02433 Stochastic Simulations

Exercise Report

Carlos Monteserin Sanchez, s141560

Kgs. Lyngby June 14, 2015

DTU Compute
Technical University of Denmark

Contents

1 Uniform Random Numbers **1**

2 Non-uniform Random Numbers **4**
2.1 Exercise 1 . . . 4
2.2 Exercise 2 . . . 4
2.3 Exercise 3 . . . 5
2.4 Exercise 4 . . . 6
2.5 Exercise 5 . . . 7
2.6 Exercise 6 . . . 8
2.7 Exercise 7 . . . 9

3 Markov chain Monte Carlo methods **11**
3.1 Exercise 1 . . . 11
3.2 Exercise 2 . . . 11
3.3 Exercise 3 . . . 13
3.4 Exercise 4 . . . 17
3.5 Exercise 5 . . . 19
3.6 Exercise 6 . . . 19

4 Monte Carlo Integration methods **21**
4.1 Exercise 1 . . . 21
4.2 Exercise 2 . . . 22

5 Monte Carlo Optimization methods **24**
5.1 Exercise 3 . . . 25
5.2 Exercise 4 . . . 26

Appendix **i**
A Day 1 . . . i
B Day 2 . . . ii
C Day 4 . . . viii
D Day 5 . . . xv

1 Uniform Random Numbers

Linear Congruential method is a tool that allows us to generate independent random numbers uniformly distributed in the interval [0,1]. In this exercise we implement the method, using the parameter values $a = 16807$, $b = 0$, $M = 2147483647$ and outcome a vector X=$\{x_i\}$ containing 1000 random numbers.

Fixed a, b and M the resulting sequence is completely determined by the initial value x_1 (seed). In order to see different realizations with this parameter values we choose x_1 randomly (in [0,M-1]). In the following we test some statistical properties of the sampled vector X.

1) We expect X entries to be uniformly distributed in the interval [0,1]. Figure 1 shows the normalised histogram of X together with the theoretical pdf U[0,1] (red line).

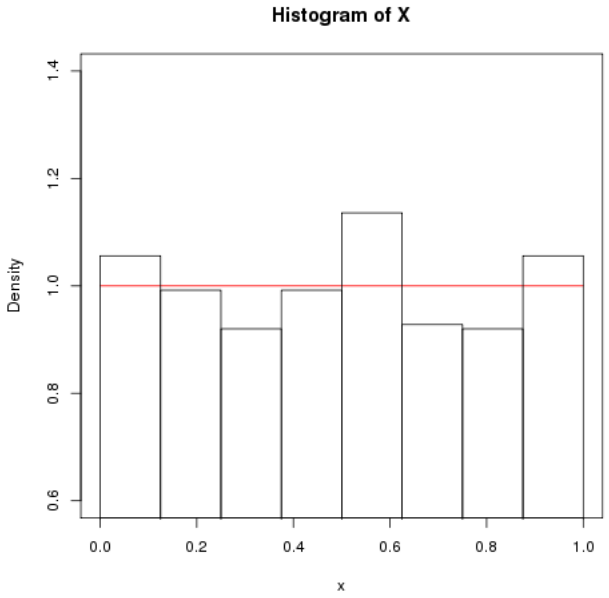

Figure 1: Histogram of X together with theoretical pdf U[0,1]

2) We expect X entries to be independent realizations. If we find any sign of correlation in the data, we can discard independence. We make here two tests to check the hypothesis H_0: X entries are independent.

   We first analyse the autocorrelation function (acf command in R), that analyse the correlations in the data entries x_i and x_{i+p} for different p values (p is the time lag). Left panel of figure 2 shows the correlogram, where all the modes lie inside the confidence interval. We cannot reject the hypothesis.

Right panel of figure 2 shows the spatial position of the points in the form (x_i,x_{i+2}) where $i = 1,...,n-2$. The points look uniformly distributed, there are neither alignments or signs of correlation. These points are used by acf when lag=2 and we can see in left panel that the correlation of this mode is close to 0.

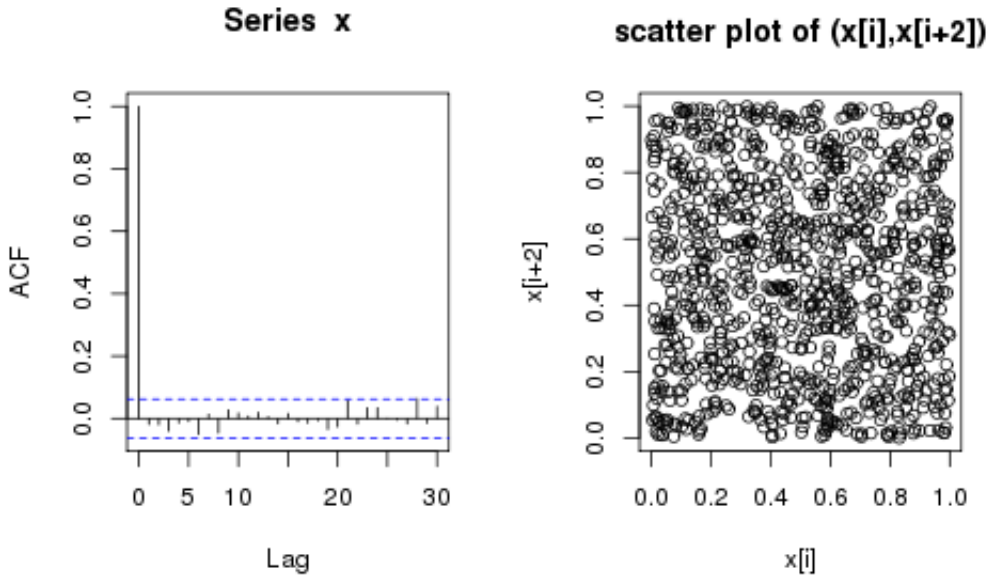

Figure 2: Left panel: correlogram of X. Right panel:scatter plot of the points (x_i,x_{i+2})

3) Now our hypothesis H_0 is X follows a uniform distribution in [0,1], we test the new hypothesis using a Kolmogorov-Smirnov test (command ks.test in R).

This well-known method calculates the maximum distance D between theoretical and empirical cumulative functions (see equation (1.1)), and produces a significance value p in [0,1]. Small p values mean that the distance is very big to be produced by a uniformly distributed sample. On the contrary high values of p mean that D is compatible with the theoretical distribution.

$$\begin{aligned} F_{teo}(x) &= \int_0^x dt = x \qquad\qquad x \in [0,1] \\ F_{emp}(x) &= \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,x]}(x_i) \end{aligned} \tag{1.1}$$

where $I_{(-\infty,x]}(y)$ is an indicator function, i.e. 1 if y $\leq$ x and 0 if y>x. Figure 3 shows the agreement between theoretical (red) and empirical (black) cumulative functions. The maximum distance is D= 0.0191, and p=0.8603.

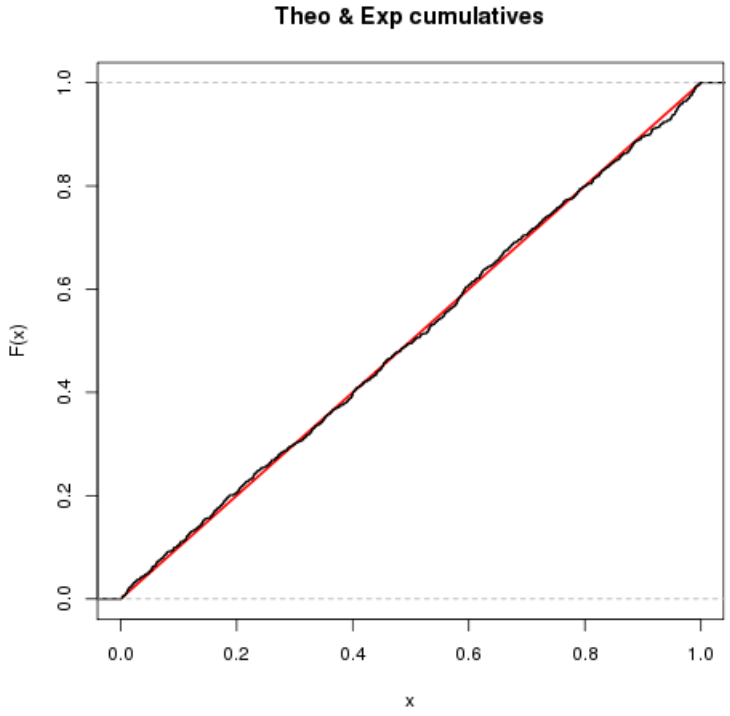

Figure 3: Theoretical (red line) and empirical (black) cumulative distribution functions

2 Non-uniform Random Numbers

2.1 Exercise 1

C(0,1) probability and cumulative distribution functions are given by:

$$\begin{aligned} f(x) &= \frac{1}{\pi\left(1+x^2\right)} \\ F(x) &= \frac{1}{2} + \frac{1}{\pi}\arctan(x) \end{aligned} \tag{2.1}$$

We can apply inversion method in this case because the cdf of the target distribution is easy to invert. Lets name u=F(x) then:

$$x = F^{-1}(u) = \tan\left(\pi\left(u - \frac{1}{2}\right)\right) \tag{2.2}$$

We generate n=10000 random numbers uniformly in [0,1] (u=runif(n,0,1)), and then $y = F^{-1}(u)$ will follow the target distribution f(x). Figure 4 shows the histogram of y together with the theoretical pdf of Cauchy(0,1) (red line).

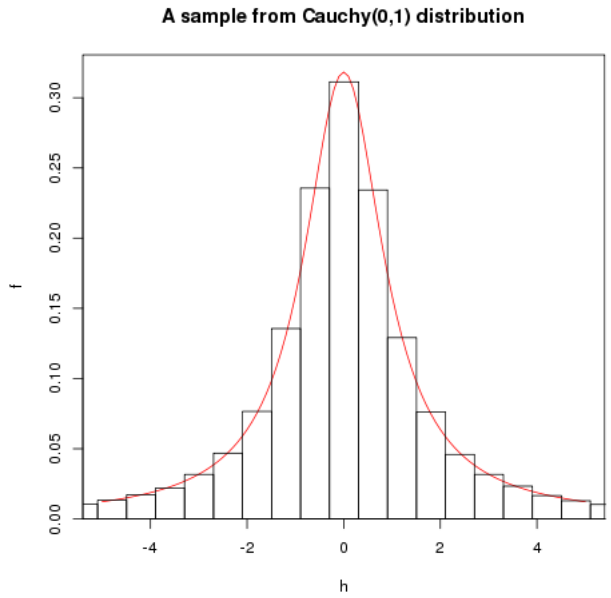

Figure 4: Histogram of y together with theoretical pdf C(0,1)

2.2 Exercise 2

In the rejection method we have a target pdf f(x) (β(2,5) in this case) that we don't know how to sample and also an auxiliar distribution g(x) (U[0,1] in this case) that we do know how to sample.

We define the set $S_g \in \mathbb{R}^2$ as:

$$S_g = \{(x,y) \in \mathbb{R}^2 \ / \ x \in [0,1] \ and \ y \in [0,K * g(x)]\} \tag{2.3}$$

where K must be chosen such that K*g(x) > f(x) for all x. The idea of rejection method is generating random points (x,y) in S_g and then keep those ones for which $y < f(x)$ (throwing away the rest). The points under the target pdf are by construction uniformly distributed thus, the x component of these points must follow the target distribution.

We generate n=10000 points in S_g as follows:

1) x following $\beta(2,5)$ distribution takes values in the interval [0,1], Therefore we generate the x component of the points using runif(n,0,1).

2) $\beta(2,5)$ distribution peaks around 0.2 taking the supreme value: max(f(x))=2.5. Thus we define K=2.6 that guarantees $Kg(x) > f(x) \ \forall \ x \in [0,1]$

3) for a given x component, we define the corresponding y component as a random number in [0,K*g(x)]. Therefore every point (x,y) belongs to S_g.

