**x1 = rnorm(n) ; x2 = rnorm(n)**

**X = rbind(x1,x2)**

**Y = L %\*% X**

**Y = t(Y)**

**plot(Y,asp=1) # one data unit in the x direction is equal in length to asp \* one data unit in the y direction.**

**## cheking that the empirical var-covar matrix is close to the target. ij = Cov(Xi ;Xj )**

**cov(Y)**

**Sigma**

**## cheking that empirical bivariate density**

**require(MASS)**

**png('empirical bivariate density.png',height=270,width=450)**

**par(mfrow=c(1,2))**

**image(kde2d(x1,x2),asp=TRUE) ; points(X,cex=.3,col=3)**

**contour(kde2d(x1,x2),add=TRUE)**

**image(kde2d(Y[,1],Y[,2]),asp=TRUE) ; points(Y,cex=.3,col=3)**

**contour(kde2d(Y[,1],Y[,2]),add=TRUE)**

**dev.off()**

**## extrate y2|y1 =0.5**

**Y2 = Y[,2]**

**Y1 = Y[,1]**

**Y2 = Y2[Y1>=.49 & Y1<=.51]**

**Y1 = Y1[Y1>=.49 & Y1<=.51]**

**Y2 = t(Y2)**

**Y1 = t(Y1)**

**YYY = rbind(Y1,Y2)**

**m2 = mean(Y2)**

**v2 = var(t(Y2))**

**png('EmpiricalHist\_normalDensity.png')**

**hist(Y2,breaks=10,prob=TRUE,main="EmpiricalHist\_normalDensity",xlim=c(-3,3)) #empirical**

**lines(density(Y2, kernel = "gaussian"),col=3,lwd=2)**

**lines(seq(-3,3,0.1),dnorm(seq(-3,3,0.1),m2,sd(t(Y2))),col=2,,lwd=2)**

**dev.off()**

**Exercise 03**

**rm(list=ls())**

**##1####**

**#Simulate a Markov chain with values in S = {0,1} such that P(xi=1|xi-1=0) = 1/sqrt(2) and P(xi = 1|xi-1 =1) = 1/pi Plot the result.**