Now we keep those points s.t. $y < f(x)$ (red points in left panel of figure 5). The x-component of these points follow the target distribution $\beta(2,5)$, right panel of figure 5 shows the normalised histogram of these x components together with the beta distribution

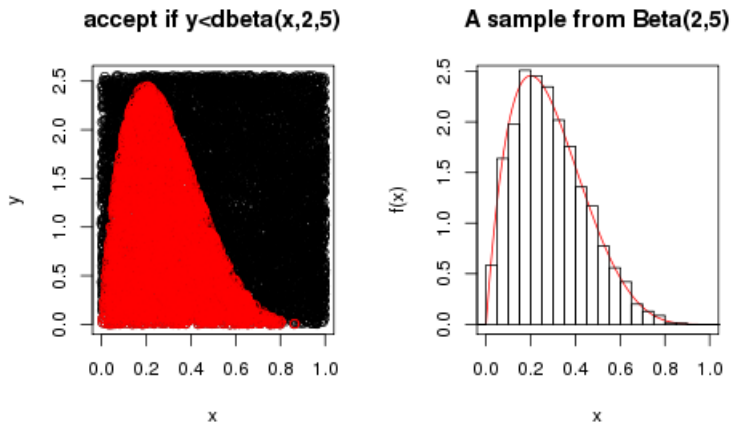

Figure 5: Left panel: Accepted and rejected points by the method. Right panel: Histogram of x-component of accepted points together with $\beta(2,5)$

2.3 Exercise 3

In this exercise the rejection method is used targeting a normal distribution (f(x)=N(0,1)) and choosing as auxiliar one the double exponential (with $\alpha = 1$). Therefore we have

analytical expressions for f and g:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-0.5x^2}$$
$$g(x) = \frac{1}{2} e^{-|x|} \tag{2.4}$$

The first step is calculate the scaling factor K. N(0,1) decays fast to zero when $|x|$ grows. The probability to obtain a number out of the interval [-4,4] is negligible. We define a vector h=seq(-4,4,0.01) and evaluate f(h) and g(h). we define the vector fog=f(h)/g(h) using the element-wise division. We set K=max(fog)+0.5, this value ensures that $Kg(x) > f(x)\ \forall\ x \in [-4,4]$. Figure 6 shows f(x) (green line) together with g(x) and Kg(x) (red lines).

Following the procedure described in exercise 2, we generate n=10000 points (x,y)

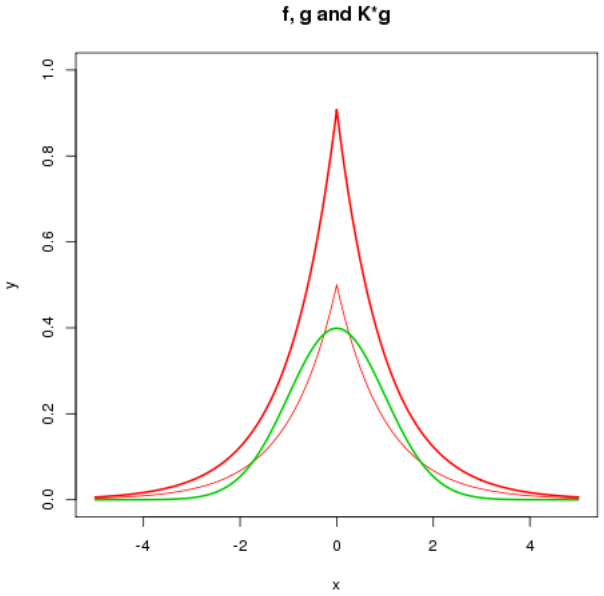

Figure 6: f(x) (green line) together with g(x) and Kg(x) (red lines)

uniformly distributed in S_g. We sample the double exponential using rexp command in R as follows: x = c(-rexp(n/2,rate=1), rexp(n/2,rate=1)). Afterwards, we keep those points satisfying y<f(x) and reject the rest (See left panel of figure 7). The x components of accepted points follows the targeted normal distribution (right panel of figure 7).

2.4 Exercise 4

Everytime we throw a needle the tail falls in a plank at a position (x,y). On the one hand, we assume that the planks are infinity long and we look for the intersection of the needles with the vertical lines (plank joints), therefore, we just care about the x component of

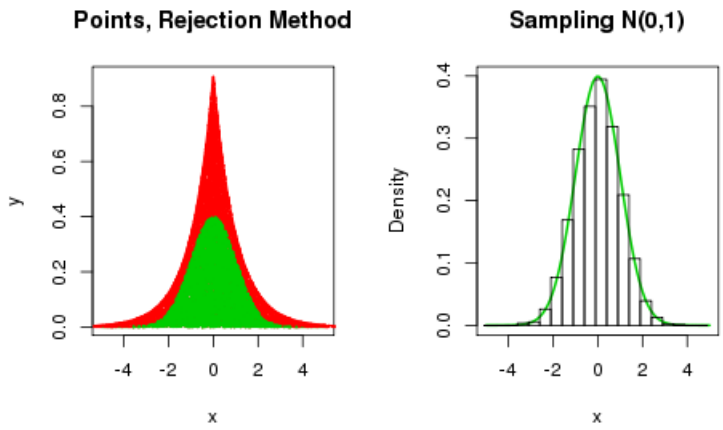

Figure 7: Left panel: Accepted and rejected points by the method. Right panel: Histogram of x-component of accepted points together with $N(0,1)$

the tail position. On the other hand, we name θ the angle that the needle form with the horizontal axis.

We consider that a needle throwing can be simulated using random numbers, in the following way:

1) x position of the needle's tail is chosen randomly in [0,L] and each position is equally probable, i.e. x=runif(n=1,0,L).

2) θ is chosen randomly in [0,2π] and each angle is equally probable, i.e. θ=runif(n=1,0,2π)

Given the pair (x,θ) we determine if the needle intersects or not a vertical line calculating the position of the needle's pin $x_p = x + L\cos(\theta)$. If $0 < x_p <$ L then the needle is completely inside the plank otherwise the needle intersects a vertical line.

We repeat the experiment using n=10000, and count the number of intersections m<n. We estimate the probability of intersection as p= m/n = 0.6429. L=1 has been chosen in our analysis, but any value might produce the same result.

2.5 Exercise 5

We use Poisson(10) distribution to model the offer we get for the car, and we count the number of offers until we get one over $x_1 = 12$. We repeat the experiment n=1000 times and construct a vector t containing these times (Left panel of figure 8 shows the normalised histogram of t), based on this analysis we estimate the waiting time as the average T=3.386.

We can say this analysis is conditioned by the first offer $x_1 = 12$. We can repeat the experiment in average over all possible x_1 values under the assumption that x_i's are

i.i.d Poisson(10). For this unconditionally case we simply use x_1=rpois(n=1,10), since E(x_1)=10<12, we expect the waiting time to be shorter than in the conditionally case. Right panel of figure 8 shows the normalised histogram of t for this unconditionally experiment. The averaged waiting time is in this case T=1.856.

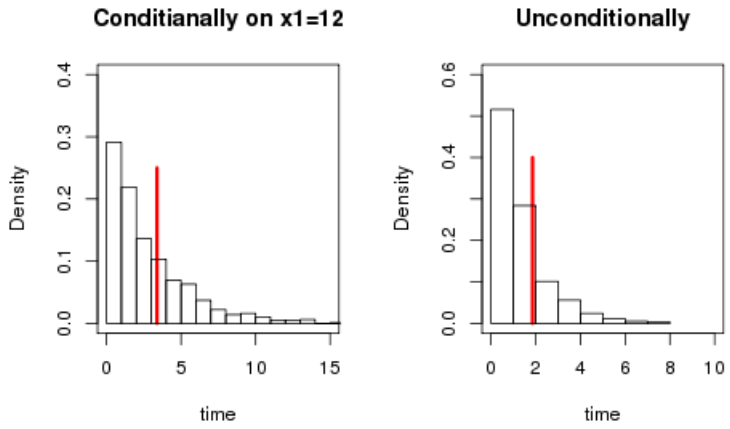

Figure 8: Normalised histogram of waiting times for the cases: conditionally on $x_1 = 12$ (left panel) and unconditionally (right panel)

2.6 Exercise 6

We can use bootstrap technique to evaluate the variance in the estimation of the median of a Poisson distribution. We first construct a vector X=$\{x_i\}$ containing n=200 realizations of Poisson(10) and estimate the median as the average value E[x]=$\sum_{i=1}^{n} x_i$= 9.72. In the following we refer to this vector X as our data set.

In order to estimate the variance of this median value, we might repeat the experiment several times collect the mean of each realization in a vector M_{th} and analyse its mean and variance. Alternatively we can sample the original data set (using sample command in R) to obtain a new set of experiments, collect the means in the vector M_{bt} and analyse mean and variance.

We make these two analysis repeating the experiment B=1000 times, resulting:

$$\begin{aligned} \hat{M}_{th} &= 9.994571 \qquad & Var(M_{th}) &= 0.0486822 \\ \hat{M}_{bt} &= 9.719475 \qquad & Var(M_{bt}) &= 0.0483515 \end{aligned} \tag{2.5}$$

Additionally we estimate the variance using the bootstrap estimation of the variance:

$$V(M_{bt}) = \frac{1}{B}\sum_{i=1}^{B}\left(M_{bt}^{i} - \left(\frac{1}{B}\sum_{j=1}^{B} M_{bt}^{j}\right)\right) \tag{2.6}$$

resulting the value $V(M_{bt}) = 0.04834665$.

2.7 Exercise 7

Read the instructions in lecture notes we simulate a sample of n=50000 points y=(y_1,y_2) from a bi-variate normal distribution with covariance matrix $\Sigma = [1\ 0.7; 0.7\ 1]$. The process is as follows:

1) We create 2 independent N(0,1) distributed vectors of size n that we save in a matrix x=(x_1,x_2). x points follow a bi-variate normal with covariance matrix I_2.

2) We obtain y by applying a linear transformation on x, i.e. y=Lx, where L is the lower triangular matrix corresponding to the Cholesky factorization of Σ.

3) We check that var(y) is a matrix close to the original Σ and compare the scatter plot of both samples x and y (see figure 9)

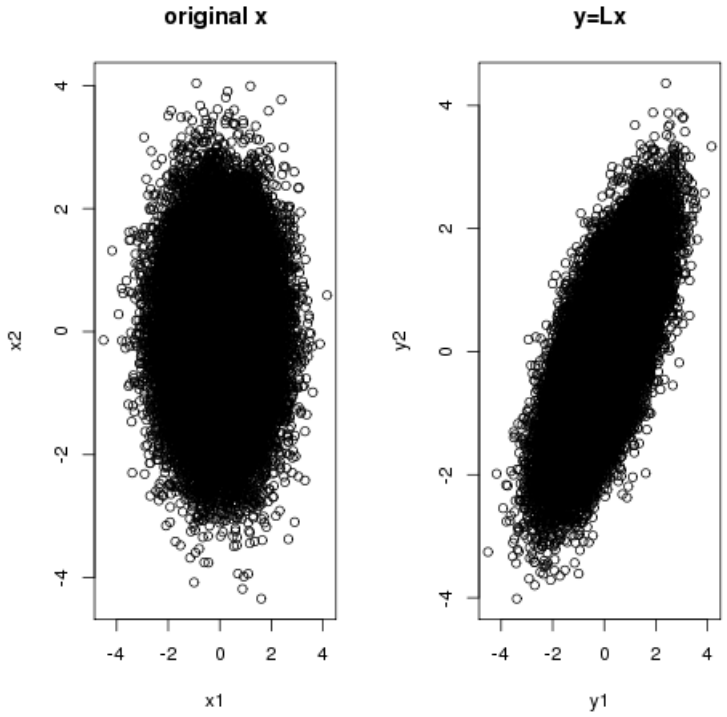

Figure 9: Scatter plot of sampled variables x (left panel) and y (right panel) both of them of size n=50000.

We now extract those points of y=(y_1,y_2) sample where $0.49 < y_2 < 0.51$ (361 points in total) and save their x-component y_1 in the vector z. We use mean and var commands in R to estimate the mean and variance resulting:

$$mean(z) = 0.293 \qquad var(z) = 0.554 \tag{2.7}$$

Finally we plot the histogram of z and compare it with:

1) N(mean(z),sqrt(var(z))) (see right panel of figure 10)

2) the bi-variate normal distribution taking y_2=0.5, i.e. f($y_1|y_2$=0.5). Notice that we need to normalize this function to get the figure we show in right panel of 10.

$$f(y_1|y_2 = 0.5) = \frac{1}{2\pi|\Sigma|} e^{\frac{-A}{8}} e^{-\frac{1}{2}\left(\frac{1}{2}y_1(C+B)+y_1^2 D\right)} \tag{2.8}$$

where we named $\Sigma^{-1} = [A\ B; C\ D]$.

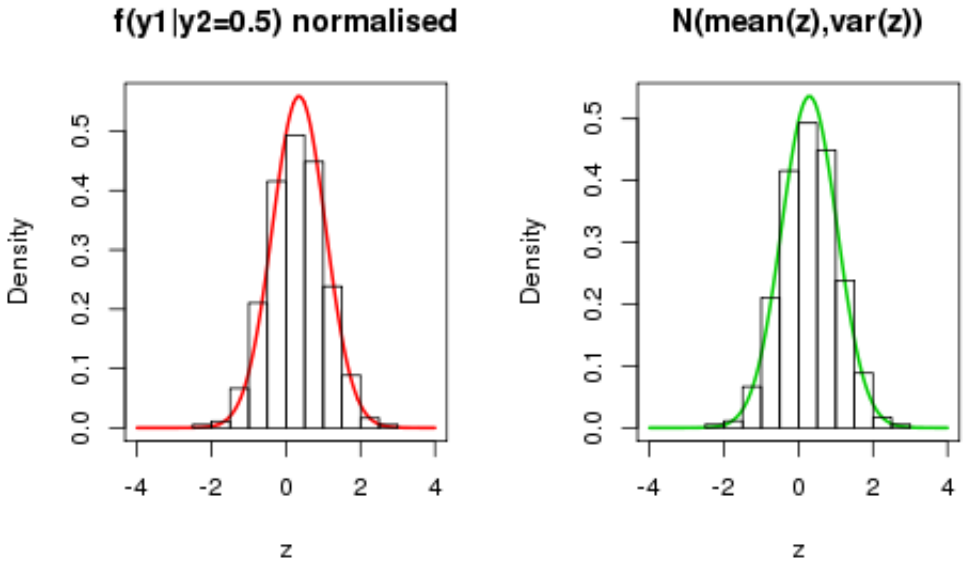

Figure 10: Normalised histogram of z compared with: f($y_1|y_2$=0.5) (left panel). N(mean(z),sqrt(var(z))) (right panel).

3 Markov chain Monte Carlo methods

3.1 Exercise 1

We want to simulate a Markov chain X=$\{x_i\}$ whose entries belong to the set S={0,1}. We first initialise the sequence randomly in S (x_1=rbinom(n=1,size=1,prob=0.5)) and consider the transition kernel:

$$\begin{aligned} P\left(x_i = 1 | x_{i-1} = 0\right) &= \frac{1}{\sqrt{2}} \\ P\left(x_i = 1 | x_{i-1} = 1\right) &= \frac{1}{\pi} \end{aligned} \tag{3.1}$$

a sequence containing n=1000 entries is constructed and its entries are shown in figure 11. We can see the entries change from 0 to 1 when we move in the iterations.

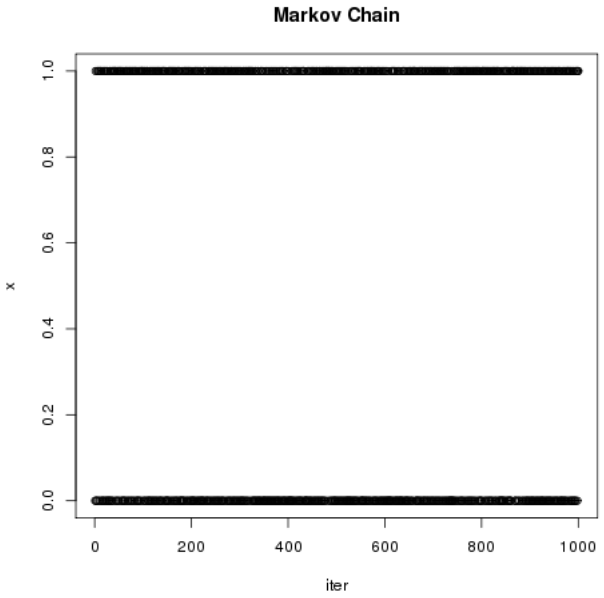

Figure 11: Markov sequence with n=1000 entries in S={0,1}.

3.2 Exercise 2

Taking the metropolis algorithm given in the lecture notes, we construct a Markov chain of n=10000 elements targeting Cauchy(0,1) and using N(0,1) as proposal distribution. Left panel of figure 12 shows the complete chain and right panel compares the normalised histogram of the last 1000 entries with C(0,1).

It could happen that a candidate value Y is on the tails of the proposal, in this scenario f(Y) $\gg$ dnorm(Y,0,1) because the Gaussian decays faster than Cauchy(0,1). Therefore,

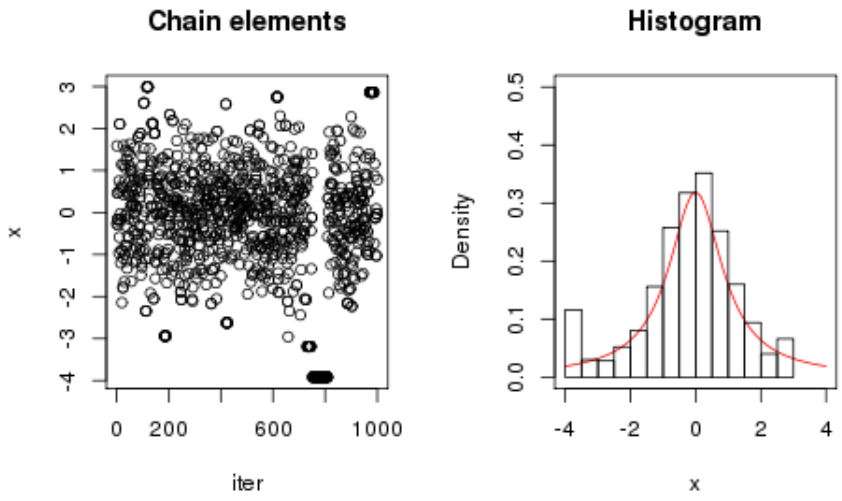

Figure 12: Left panel: complete Markov chain. Right panel: normalised histogram corresponding to the last 1000 entries of the chain together with C(0,1) (red line).

the ratio R will be greater than 1 and the candidate will be accepted.

In the next iteration we will have the opposite situation. Since f(x_i) $\gg$ dnorm(x_i,0,1), R will be a small number and the algorithm might produce many consecutive rejections (see left panel of figure 12). As a consequence of this problematic we can see for some realizations of this Markov chain that the corresponding histogram presents peaks in the tails (see right panel of figure 12)

In addition to this, figure 13 shows the correlogram corresponding to the last entries of the chain showing non-negligible correlations even for big lag values. Perhaps N(0,1) is not the best proposal to target Cauchy(0,1) using metropolis algorithm. In the following exercise we implement Metropolis-Hastings algorithm where the proposal distribution depends on the current state of the chain.

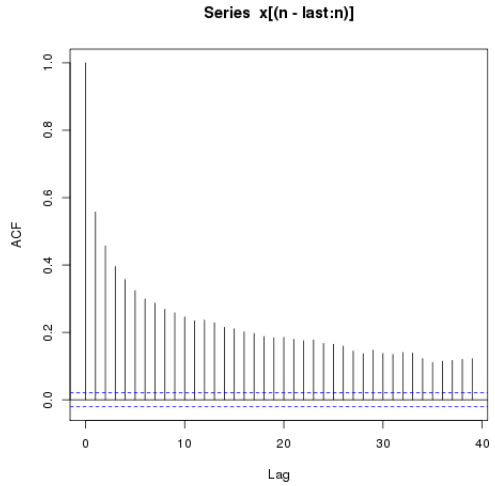

Figure 13: Correlogram corresponding to the last 1000 entries of the chain.

3.3 Exercise 3

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],σ=1) as auxiliar pdf and repeat the process we performed in previous exercise.:

1) Construct a chain of n=10000 elements using the new proposal.

2) Analysis of the last 1000 entries.

2a) Plotting chain elements (see figure 14)

2b) Constructing the corresponding histogram and correlogram (see figure 15)

We can see in figure 14 that the new proposal is producing less rejections in the algorithm than in exercise 2. Notice that the points have a continuous appearence, meaning that the next element of the chain is close to the current one in contrast with the jumps observed in left panel of figure 12. This is due to the new proposal's dependence on current entry. The correlogram indicates that the correlations are even bigger than for the ordinary metropolis.

We might expect that decreasing the value of σ in the proposal, the following candidate will be likely closer to the current entry, producing a more continuous look for the points and consequently higher level of correlations in the correlogram (see figures 16 and 17).

On the contrary increasing the value of σ, we allow the new candidates to be further away from the current value, producing less correlated sequences (see figures 18 and 19).

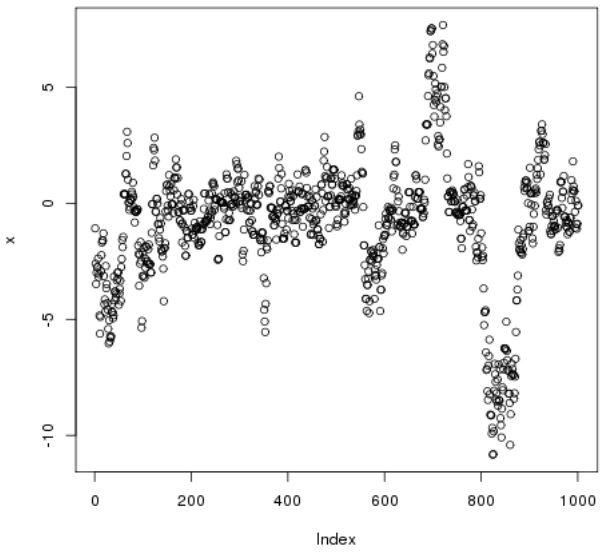

Figure 14: Last elements of the chain using σ=1.

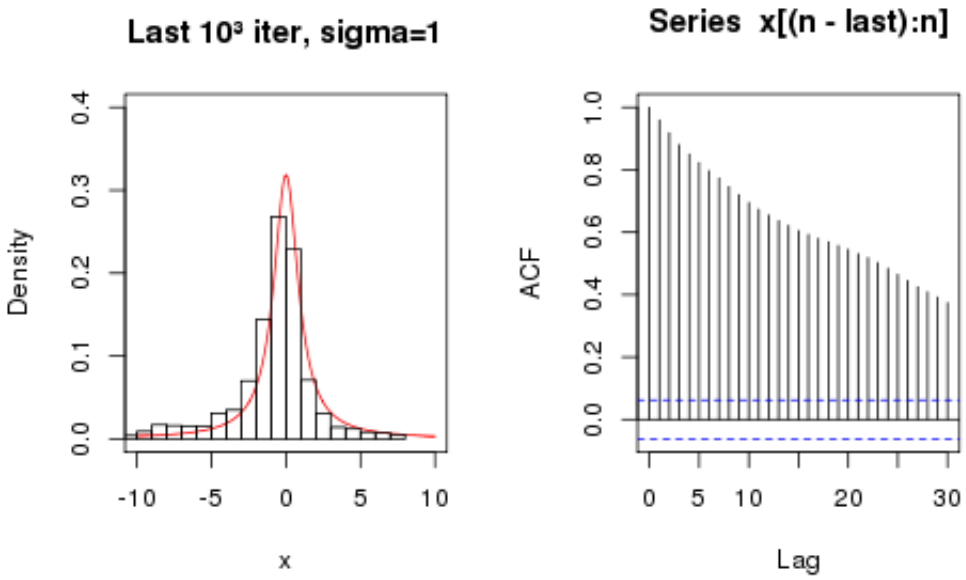

Figure 15: Left panel: normalised histogram corresponding to the last 1000 entries of the chain together with C(0,1) (case σ=1). Right panel:corresponding correlogram.

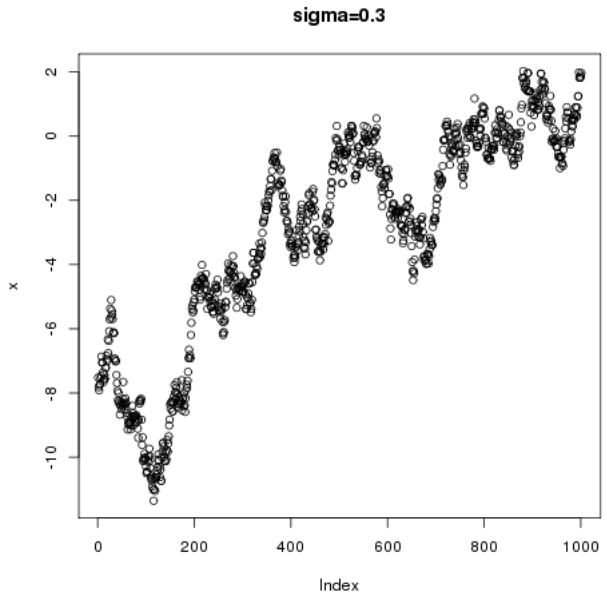

Figure 16: Last elements of the chain using σ=0.3

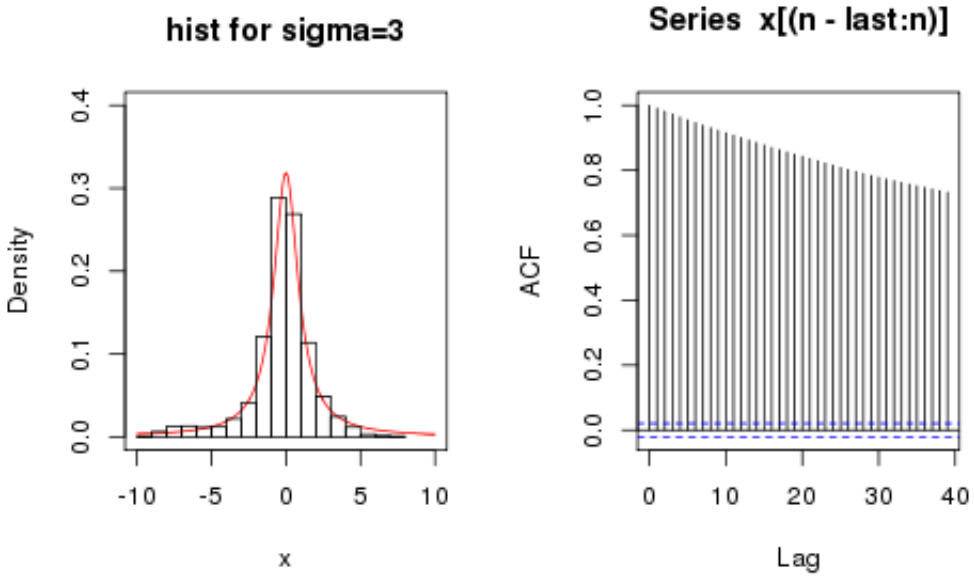

Figure 17: Left panel: normalised histogram using σ=0.3. Right panel:corresponding correlogram.