**n <- 5000 ; x <- rep(NA,n)**

**x[1] <- rbinom(n=1,size=1,prob=1/2)**

**for(i in 2:n)**

**{**

**x[i] <- ifelse(x[i-1]==0,**

**rbinom(n=1,size=1,prob=1/sqrt(2)),**

**rbinom(n=1,size=1,prob=1/pi))**

**}**

**png('Simple MC.png')**

**plot(x)**

**dev.off()**

**##2 Metropolis algorithm####**

**n <- 10000 ; X <- rep(NA,n)**

**X[1] <-rnorm(n=1,m=0,sd=1) # Init**

**f<- function(X) {dcauchy(X,location = 0, scale = 1) }**

**#f<- function(X,l=0,s=1) {1/(pi\*s (1 + ((X-l)/s)^2))}**

**for(i in 1:(n-1)) # Iterate random update**

**{**

**Y <- rnorm(n=1,m=0,sd=1) # propose candidate value**

**# compute Metropolis ratio**

**R <- (f(Y) \* dnorm(X[i],m=0,sd=1)) / (f(X[i]) \* dnorm(Y,m=0,sd=1))**

**# compute acceptance probability**

**p <- min(1,R)**

**# toss an unfair coin to decide if candidate accepted**

**accept <- rbinom(n=1,size=1,prob=p)**

**X[i+1] <- ifelse(accept==1, Y, X[i])**

**}**

**png('Metropolis.png',height= 260, width=600)**

**par(mfrow=c(1,3),pty = "s",mex=.7,pin=c(1.9,1.9),cex=1.2)**

**plot(X,type='b')**

**hist(X,prob=T,breaks=50)**

**lines(density(X),col=2)**

**acf(X)**

**dev.off()**

**## 3 Metropolis-Hastings algorithm ####**

**n <- 500 ; X <- rep(NA,n)**

**X[1] <- rnorm(n=1,m=0,sd=1) # Init**

**sigma <- 3**

**f<- function(X) {dcauchy(X,location = 0, scale = 1) }**

**for(i in 1:(n-1)) # Iterate random update**

**{**

**Y <- rnorm(n=1,mean=X[i],sd=sigma) # propose candidate value**

**# compute Metropolis ratio**

**R <- (f(Y) \* dnorm(X[i],mean=Y,sd=sigma)) /**

**(f(X[i]) \* dnorm(Y,mean=X[i],sd=sigma))**

**# compute acceptance probability**

**p <- min(1,R)**

**# toss an unfair coin to decide if candidate is accepted**

**accept <- rbinom(n=1,size=1,prob=p)**

**X[i+1] <- ifelse(accept==1, Y, X[i])**

**}**

**png('Metropolis\_Hastings.png',height= 260, width=600)**

**par(mfrow=c(1,3),pty = "s",mex=.7,pin=c(1.9,1.9),cex=1.2)**

**plot(X,type='b')**

**hist(X,prob=T,breaks=50,xlim=c(-10,10))**

**lines(density(X),col=2)**

**acf(X)**

**dev.off()**

**## 4 Rejection & Metropolis-Hastings ####**

**# Rejection: probability of rejection increases exponentially as dimension increases**

**n = 1000;**

**u1 = runif(n);**

**u2 = runif(n); # majorating density**

**#C <- max(dbeta(h,2,5,ncp=0)) + 0.1**

**C=2;**

**z = u2\*C\*dbeta(u1,1/2,1/2) # Y = Ug(X)**

**h = seq(-.1,1.1,0.01)**

**png('RejectionMethod.png',height= 260, width=600)**

**par(mfrow=c(1,2),pty ="s",mex=.7,pin=c(1.9,1.9),cex=1.2)**

**plot(u1,z,col=2,main='beta(1/2,1/2) Rejection')**

**#points(u1,z,col=2,cex=.8)**

**z1 <- z <dbeta(u1,1/2,1/2)**

**points(u1[z1],z[z1],col=1,cex=.8)**

**lines(h,dbeta(h,1/2,1/2),col=3,lwd=3)**

**acf(z[z1])**

**dev.off()**

**# MH: dimension insensitive**

**n <- 1000 ; X <- rep(NA,n)**

**X[1] <-runif(n=1, min = 0, max = 1) # Init**

**sigma <- 1**

**f<- function(X) {dbeta(X,.5, .5,ncp = 0) }**

**for(i in 1:(n-1)) # Iterate random update**

**{**

**Y <- runif(n=1, min = 0, max = 1) # propose candidate value**

**# compute Metropolis ratio**

**R <- (f(Y) \* dunif(X[i])) / (f(X[i]) \* dunif(Y))**

**# compute acceptance probability**

**p <- min(1,R)**

**# toss an unfair coin to decide if candidate is accepted**

**accept <- rbinom(n=1,size=1,prob=p)**

**X[i+1] <- ifelse(accept==1, Y, X[i])**

**}**

**png('MH\_beta.png',height= 260, width=700)**

**par(mfrow=c(1,3),pty ="s",mex=.7,pin=c(1.9,1.9),cex=1.2)**

**plot(X,main='beta(1/2,1/2) H-M')**

**hist(X,prob=T,breaks=50)**

**s <- seq(0,1,.1)**

**lines(s,dbeta(s,.5,.5),col=3,lwd=3)**

**#lines(density((X)),col=2)**

**acf(X)**

**dev.off()**

**## 5####**

**#5 In the three examples above, estimate the auto-correlation functions of the simulated process (use the R function acf).**