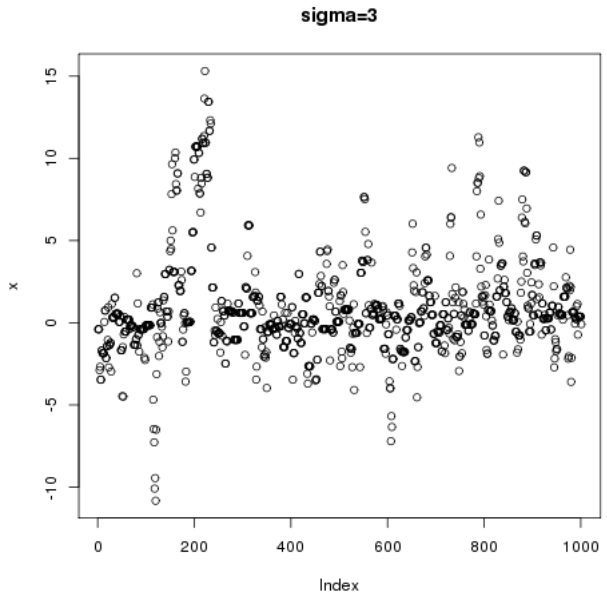

Figure 18: Last elements of the chain using σ=3

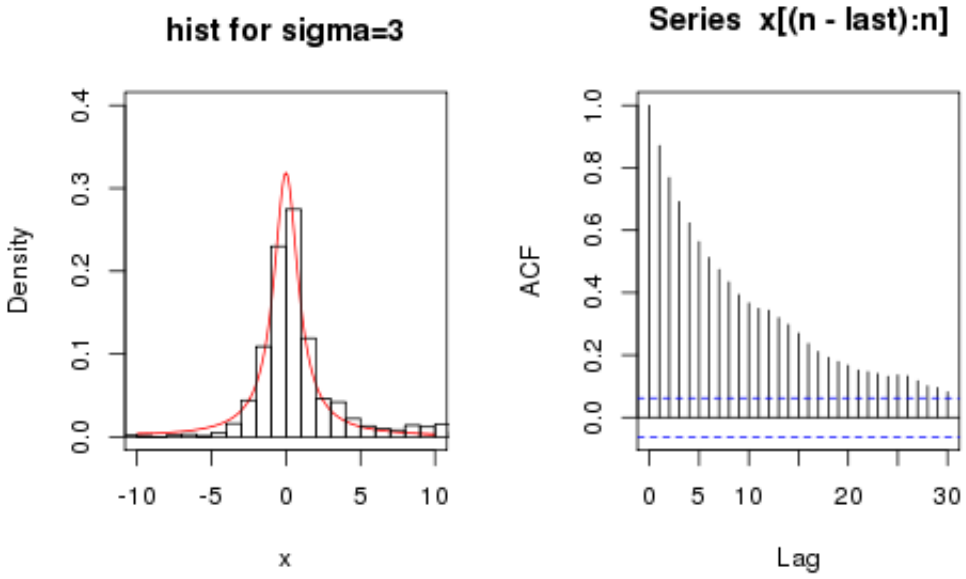

Figure 19: Left panel: normalised histogram using σ=3. Right panel:corresponding correlogram.

3.4 Exercise 4

We are requested to simulate a sample from $\beta(1/2,1/2)$ using two different techniques: rejection method and M-H algorithm.

for the rejection method we use as auxiliar distribution U[0,1] and follow the procedure detailed in the second exercise of the second day. Notice that $\beta(1/2,1/2)$ diverges for x=0 and x=1, so we must be careful with the scaling parameter K. Figure 20 shows some results of the outcome sequence.

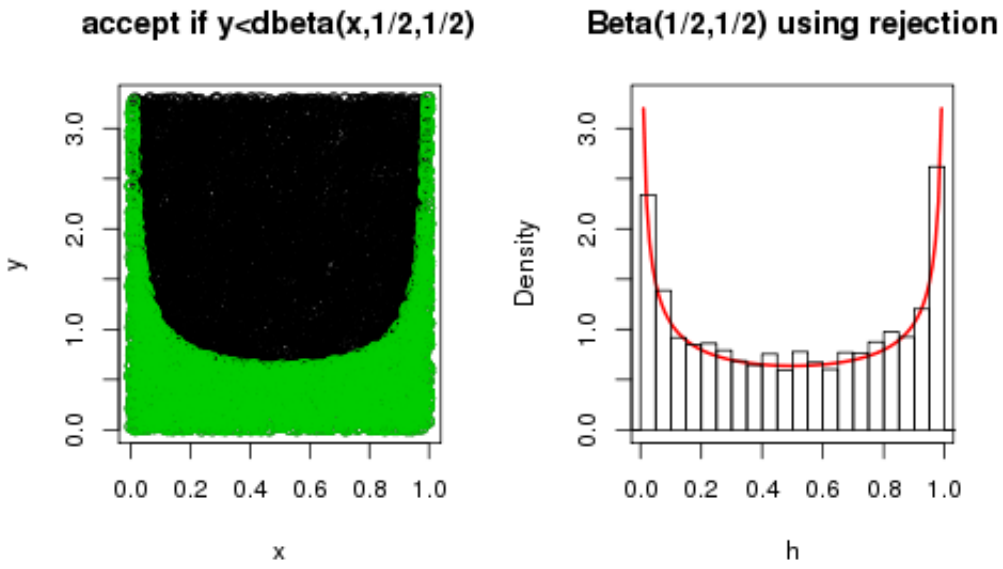

Figure 20: Left panel: Accepted (green) and rejected (black) points. Right panel: normalised histogram of x-components of accepted points. $\beta(1/2,1/2)$ pdf is paint in red.

Regarding M-H algorithm we use $\sigma = 1$ for the proposal. Figure 21 shows some results of the last elements of the Markov chain.

Besides of computational efforts, both methods produce sequences compatible with the target distribution. Notice that only the rejection method is producing a uncorrelated sequence. Figure 22 shows the correlogram corresponding to each technique.

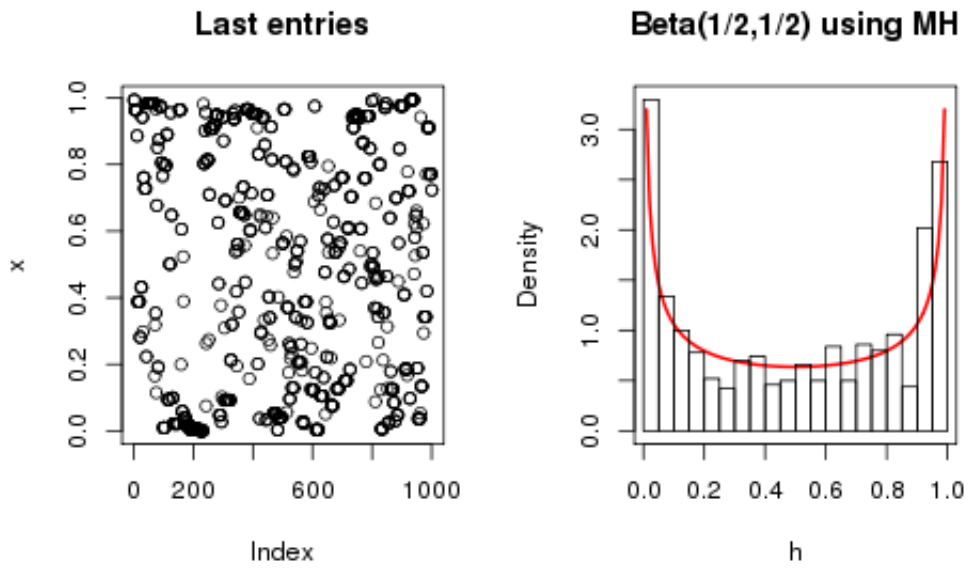

Figure 21: Left panel: Last entries of the chain. Right panel: normalised histogram.

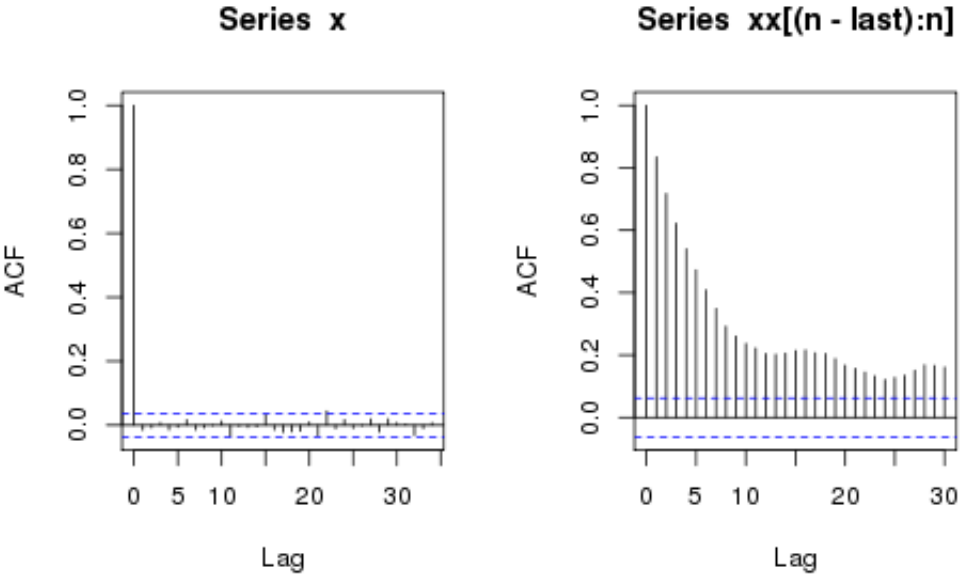

Figure 22: Correlograms corresponding to rejection method (left) and M-H algorithm (right).

3.5 Exercise 5

The correlograms corresponding to the MCMC constructed in previous exercises show correlations in the entries. After the burn-in period, the elements of the sequence follow the target distribution but they are not independent.

A simple procedure to obtain independence consist on thinning the chain. The idea is use the original sequence X=$\{x_i\}$ (after burn-in) to construct a shorter one $\tilde{X} = \{\tilde{x}_j\}$ defined by:

$$\tilde{x}_j = x_{1+(j-1)s} \tag{3.2}$$

In words, we keep the first entry discard the following s-1 elements, take the next one and so on. The correlograms in previous exercises show a decay in the correlations when the time lag grows. We can use this tool to investigate which value of s should be used to thin a chain. In general we are discarding many points meaning that if we want a thinned chain of m elements we have to simulate (s-1)m numbers (after burn-in period) in the original chain.

In exercise 4 we notice that Metropolis-Hastings algorithm produces a sequence less correlated when a big value of σ is chosen for the proposal. Therefore, the corresponding thinned chain can be obtained less expensively in computational terms.

3.6 Exercise 6

For this exercise we implemented the Gibbs sampler following the pseudo-code given in lecture notes. We target a bivariate normal with covariance matrix Σ=[1 ρ ; ρ 1] and consider the conditional distribution Y|X=x $\sim N(\rho$x,1-$\rho^2)$.

choosing $\rho = 0.7$, n=10000 points are sampled to construct a matrix z containing only the last 1000 entries. The covariance matrix of z is close to the desired Σ and the scatter plot of the sampled points (See figure 23) looks compatible with a bi-variate normal. Both tests get better when we include more elements of the chain in z.

$$var(z) = \begin{bmatrix} 1.041 & 0.695 \\ 0.695 & 1.013 \end{bmatrix} \tag{3.3}$$

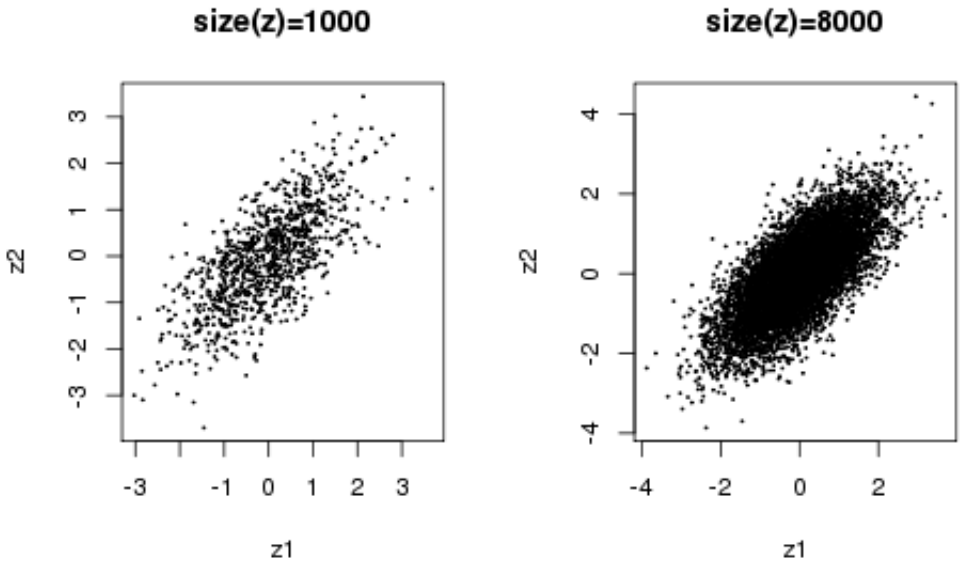

Figure 23: Scatter plot of z when we just keep last 1000 entries of the 2D Markov chain (left panel). Same keeping the last 80% of the entries (right).