**## 6####**

**iter<-10000; n <- 1000 ;mu = 0; sigma<-.5**

**#f<- functin(X){mvrnorm(n = X, mu, Sigma, tol = 1e-6, empirical = FALSE, EISPACK = FALSE)}**

**X <- rep(NA,n);**

**Y <- rep(NA,n-1);**

**X[1] <- rnorm(n=1,mu,sigma) # Initial**

**for(i in 1:(n-1)) # Iterate random update**

**{ # Two-stage Gibbs sampler**

**Y[i] <- rnorm(n=1,mean=sigma\*X[i], sqrt(1-sigma^2)) # propose candidate values**

**X[i+1] = rnorm(n=1,mean=sigma\*Y[i], sqrt(1-sigma^2))**

**}**

**png('Gibbs\_bivariate.png',height= 700, width=700)**

**layout(matrix(c(1,2,3,4), 2, 2, byrow = TRUE))**

**plot(X[-n],Y)**

**#plot(X)**

**ccf(X[-n],Y)**

**var(cbind(X[-n],Y))**

**hist(X,prob=T)**

**lines(seq(-3,3,.1),dnorm(seq(-3,3,.1),mean(X),sd(t(X))), col=2)**

**hist(Y,prob=T)**

**lines(seq(-3,3,.1),dnorm(seq(-3,3,.1),mean(Y),sd(t(Y))), col=2)**

**#lines(density(X),col=2)**

**dev.off()**

**# compute Metropolis ratios**

**R <- (f(Y) \* dnorm(x=XX[i,],mean=sigma\*XX[i,], sigma= 1-sigma^2)) / (f(XX[i,]) \* dnorm(x=Y,mean=sigma\*XX[i,], sigma= 1-sigma^2))**

**# For q chosen to be equal to f , R = 1.**

**# compute acceptance probabilities**

**p = rep(1,nchains)**

**p[R<1] = R[R<1]**

**# toss an unfair coin to decide if candidates are accepted**

**accept <- as.logical(rbinom(n=nchains,size=1,prob=p))**

**XX[i+1,accept] = Y[accept]**

**XX[i+1,!accept] = XX[i,!accept]**

**}**

**plot(XX[,1],type="l")**

**for(ichain in 1:10)**

**{ lines(XX[,ichain],col=ichain) }**

**par(mfrow=c(2,2))**

**for(iter in c(1,10,1000,10000))**

**{**

**hist(XX[iter,],breaks=seq(-15,15,0.1),**

**main=paste('Iteration',iter),xlab='x',prob=TRUE)**

**lines(seq(-10,10,0.01),f(seq(-10,10,0.01)),col=2)**

**}**

**y = c(rexp(n=100000,rate=1),-rexp(n=100000,rate=1))**

**lines(density(y),col=3,lty=2,lwd=3)**

**var(y)**

**for(iter in c(1,10,1000,10000))**

**{ print(paste("Iteration",iter,"; Variance=",var(XX[iter,]))) }**

**plot(XX,type='b')**

**hist(XX,prob=T,breaks=50)**

**lines(density(XX),col=2)**

**acf(XX)**

**Exercise 04**

##exercise Monte Carlo integration I ####

p = 1 ; n = 100000

X =rep(0,n)

for(i in 1:n) {X[i] =runif(n=p,min=0,max=1)} #g(x)

hist(X,prob=T)

h <-function(X){ (cos(50\*X) + sin(20\*X))^2} #h(x)

png('integral.png')

s=seq (0 ,1 ,0.001)

plot (s,h(s),type ='l',col =1, xlab ="x", ylab ="h(x)")

dev.off()

I = mean(h(X))

integrate(h,0,1)

##exercise Monte Carlo integration II ####

rm(list=ls())

p = 2 ; n = 10000

X = matrix(nrow=n,ncol=p,data=1)

for(i in 1:n)

{

while(sum(X[i,]^2)>1) { X[i,] = runif(n=p,min=-1,max=1) }

} # g(x)

png("Ball.png")

#png("ising1\_inter1to10.png")

# par(mfrow=c(1,2),pty = "s",mex=.7,pin=c(1.9,1.9),cex=1.2)

plot(X[,1],X[,2],col=2)