4 Monte Carlo Integration methods

One in a number of applications of MCMC tecniques is solving integrals of high dimensional functions g:$\mathbb{R}^d \longrightarrow \mathbb{R}$. The idea consist in rewriting the integral as the expected value of a quantity h(x) using a suitable pdf f(x).

$$I = \int_{\mathbb{R}^d} g(x)dx = \int_{\mathbb{R}^d} \frac{g(x)}{f(x)} f(x)dx = \int_{\mathbb{R}^d} h(x)f(x)dx \tag{4.1}$$

In stochastic integration the dimension of the integral is not a problem as long as we can sample numbers x following f. MCMC technique can be used to sample f and construct a sequence $X = \{x_i\}$ of random numbers in $\mathbb{R}^d$. The integral can be estimated as the expected value of h as follows:

$$\int_{\mathbb{R}^d} h(x)f(x)dx = E[h] = \frac{1}{n}\sum_i h(x_i) = \hat{I} \tag{4.2}$$

If the random numbers $\{x_1,x_2,...,x_n\}$ are i.i.d variables, we can produce an estimation of the accuracy of our result:

$$V[\hat{I}] = \int_{\mathbb{R}^d} (h(x) - I)^2 f(x)dx = E[\left(h - \hat{I}\right)^2] = \frac{1}{n^2}\sum_i \left(h(x_i) - \hat{I}\right)^2 = \hat{V[}\,\hat{I}] \tag{4.3}$$

If MCMC methods are used to sample f we might be aware to the correlations in the outcome sequence (see exercise 5 of previous section).

4.1 Exercise 1

Lets consider the following one-dimensional integral:

$$I = \int_0^1 g(x)dx = \int_0^1 \left[\cos(50x) + \sin(20x)\right]^2 dx \tag{4.4}$$

Figure 24 shows the curve we are about to integrate. Standard integration methods produce a solution I_{true}= 0.9652009 which is relatively expensive to compute due the sharpness of the function (we used integrate command in R).

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:

$$\hat{I} = E[I] = \frac{1}{n}\sum_i g(x_i) \tag{4.5}$$

where X=$\{x_i\}$ is sampled from U[0,1]. Using n=10^5 numbers we obtain $\hat{I}$= 0.965215 which is close to the true value.

We investigate the dependence of $|\hat{I} - I_{true}|$ with the number of simulations n used for

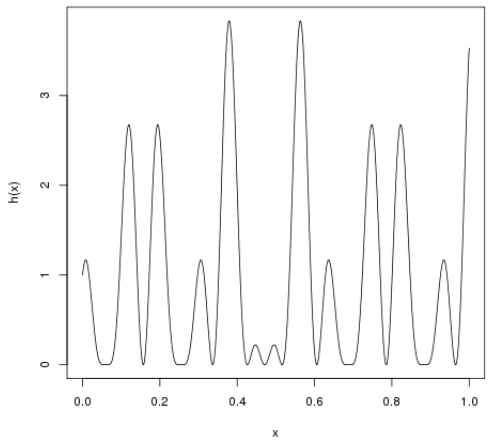

Figure 24: h(x) in the interval [0,1].

the stochastic estimation. Figure 25 shows the loglog scaled convergence plot. The resulting line scales like $O(1/\sqrt{n})$ does. This result seems compatible with the expected decrease rate 1/n of the variance commented in lecture notes. Notice that X=$\{x_i\}$ are i.i.d. variables generated with runif.

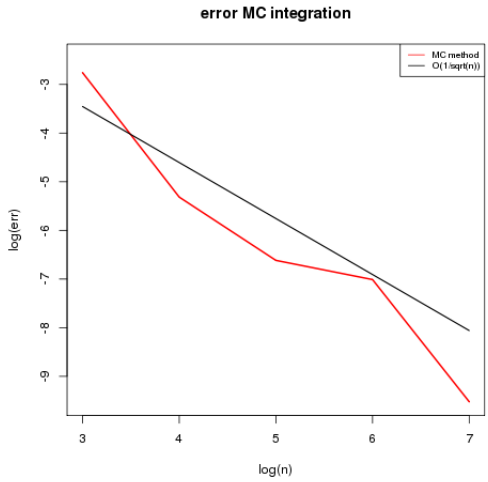

Figure 25: Loglog scaled convergence test of $|\hat{I} - I_{true}|$ for different n values. Auxiliar line $O(1/\sqrt{n})$ is painted in black.

4.2 Exercise 2

The unit Ball B_d in $\mathbb{R}^d$ is the set of points x=(x_1,...,x_d) such that $\sum_i x_i^2 < 1$. The goal of this exercise is calculate the hyper-volume V_d of the unit ball by stochastic integration.

We can define the hyper-cube C_d as the set of points x=(x_1,...,x_d) such that $|x_i| < 1$. It can be shown that $B_d \subset C_d \; \forall \; d$. We know how to simulate points uniformly distributed in the hyper-cube. If we sample n numbers in the cube and count m of them lying inside the unit ball we can say that the ratio n/m is an estimation of the ratio of the volumes $V_d/|C_d|$ (See figure 26). Notice that an hypercube of side 2 has $|C_d| = 2^d$.

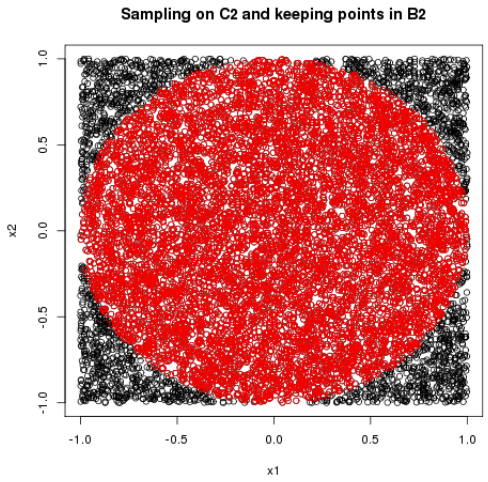

Figure 26: 10^4 points uniformly distributed in C_2, red ones lie inside B_2.

This intuitive result can be expressed in terms of an integral using the indicator functions[1]of ball $I_{B_d}(x)$ and cube $I_{C_d}(x)$:

$$V_d = \int_{\mathbb{R}^d} I_{B_d}(x)dx = |C_d| \int_{\mathbb{R}^d} I_{B_d}(x) \frac{I_{C_d}(x)}{|C_d|} dx \tag{4.6}$$

Following the notation previously introduced:

$$f(x) = \frac{I_{C_d}(x)}{|C_d|} \qquad h(x) = |C_d| I_{B_d}(x) \tag{4.7}$$

We can sample n points $x_i \in \mathbb{R}^d$ following f, just composing vectors of d components (i.e. X[i,]=runif(d,-1,1) i=1,...,n). The integral is estimated as the expected value of h (i.e. $\hat{I}$=E[h]=1/n $\sum_i h(x_i)$).

Table 1 shows the results when we average on B=100 experiments (n=10^4 points are simulated in each one). We investigate d={2,3,10}). The stochastic integration agrees the true value corresponding to $V_d = \pi^{d/2}/\Gamma(1 + d/2)$.

[1]Notice that $\int_{\mathbb{R}^d} I_{B_d}(x)dx = \int_{\mathbb{R}^d} I_{B_d}(x) I_{C_d}(x)dx$ because they only are 1 simultaneously inside the Ball.

d	V_{true}	V_d (averaged on 100 exp)
2	3.14159	3.139 ± 0.0159
3	4.18879	4.195 ± 0.0431
10	2.55016	2.562 ± 0.5399

Table 1: Volume of the unit Ball V_d for different d values.

Notice that the n points are i.i.d variables, and we can use the estimator of the variance above introduced (this is a 1 simulation estimation of the variance). In table 2 we compare this result with the variance computed on the B=100 experiments, we can see the goodness of the estimation using only 1 simulation.

d	Var(V_d) (1sim estimation)	Var(V_d) (averaged on 100 exp)	$V_d/\|C_d\|$
2	2.617e-04	2.527e-04	0.7854
3	1.595e-03	1.861e-03	0.5236
10	2.824e-01	2.915e-01	0.245e-02

Table 2: Variance of the estimation V_d.

The volume is not strictly increasing with d. However, the variance of the estimation grows with the dimension of the ball. This can be explained in terms of the ratio $V_d/|C_d|$ (last column in table 2). When d grows much more points lie outside of the ball (under 0.25% for d=10), and we lose accuracy in our estimation.

We could be more precise for high d values by choosing a more appropriate distribution for the sampling. We are suggested to simulate using a multivariate normal ($\Sigma = I_d$). We repeat the same analysis and the results, being compatible with true values of V_d, do not outperform the ones obtained using runif (see details in day5ex2part3.R in the appendix).

The code is prepared to sample numbers from a multivariate normal with Σ=diag(c,d,d). Further investigation on the optimal value of c could be interesting, we think c<1 should work better.

5 Monte Carlo Optimization methods

Stochastic simulations can be also used to find maximizers of multidimensional scalar functions. Let be h:$D \subset \mathbb{R}^n \longrightarrow \mathbb{R}$, we say that x^* is a global maximizer of h (in D) if $\forall\, x \in D$, $h(x^*) > h(x)$. In this section we revise three different techniques in two different problems.

5.1 Exercise 3

We consider here a function h:$\mathbb{R} \longrightarrow \mathbb{R}$ linear combination of three different Gaussian functions:

$$h(x) = 3\ G(x,-2,0.2) + 6\ G(x,1,0.5) + G(x,3,0.3)$$
$$G(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.1}$$

The function is positive and decay fast to zero when $|x|$ grows. Out of the interval [-5,5] we can consider h(x)=0. x=-2 is the global maximizer of h that present other 2 local maximizers at x=1 and x=3.

We can sample n numbers uniformly on [-5,5] (i.e. X=$\{x_i\}$) and simply check which one produce a bigger value h(x_i). Using only n=100 we find x^* =1.991458 which is quite close to the true solution. Left panel of figure 27 shows the function h(x) and the scatter plot of this uniform sample (included the maximizer found in red).

However, when sampling is made uniformly in [-5,5] many points lie away from the peaks, and we waste computational time constructing them and evaluating h. A good idea is sampling using the proper function h as pdf. If h>0 and $\int_D h(x)dx < \infty$ there exist a constant c s.t. h(x)/c is a pdf (c=10 in this exercise).

In this second part we sample numbers from a non-uniform distribution given by the normalised h function. We use for the sampling MCMC technique through a Metropolis algorithm with U[-5,5] as proposal distribution. Notice, that this sampling:

1) requires to generate extra numbers to reach the target distribution (we use n=10^4 to extract 100 numbers)

2) produces non-independent numbers, could be that starting with a positive x value and using only n=100 numbers, more points are sampled under the lower peaks than under the one at x=-2, which is not helping in our detection.

Right panel of figure 27 shows the sampled points in one realization of n=100 numbers, the maximizer is reported at x^* =1.975628 close to the true one at x=-2.

A common problematic in optimization algorithms is getting stack in a local minimizer (maximizer) of the Objective. In this exercise, optimize command of R produces the solution $x^* = 1.000001$ which is not the global maximizer.