# plot(X[,1])

# hist(X[,1],prob=T)

library(rgl)

plot3d(X, col="red", size=3)

dev.off()

Cd =2^p; inliner =0;

I.Bd <- function (x){ifelse (sqrt(sum(x ^2))<1, 1,0)}

X = y = matrix (nr=n,nc=p, data =0) # allocate memory

index =rep (1,n)

X[ ,1]= runif (n , -1 ,1)

X[ ,2]= runif (n , -1 ,1)

for(i in 1:n){

inliner = inliner + I.Bd(X[i ,])

if( I.Bd(X[i ,]) ==0) { index [i]=0}

}

for(i in 1:n){

if( index [i]==1) {

Y[i,]=X[i,]

}

}

png ("Uniball\_dimension=2.png ")

plot (X[,1],X[,2], main ='Sampling on C2 and keeping points in B2',xlab ='x1',yla='x2')

points (Y[,1] ,Y[,2] , col =2)

dev.off ()

Vd\_true =pi ^(p/2)/ gamma (1+p/2)

Vd= Cd\* inliner /n # Cd\* rario of (unit ball / hypercube)

h = sqrt(apply(X=Y^2,MARGIN=1,FUN=sum))# radius of each p in the unit circle

I = mean(h) # E[h(X)] mean of radius

pi\*I^2

# Estimator of the variance of the Monte Carlo estimate

var <- 1/n^2 \* sum((h-I)^2)

# approximate confidence interval:

a=0.05

1-a/2

q <- abs(qt(p=a/2, df=n)) # 0.975 confidence, 2 sided

CI <- c( Vd - 1.96 \* sqrt(var), Vd + 1.96 \* sqrt(var) )

##var####

p=10; Cd =2^p; B=100; In = rep(0,B); col=1:p

I.Bd <- function (x){ifelse (sqrt(sum(x ^2))<1, 1,0)}

X = y = matrix (nr=n,nc=p, data =0) # allocate memory

#index =rep (1,n)

# X[ ,1]= runif (n , -1 ,1)

# X[ ,2]= runif (n , -1 ,1)

for(j in 1:B){

inliner =0;

for(k in 1:p){

X[,k]= runif (n ,-1 ,1)}

for(i in 1:n){

inliner = inliner + I.Bd(X[i ,]);

#if( I.Bd(X[i ,]) ==0) { index [i]=0}

}

In[j] = inliner;

}

Vd= Cd\* In /n;

I = mean(Vd);

var <- sum((Vd-I)^2) / (n^2)

var(Vd)

##var using rnorm####

p=2; Cd =2^p; B=100; In = rep(0,B); col=1:p

I.Bd <- function (x){ifelse (sqrt(sum(x ^2))<1, 1,0)}

X = y = matrix (nr=n,nc=p, data =0) # allocate memory

#index =rep (1,n)

# X[ ,1]= runif (n , -1 ,1)

# X[ ,2]= runif (n , -1 ,1)

for(j in 1:B){

inliner =0;

for(k in 1:p){

X[,k]= rnorm (n ,0,1)}

for(i in 1:n){

inliner = inliner + I.Bd(X[i ,]);

#if( I.Bd(X[i ,]) ==0) { index [i]=0}

}

In[j] = inliner;

}

Vd= Cd\* In /n;

I = mean(Vd);

var <- sum((Vd-I)^2) / (n^2)

var(Vd)

**Exercise 05**

##Exercises I ####

# uniform simulation

h <- function(x){3\*dnorm(x,m=-2,sd=.2) +

6\*dnorm(x,m=1,sd=.5) +

1\*dnorm(x,m=3,sd=.3)}

n <- 100

U <- runif(n=n,min=-5,max=5)

s=seq(-4,4,0.1)

plot(s,h(s),type='l')

istar <- which.max(h(U))

xstar <- U[istar]

hstar <- h(xstar)

png('uniform-max.png')

plot(s,hh(s),type='l')

points(U,rep(0,n),col=2)

abline(v=xstar,col=3,lwd=3)