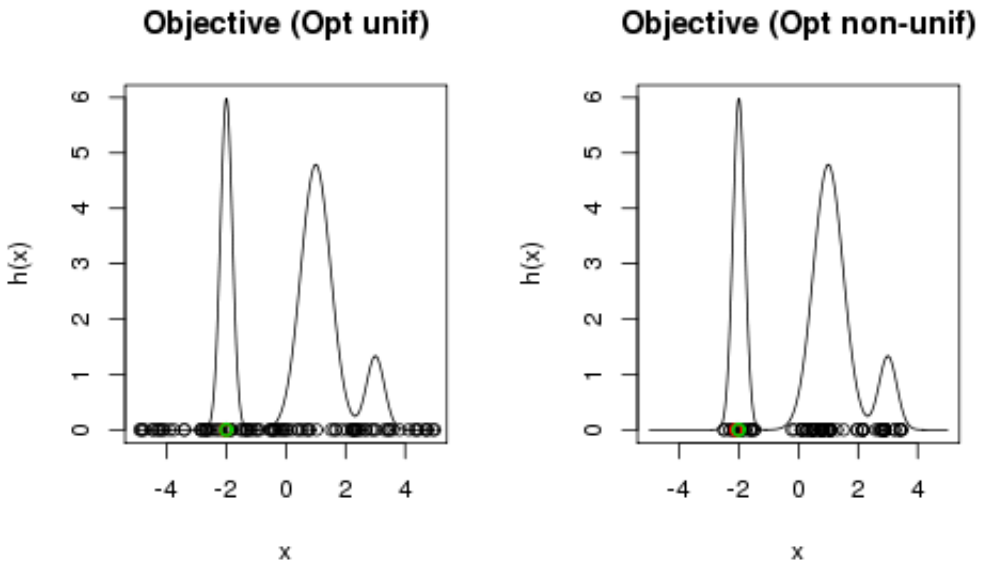

Figure 27: Objective function h(x) with sampled points following U[-5,5] (left panel) and 0.1*h(x) (right panel). Maximizer detected is coulored in red and the true one in green.

5.2 Exercise 4

In this second exercise the objective is a 2D scalar function (i.e. x=(x_1,x_2) $\in \mathbb{R}^2$), a combination of 1D Gaussians defined by:

$$\begin{aligned} h(x) =& 3\ G(x_1,0,0.5)G(x_2,0,0.5) + G(x_1,-1,0.5)G(x_2,1,0.3) + ... \\ &...+G(x_1,1,0.5)G(x_2,1,0.3) \end{aligned} \tag{5.2}$$

The function is positive and decay fast to zero when $\|x\|_2$ grows. x=(0,0) is the global maximizer of h that present other 2 local maximizers at x=(-1,1) and x=(1,1). Figure 28 shows the contour plot of the 2D function.

Annealing technique is a sequential method that mixes features of the stochastic gradient method and the tempered density simulation. Read the indications in lecture notes we implement the following pseudo-code:

1) We consider the deterministic sequence $T_i = 1/\text{Ln(i)}$ for $i \geq 1$ (value i=1 is not a problem)

2) We choose 2 different auxiliar distributions a) uniform in $[-3,3]^2$ and b) a bivariate normal with zero mean and $\Sigma = I_2$

3) Initialize X[1,]=c(runif(1,-3,3),runif(1,-3,3))

4) Main bucle i=2,...,n=10^4

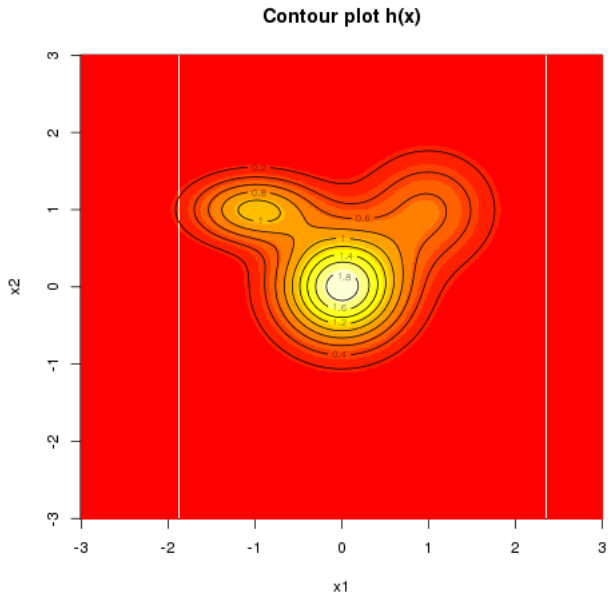

Figure 28: Contour plot of the objective function h(x).

a) Sample ζ from g

b) Propose a new candidate value Y=X[i,]+ ζ

c) If Y $\notin$ [-3,3]^2 we change Y to the closer point inside $[-3,3]^2$

d) We accept to move from X[i,] to X[i+1,]=Y with probability ρ=min(exp($\Delta h/T_i$),1), i.e. accept=binom(1,1,ρ) ; X[i+1,1] <- ifelse(accept==1,Y[1],X[i,1]) ; X[i+1,2] <- ifelse(accept==1,Y[2],X[i,2]).

Figure 29 shows the sequences for the two different auxiliar g's in treatment. In both cases the last iteration is quite close to the true maximizer. However, we notice here:

1) One good thing of the algorithm is that choosing g covering a big area, we don't get stack in local maximizers

2) In both cases we find a high number of rejections

3) The outcome sequence $\{x_i\}$ is not progressively reducing the residual $|h(x_i)$-h((0,0))|, because $\{h(x_i)\}$ is not monotonously increasing with i. Thus, we can find points in the sequence which are closer to the true maximizer than the last element.

We could decide to accept a candidate Y if we make a progress in enlarging the objective (i.e. when $\Delta h>0$ instead of c and d steps in the loop), with this change a better estimation of the maximizer is found, figure 30 shows the new sequences for each auxiliar g in use.

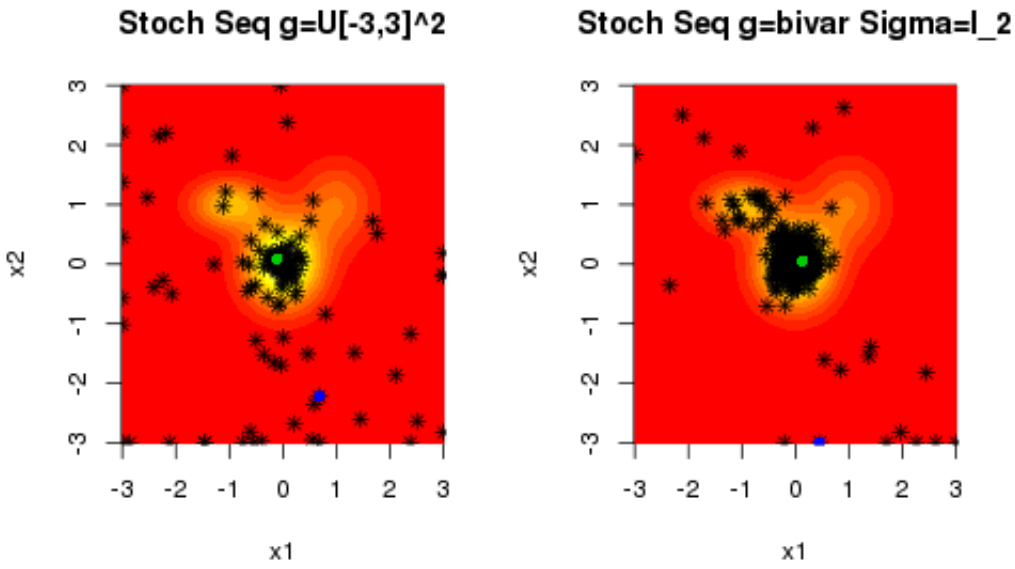

Figure 29: Stochastic sequences, using $U[-3,3]^2$ as auxiliar g (left panel) and Bi-variate mean=(0,0) $\Sigma = I_2$ (right panel). Initial and final points are coloured in blue and green respectively.

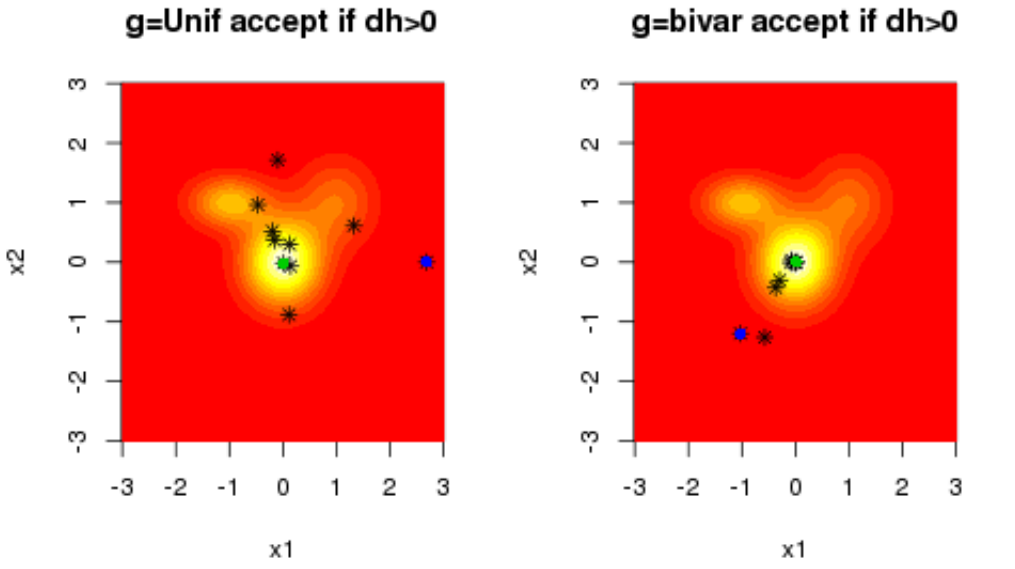

Figure 30: Same than figure 29 but accepting the candidate iif $\Delta h > 0$.

Appendix

A Day 1

```
#Day1Exercise1
#1) Implement the congruence method
# use: a=16807, b = 0 and M = 2147483647


## Linear congruential method
a = 16807 ; b =0 ; M =2147483647
n = 1000 # nb of iterations
x = numeric(length=n)
x[1] = floor(runif(1,0,(M-1)))
#x[1]=3
for(i in 2:n)
{
  x[i] = (a*x[i-1] + b) %% M
}
x = x/M

#is x uniformly distributed in [0,1]?
h=seq(0,1,0.01)
txt='Histogram of X'
png("d1e1_fig1.png")
plot(h,dunif(h,0,1),type='l',col=2,main=txt,xlab='x',ylab='
    Density')
hist(x,breaks=seq(0,1,0.125),prob=T,add=T)
dev.off()
#looks good

#2) Now some correlation analysis
#a) The autocorrelation function and scatter (xt,xt+2)
Y=matrix(nr=(n-2),nc=2,data=0)
for(i in 1:(n-2)){
 Y[i,1]=x[i]
 Y[i,2]=x[i+2]
} #Y components contains the points we want to plot
txt2='scatter plot of (x[i],x[i+2])'
png("d1e1_fig2.png",height=300)
par(mfrow=c(1,2))
acf(x)
plot(Y[,1],Y[,2],xlab='x[i]',ylab='x[i+2]',main=txt2)
dev.off()
#acf plot shows that for any lag the corr is under the threshold
    .
#The scatter plot x[i] vs x[i+2] look like uncorrelated
    variables

#2) KS test. H0-> x is uniformily distributed in [0,1]
```

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

#We are getting that D = 0.0183, p-value = 0.8903, p is quite
#big so we cannot reject H0 (x looks like follows U[0,1])

#lets try to plot the theoretical cdf i.e.(y=x for x in [0,1])
#the experimental one is
Fexp=ecdf(x)
xx=sort(x)

txt3='Theo & Exp cumulatives'
png("d1e1_fig3.png")
plot(xx,punif(xx,0,1),type='l',lwd=2,col=2,xlab='x',ylab='F(x)',
   main=txt3)
lines(ecdf(x),lwd=1.5)
dev.off()
# bin=100
# cdf.x=rep(0,bin)

# for(i in 0:bin-1)
#   lim=i/bin
#    tot=tot+1
#   cdf.x[i+1]=
```

B Day 2

```
# Day 2  Exercises 1 to 4

# 1) Simulate a sample from a Cauchy(0,1) distrib with inversion
#method
n=100000 # n big enough
u=runif(n,0,1)
h=seq(-5,5,.1)
f=1/(pi*(1+h^2))

#Plot the theoretical pdf of Cauchy(0,1)
txt="A sample from Cauchy(0,1) distribution"
png("d2e1_fig1.png")
plot(h, f, type="l",col=2, main=txt)

#now we use inverse of cauchy cdf to create a sample of a rv
# wich follows a Cauchy(0,1)pdf F(x)=y -> y=F^-1(x)
y=tan(pi*(u-0.5))

#Plot the histogram
lb = min(y)-0.2
ub = max(y)+0.2
hist(y, breaks = seq(lb,ub,0.6),add = T,prob=T)
```

```
dev.off()
#####################################################
#####################################################
#2) Simulate a sample from Beta(2,5) ditrib using rejection
# method with a uniform distribution as auxiliar distribution.

n = 10000
h = seq(0, 1, 0.01) #beta(2,5) pdf decays fast
K = max(dbeta(h,2,5)+0.1) #maximum of Beta pdf
#now we generate n random points in the box [0,1]x[0,K]
u1 = runif(n)
u2 = runif(n,0,K)
#and select those ones below the target pdf
i = u2<dbeta(u1,2,5)
x = u1[i]
#We plot the points in the left
png("d2e1_fig2.png",height=300)
par(mfrow=c(1,2))
plot(u1,u2,xlab='x',ylab='y',main='accept if y<dbeta(x,2,5)')
lines(u1[i],u2[i],type='p',col=2)

# x should be distributed following dbeta(2,5)
txt = "A sample from Beta(2,5)"
plot(h, dbeta(h,2,5), col=2,type="l", main=txt,xlab='x',ylab='f(
    x)')
hist(x,seq(-0.1,1.1,0.05),prob=T,add=T) #prob=T normalizes the
    hist (area=1)
                                        #add=T is a hold on
dev.off()
#############################################################
#############################################################
#3) Simulate a sample from an N(0,1), distribution with the
    rejection method
# using the doble exponential distribution as auxiliar variable

h = seq(-5, 5, 0.01)
#how is this doble exp pdf?

png("d2e1_fig3.png")
plot(h, seq(0,1.0,length=length(h)),type='n',xlab='x',ylab='y',
     main="f, g and K*g")

lines(h,0.5*exp(-abs(h)),type='l',col=2) # auxiliar g case alpha
    =1
                          # symmetric, normalised
lines(h,dnorm(h,0,1),type='l',col=3,lwd=2) # f, target pdf
#we define K s.t. Kg(x)>f(x) forall x

K=max(abs(dnorm(h,0,1)/(0.5*exp(-abs(h)))))+0.5
```

```
#we rescale with this one
lines(h,K*0.5*exp(-abs(h)),col=2,lwd=2)
dev.off()

#Now we use rejection method
#We generate uniform numbers in the area under Kg(x)

n=10000
#1)First the X component following g(x)
u1 = c(-rexp(n/2,rate=1), rexp(n/2,rate=1))
#hist(u1,breaks=seq(min(u1)-0.1,max(u1)+0.1,0.2),add=T,prob=T)
#2) Y component is generated uniform in 0, Kg(x)
u2 = runif(n,0,K*0.5*exp(-abs(u1)))

#the pairs (u1,u2) are in the box [min(u1),max(u1)]x[0,K*g(u1)]
# we neglect those pairs above N(0,1)
i = u2 < dnorm(u1,0,1)
x = u1[i]
#Plot all sampled points
png("d2e1_fig4.png",height=300)
par(mfrow=c(1,2))

plot(h,K*0.5*exp(-abs(h)),col=2,lwd=2,type='l',xlab='x',ylab='y'
    ,
     main='Points, Rejection Method')
lines(h,dnorm(h,0,1),type='l',col=3,lwd=2)
points(u1,u2,pch=19,cex=0.2,col=2)
#and the points that are inside the target distribution
points(x,u2[i],pch=19,cex=0.21,col=3)

# x must be normal distributed
txt = "Sampling N(0,1)"
plot(h, dnorm(h,0,1), type="l",xlab='x',ylab='Density', lwd=2,
     col=3, main=txt)
hist(x,seq(-5.1,5.1,0.5),prob=T,add=T)
dev.off()



##############################################
##############################################
#4) basic experiment. Everytime we throw a needle it falls in
# a planck at a given position (x,y), we can assume the plancks
   are
# infinity long and we just care about the intersection of
   vertical lines.

# we take the x randomly in [0,l]
n=10000
l=1
```

```
x=runif(n,0,l) #this is the sharp of the needel
#analogously the angle formed with with x-axis can be sampled
#uniformly in [0,2pi]
theta=runif(n,0,2*pi)
#each pair (x,theta) characterize the needle. if the tail
#   x_tail = x + l*cos(theta)
#is out of the bouds [0,l] means that we have an intersection
# we count the needles that intersect the right line
right=sum(x+l*cos(theta)>l)
# and left line
left=sum(x+l*cos(theta)<0)
#the probability of intersection will be the favoral cases over
   the total
p_inter = (right+left)/n
```

```
# Day2 Exercises 5 to 7
#
# 5) sampling the waitining time for an offer>12 for my car
# we use Poisson(10) to simulate an offer -> time is a
# discrete variable
n=1000 # experiments
time=rep(1,n) #init of times
for (i in 1:n){
  while(rpois(1,10)<12) {
    time[i]=time[i]+1
  }
}
# Now we take the average time
time_ave=sum(time)/n

#part2

n=1000 # experiments
time2=rep(1,n) #init of times
for (i in 1:n){
  while(rpois(1,10)<rpois(1,10)) {
    time2[i]=time2[i]+1
  }
}
# Now we take the average time
time2_ave=sum(time2)/n

#Figure
png("d2e2_fig1.png",height=300)
par(mfrow=c(1,2))
txt1='Conditianally on x1=12'
plot(c(0,15),c(0,0.4),type='n',main=txt1,xlab='time',ylab='
   Density')
lines(rep(time_ave,21),seq(0,0.25,0.25/20),type='l',col=2,lwd=3)
```

```
hist(time,seq(0,max(time),1),prob=T,add=T)

txt2='Unconditionally'
plot(c(0,10),c(0,0.6),type='n',main=txt2,xlab='time',ylab='
   Density')
lines(rep(time2_ave,21),seq(0,0.4,0.4/20),type='l',col=2,lwd=3,
     main=txt2,xlab='time',ylab='Density')
 hist(time2,seq(0,max(time2),1),prob=T,add=T)

dev.off()


#####################################################
#####################################################
# 6)Variance in the estimation of the median of a Poisson
# distribution
# a) simulate dataset of n=200 observ of Poisson(10)
n=200
dataset=rpois(n,10)
#estimation of the mean taking this sample
mu=mean(dataset)

#b)now we analyse
B=10000
X.theo <- X.boot <- matrix(nrow=n,ncol=B) # mat init
# we construct B independent samples using the theoret distrib
# and using sample comand
for(b in 1:B) {
  X.theo[,b] <- rpois(n,10)
  X.boot[,b] <- sample(x=dataset,size=n,replace=T)
}
# we compute the median for every column of these matrices, and
# we have a theo and a boot vector of means
mean.theo <- apply(X.theo,MARGIN=2,FUN=mean)
mean.boot <- apply(X.boot,MARGIN=2,FUN=mean)

mean(mean.theo)
var(mean.theo)
mean(mean.boot)
var(mean.boot)

# they are different. Estimating V(mean) with the so-called
# bootstrap estimator
Var_boot = (1/B)*sum((mean.boot - (1/B)* sum(mean.boot))^2)
#this number is very close to var(mean.boot)

#####################################################################

#####################################################################
```

```
# 7)Simulate a sample of size 50000 from a bivariate centred,
# standarized Gaussian vector with corr_coeff rho=0.7.

#we first create two independent normal distributed variables
n=50000
x1=rnorm(n,0,1)
x2=rnorm(n,0,1)
x=matrix(nrow=n,ncol=2) # x is nx2
x[,1]=x1;x[,2]=x2
xt=t(x) #xt is 2xn
# x must be uncorrelated and Var(x)=I_2
# we want y a bivariate sample with covariance matrix SIGMA
SIGMA=matrix(nrow=2,ncol=2)
SIGMA[1,1] <- SIGMA[2,2] <- 1
SIGMA[1,2] <- SIGMA[2,1] <- 0.7
# Cholesky factorization of SIGMA and y construction
U = chol(SIGMA) ; L = t(U)
yt=L %*% xt #y is 2xn
y=t(yt) # y is nx2

#now we plot the points for x and y
png("d2e2_fig2.png")
par(mfrow=c(1,2))
plot(x[,1],x[,2],main='original x',xlab='x1',ylab='x2',pty = "s
    ")
plot(y[,1],y[,2],main='y=Lx',xlab='y1',ylab='y2',pty = "s")
#this looks good
dev.off()
var(y) # and the numbers are close to the target SIGMA

#checking the empirical bivariate density, Contour plot
#require(MASS)
#image(kde2d(y[,1],y[,2]),asp=T);points(y,cex=1,col=3)
#contour(kde2d(y[,1],y[,2]),add=T)

# Now we extract a sample of a random variable approximately
# distributed as y2|y1=0.5.
#so we take y[i,j] matrix and extract y[,2] when y[,1]=0.5
i1=y[,1]>0.49
i2=y[,1]<0.51
i=i1*i2 # logical with position of the entries we want
tot=sum(i)
# we extract y[2,i] into z
z=vector(mode="numeric",length=tot) # we initialize
cont=1
for (j in 1:n){
  if(i[j]==1) {z[cont]=y[j,2];cont=cont+1}
}
# Lets plot this sample
hist(z)
```

```
mean(z)
var(z)
#and compare with the pdf of a bivariate with SIGMA and y1=0.5
SIGMA.INV=solve(SIGMA)
A=SIGMA.INV[1,1];B <- C <- SIGMA.INV[1,2];D=SIGMA.INV[2,2]

cte1=1/(2*pi)*(1/sqrt(det(SIGMA)))
cte2=exp(-0.5*(0.5^2)*A)
h=seq(-4,4,0.01)
Biv.y1.0p5=cte1*cte2*exp(-0.5*(0.5*h*(C+B) +h*h*D))
#In order to make the comparison we need to normailze this curve
Area=sum(0.01*Biv.y1.0p5) #
Theo=Biv.y1.0p5/Area

png("d2e2_fig3.png",height=300)
par(mfrow=c(1,2))
plot(h,Theo,type='l',col=2,lwd=2,pty = "s",
     main='f(y1|y2=0.5) normalised',xlab='z',ylab='Density')
hist(z,add=T,prob=T) # This looks pretty good :)

plot(h,dnorm(h,mean(z),sqrt(var(z))),type='l',col=3,lwd=2,pty =
    "s",
     main='N(mean(z),var(z))',xlab='z',ylab='Density')
hist(z,add=T,prob=T)

dev.off()
```

C Day 4

```
# Day4 Exercise1
# 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=1000 #number of iterations
x=rep(1,n) #allocate the vector
x[1]=rbinom(n=1,size=1,prob=0.5) #initial guess 0 or 1 50%
#we start the iterative process
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("d4e1.png")
plot(x,xlab='iter',ylab='x',main='Markov Chain') #We can see how
    x
                        # change from 0 to 1 and there is not
                           any
                        # convergence to any value as expected
```

```
dev.off()
```

```
# Day 4 exercise 2
# 2) Simulate a Markov Chain with the metropolis algorithm with
# the Cauchy distribution C(0,1) as a target distribution.
# Use e.g. a normal pdf with 0 mean and a unit variance as a
   proposal q

n=10000 # number of iterations we want
x=rep(1,n) # allocating x
x[1]=rnorm(1,0,1) #initial iteration
#Metropolis algorithm
for (i in 1:n-1){
  #propose a candidate
  Y <- rnorm(1,0,1)
  #compute the ratio
  R <- (dcauchy(Y,0,1)*dnorm(x[i],0,1))/
    (dcauchy(x[i],0,1)*dnorm(Y,0,1))
  #compute acceptance prob
  p <- min(1,R)
  #We accept or not the candidate randomly
  accept <- rbinom(1,1,p)
  x[i+1] <- ifelse(accept==1,Y,x[i])
}
plot(x) # This is not very clear

# In theory, with the iterations x[i] should follow C(0,1),
# we can check this with
h=seq(-4,4,0.01)
last=1000

png("d4e2_f1.png",height=300)
par(mfrow=c(1,2))
#left panel
plot(x[(n-last):n],xlab='iter',ylab='x',main='Chain elements')
#right panel
plot(h,c(0,rep(0.5,length(h)-1)),type='n',xlab='x',ylab='Density
   ',
     main='Histogram')
lines(h,dcauchy(h,0,1),type='l',col=2)
# lets look at the last iterations
lb=min(x[(n-last):n])-0.1
ub=max(x[(n-last):n])+0.1
hist(x[(n-last):n],breaks=15,add=T,prob=T)
dev.off()
# The target pdf is not verywell obtained, we can see those high
# peaks in the borders. In the slides we were warning about this
    kind
# of problematic. Perhaps N(0,1) is not the most appropriate
   auxiliar
```

```
# distribution if we target Cauchy(0,1)

png("d4e2_f2.png")
acf(x[(n-last:n)],type="correlation",plot=T)
dev.off()
```

```
# Day4 exercise 3
# 3)Simulate a Markov Chain with the metropolis-hastings
   algorithm
# with the Cauchy distribution C(0,1) as a target distribution.
# Use e.g. Gaussian proposal. How does the simulated markov
   chain
# depend on the variance of the proposal distribution?

n=10000 # number of iterations we want
x=rep(1,n) # allocating x
x[1]=runif(1,0,1) #initial iteration chosen in [0,1]
#Metropolis-hanstings algorithm
sigma <-1 # we can change sigma easily here
for(i in 1:n) {
  #propose a candidate
  Y <- rnorm(1,x[i],sigma)
  #compute the ratio
  R <- (dcauchy(Y,0,1)*dnorm(x[i],Y,sigma))/
    (dcauchy(x[i],0,1)*dnorm(Y,x[i],sigma))
  #compute acceptance prob
  p <- min(1,R)
  #We accept or not the candidate randomly
  accept <- rbinom(1,1,p)
  x[i+1] <- ifelse(accept==1,Y,x[i])
}
# Lets plot the last 10^3 elements
last=1000
png("d4e3_s1f1.png")
plot(x[(n-last):n],main='',ylab='x')
dev.off()