# points(xn,yn,col=2)

# points(xn,rep(0,length(xn)),col=3)

dev.off()

# non-uniform simulation: rejection

n <- 1000

h <- function(x){3\*dnorm(x,m=-2,sd=.2) +

6\*dnorm(x,m=1,sd=.5) + 1\*dnorm(x,m=3,sd=.3)}

C = 2

s <- seq(-4,4,0.1);

x = runif(n,-4,4)

y = runif(n,0,1);

hh <- function(x){return(h(x)/integrate(h,-5,5)$value)}

accept <- y<hh(x) # uniform dist under the hh(x) curve

xn = x[accept]

yn = y[accept]

istar <- which.max(hh(xn))

h(xn)[istar] #5.9840

xn[istar] # maximum

png('non-uniform-rejection.png')

plot(s,hh(s),type='l')

points(xn,rep(0,length(xn)),col=2)

# points(x,y)

# points(xn,yn,col=2)

# points(xn,rep(0,length(xn)),col=3)

abline(v=xn[istar],col=3, lwd=3)

dev.off()

# R inbuilt

optimize(h,lower=-4, upper = 4, maximum = T)

##Exercises II: Annealing ####

rm(list=ls())

h = function(x,y)

{

3\* dnorm(x,m=0,sd=.5)\*dnorm(y,m=0,sd=.5) +

dnorm(x,m=-1,sd=.5)\*dnorm(y,m=1,sd=.3) +

dnorm(x,m=1,sd=.5)\*dnorm(y,m=1,sd=.3)

}

png('contour.png')

x = y = seq(-3,3,0.01)

H = outer(x,y,h)

image(x,y,H,main='contour plot h(x)')

contour(x,y,H,add=TRUE)

dev.off()

image(x,y,H)

X = cbind(x,y)

# lines(X,type='l',lwd=.5)

# n = 100

# points(X[1,1],X[1,2],col=4,pch=16)

# points(X[n,1],X[n,2],col=3,pch=16)

# Annealing

n <- 10000 ;p=2; X <- matrix(nr= n, nc=p,data=NA);totalACP=0

X[1,] <- runif(n=2,min=-3,max=3) # Init

#X[1,] = rnorm(n=2,m=0,sd=1);

for(i in 1:(n-1)) # Iterate random update

{

t = 1/log(i)

Y = rnorm(n=2,m=0,sd=.1) # propose candidate value

Xstar = X[i,] + Y

if (Xstar<-3 || Xstar>3) {

Y = rnorm(n=2,m=0,sd=.1);

Xstar = X[i,] + Y

}

delta = h((Xstar[1]),(Xstar[2]))-h(X[i,1],X[i,2])

p = min(exp(delta/t),1) #exp of slop,only accept positive slop, looking for when h(Xstar) is bigger than h(X). get big prob to move to Xstar.As h(X) gets bigger, X gets centered to 0. the smaller the exp, the smaller the prob that X moves to Xstar.

accept = as.logical(rbinom(n=1,size=1,prob=p))

#X[i+1, accept] = Xstar[accept]

#X[i+1,!accept] = X[i,!accept]

X[i+1,] = ifelse(accept, Xstar, X[i,])

totalACP[i]=accept

}

sum(totalACP)

#small prob = stay as X, big prob = move to Xstar.

summary(X)

plot(X,type='b')

hist(X[,1],prob=T,breaks=50)

hist(X[,2],prob=T,breaks=50)

lines(density(X),col=2)

acf(X)

png('Annealing.png')

H = outer(x,y,h)

image(x,y,H)

lines(X,type='l',lwd=.8,col=3)

points(X[1,1],X[1,2],col=4,pch=16)

#points(X[,1],X[,2],col=1,pch=8)

points(X[n,1],X[n,2],col=3,pch=16)

dev.off()