# In theory, with the iterations x[i] should follow C(0,1),
# we can check this with
h=seq(-10,10,0.01)
png("d4e3_s1f2.png",height=300)
par(mfrow=c(1,2))
#layout(M)
#par(mar=c(6,4,4,2))
txt='Last 10   iter, sigma=1'
plot(h,c(0,rep(0.4,length(h)-1)),type='n',
     main=txt,xlab='x',ylab='Density')
lines(h,dcauchy(h,0,1),type='l',col=2)
# lets look at the last 1000 iterations
hist(x[(n-last):n],breaks=15,add=T,prob=T)
```

```
#and plot the autocorrelation for exercise 5
acf(x[(n-last):n],type="correlation",plot=T)
dev.off()


# we analise changing sigma, trying same thing

n=10000 # number of iterations we want
x=rep(1,n) # allocating x
x[1]=runif(1,0,1) #initial iteration chosen in [0,1]
#Metropolis-hanstings algorithm
sigma <-3 # we can change sigma easily here
for(i in 1:n) {
  #propose a candidate
  Y <- rnorm(1,x[i],sigma)
  #compute the ratio
  R <- (dcauchy(Y,0,1)*dnorm(x[i],Y,sigma))/
    (dcauchy(x[i],0,1)*dnorm(Y,x[i],sigma))
  #compute acceptance prob
  p <- min(1,R)
  #We accept or not the candidate randomly
  accept <- rbinom(1,1,p)
  x[i+1] <- ifelse(accept==1,Y,x[i])
}
last=1000
png("d4e3_s3f1.png")
plot(x[(n-last):n],main='sigma=3',ylab='x')
dev.off()

h=seq(-10,10,0.01)
png("d4e3_s3f2.png",height=300)
par(mfrow=c(1,2))
txt='hist for sigma=3'
plot(h,c(0,rep(0.4,length(h)-1)),type='n',
     main=txt,xlab='x',ylab='Density')
lines(h,dcauchy(h,0,1),type='l',col=2)
# lets look at the last 1000 iterations
hist(x[(n-last):n],breaks=20,add=T,prob=T)

#and plot the autocorrelation for exercise 5
acf(x[(n-last):n],type="correlation",plot=T)
dev.off()

# The autocorrelation function is better,

# Lets check using sigma<1
n=10000 # number of iterations we want
x=rep(1,n) # allocating x
x[1]=runif(1,0,1) #initial iteration chosen in [0,1]
#Metropolis-hanstings algorithm
```

```
sigma <-0.3 # we can change sigma easily here
for(i in 1:n) {
  #propose a candidate
  Y <- rnorm(1,x[i],sigma)
  #compute the ratio
  R <- (dcauchy(Y,0,1)*dnorm(x[i],Y,sigma))/
    (dcauchy(x[i],0,1)*dnorm(Y,x[i],sigma))
  #compute acceptance prob
  p <- min(1,R)
  #We accept or not the candidate randomly
  accept <- rbinom(1,1,p)
  x[i+1] <- ifelse(accept==1,Y,x[i])
}
last=1000
png("d4e3_s0p3f1.png")
plot(x[(n-last):n],main='sigma=0.3',ylab='x')
dev.off()

h=seq(-10,10,0.01)
png("d4e3_s0p3f2.png",height=300)
par(mfrow=c(1,2))
plot(h,c(0,rep(0.4,length(h)-1)),type='n',
     main=txt,xlab='x',ylab='Density')
lines(h,dcauchy(h,0,1),type='l',col=2)
# lets look at the last 1000 iterations
hist(x[(n-last:n)],breaks=15,add=T,prob=T)
#and plot the autocorrelation for exercise 5
acf(x[(n-last:n)],type="correlation",plot=T)
dev.off()

# The conclusion is that the autocorrelation decays faster
# when the sigma is bigger. further investigation?
```

```
# Day4 exercise4
# 4) Simulate a sample from a beta(1/2,1/2) with the rejection
# algorithm (using the uniform distrib as majority density).
# Then simulate the same sample with the MH algorithm

#a) The first part is done Day2ex2, we just change the
   parameters
#of the beta. We must be careful with the peaks
#at 0 and 1 of beta(0.5,0.5)

n = 10000
h = seq(0.01, 0.99, 0.01) #beta(1/2,1/2) pdf decays fast
K = max(dbeta(h,0.5,0.5)+0.1) #maximum of Beta pdf
#now we generate n random points in the box [0,1]x[0,K]
u1 = runif(n)
u2 = runif(n)*K
#and select those ones below the target pdf
```

```
i = u2<dbeta(u1,0.5,0.5)
x = u1[i]
# x should be distributed following dbeta(2,5)
txt = "Beta(1/2,1/2) using rejection"
png("d4e4_f1.png",height=300)
par(mfrow=c(1,2)) #
plot(u1,u2,xlab='x',ylab='y',main='accept if y<dbeta(x,1/2,1/2)'
    )
lines(u1[i],u2[i],type='p',col=3)

plot(h,c(0,rep(max(dbeta(h,0.5,0.5)+0.1),length(h)-1)),
     main=txt,type='n',ylab='Density',)
lines(h, dbeta(h,0.5,0.5), type="l",lwd=2,col=2,main=txt)
hist(x,seq(-0.1,1.1,0.05),prob=T,add=T)
dev.off()

#more or less we have x distributed as beta(0.5,0.5)

#b) Now using Metropolis-Hastings, we will use as proposal N(x[i
    ],1)

n=100000 # number of iterations we want
xx=rep(1,n) # allocating x
xx[1]=runif(1,0,1) #initial iteration chosen in [0,1]
#Metropolis-hanstings algorithm
sigma <-1
for(i in 1:n) {
  #propose a candidate
  Y <- rnorm(1,xx[i],sigma)
  #compute the ratio
  R <- (dbeta(Y,0.5,0.5)*dnorm(xx[i],Y,sigma))/
    (dbeta(xx[i],0.5,0.5)*dnorm(Y,xx[i],sigma))
  #compute acceptance prob
  p <- min(1,R)
  #We accept or not the candidate randomly
  accept <- rbinom(1,1,p)
  xx[i+1] <- ifelse(accept==1,Y,xx[i])
}

#Lets take a look of the last iterations
h=seq(0.01,0.99,0.01)
last=1000
png("d4e4_f2.png",height=300)
par(mfrow=c(1,2))
plot(xx[(n-last):n],main='Last entries',ylab='x')

txt = "Beta(1/2,1/2) using MH"
plot(h,c(0,rep(max(dbeta(h,0.5,0.5)+0.1),length(h)-1)),
     main=txt,type='n',ylab='Density',)
lines(h,dbeta(h,0.5,0.5),type='l',col=2,lwd=2)
```

```
# lets look at the last 10000 iterations
hist(xx[(n-last):n],breaks=seq(0,1,0.05),add=T,prob=T)
dev.off()

#and plot the autocorrelation for exercise 5
png("d4e4_f3.png",height=300)
par(mfrow=c(1,2))
acf(x,type="correlation",plot=T)
acf(xx[(n-last):n],type="correlation",plot=T)
dev.off()

#Besides of the computational costs, correlogram is the main
# difference between the sequences generated by both methods.
```

```
# Day4 exercise6
# 6) Lets consider (X,Y) follow a Multivariate Normal with 0
    mean
# and covariance matrix [1 rho;rho 1]. We must simulate a sample
# of size 1000 using the gibbs sampler. Check that the sampled
    has
# the target distrib after a suitable burn-in period
# Hint: Y|X = x distrib as N(rho*x,1-rho^2)

#first we define our multivariate Gaussian
f <- function(x,y,rho){
  X=matrix(nr=2,nc=1,data=c(x,y))
  sigma=matrix(nr=2,nc=2,data=c(1,rho,rho,1))
  sm1=solve(sigma)
  cte1=1/(2*pi)*(1/sqrt(det(sigma)))
  xtsm1x=t(X)%*%sm1%*%X
  Z=cte1*exp(-0.5*xtsm1x)
  return(Z)
}

#Parameter selection
rho=0.7 #we choose rho=0.7
SIGMA=matrix(nr=2,nc=2,data=c(1,rho,rho,1))


#we should work with 2 chains
niterations=10000; nchains=2
# first column of XX contais x and the 2nd contains y
XX <- matrix(nr=niterations,nc=nchains,data=NA)
XX[1,] <- runif(n=nchains) # two numbers in [0,1]
                           # we need only x
#main loop
for(i in 1:(niterations-1)){
  XX[i,2] <- rnorm(n=1,rho*XX[i,1],sqrt(1-rho^2))
  XX[i+1,1] <- rnorm(n=1,rho*XX[i,2],sqrt(1-rho^2))
}
```

```
#in the last update we left XX[i+1,2], we calculate it now
XX[i+1,2]=rnorm(n=1,rho*XX[i+1,1],sqrt(1-rho^2))

#we assume the burn-in period is small in this Gibbs sampling
# and reject 20% initial points
last=1000;last2=8000
n=niterations
txt='size(z)=1000'
png("d4e6.png",height=300)
par(mfrow=c(1,2))
plot(XX[(n-last):n,1],XX[(n-last):n,2],pch=19,cex=0.2,
     main=txt,xlab='z1',ylab='z2')
txt2='size(z)=8000'
plot(XX[(n-last2):n,1],XX[(n-last2):n,2],pch=19,cex=0.2,
     main=txt2,xlab='z1',ylab='z2')
#could be ok, we can check the variance matrix
dev.off()
z=XX[(n-last):n,]
var(z)
```

D Day 5

```
#Day5Exercise1 Monte Carlo Integration
#1) Consider the function h(x) = [cos(50x)+sin(20x)]^2
f <- function(x) {(cos(50*x)+sin(20*x))^2}
#plot
h=seq(0,1,0.001)
png("d5e1_f1.png")
plot(h,f(h),type='l',col=1,xlab="x",ylab="h(x)")
dev.off()

#2) Write an algorithm to compute the integral of h(x) on [0,1]
#with stochastic simulationns. Sampling from a uniform distrib
n=100000
X=runif(n,0,1)
suma=0
for(i in 1:n) {suma=suma+f(X[i])}
int=suma/n

integrate(f,0,1,rel.tol=1.e-06,abs.tol=1.e-06)
int_true=0.9652009
# the real integral is
# 0.9652009 with absolute error < 1.9e-10
#Looks good when we increase n

n=c(1000,10000,100000,1000000,1.e7)  #some settings
logn=log10(n)
int=rep(0,length(n))
err=int
```

```
for(j in 1:length(n)) {

  X=runif(n[j],0,1)
  suma=0
  for(i in 1:n[j]) {suma=suma+f(X[i])}
  int[j]=suma/n[j]
  err[j]=abs(int[j]-int_true)
}

# residuals figure
txt='error MC integration'
png("d5e1_f2.png")
plot(logn,c(max(log(1/sqrt(n)))+1,rep(min(log(err)),length(n)-1)
   ),
     type='n',main=txt,ylab='log(err)',xlab='log(n)')
lines(logn,log(err),type='l',col=2,lwd=2)
lines(logn,log(1/sqrt(n)),type='l',col=1,lwd=1.5)
#lines(logn,log(1/n),type='l',col=4,lwd=1.5)
# legend
txt2=c("MC method", "O(1/sqrt(n))")
legend("topright",txt2, col=c(2,1),lty=c(1,1),cex=0.7)
dev.off()
```

```
#Day5Exercise2
#Bd is the set (x1,...,xd) in R^d s.t. sum(xi^2) < 1
#Cd is the set (x1,...,xd) in R^d s.t. sum(abs(xi)) < 1
#The goal is compute the volume of the unit ball
#by stochastic integration

#Lets define the functions
norm2 <- function(x){sum(x^2)}
#norm1 <- function(x){sum(abs(x))}

#Vd = |Cd| int(IBd f(x) dx)
# we need the area of the Cd, but thats 2^d
n=10000
d=2
area.Cd=2^d
X=matrix(nr=n,nc=d,data=1) # allocate memory

X[,1]=runif(n,-1,1)
X[,2]=runif(n,-1,1) #sampling points in C2

#We define the indicator function
I.Bd <-function(x){ifelse(norm2(x)<1,return(1),return(0))}

# Vd = Area.Cd *int(I.Bd)
#Lets make a plot
suma=0
```

```
index=rep(1,n)
for(i in 1:n){
  kaka=I.Bd(X[i,])
  suma=suma+kaka
  if(kaka==0){index[i]=0}
}
Y=matrix(nr=suma,nc=d,data=0)

#points inside B2
cont=0
for(i in 1:n){
  if(index[i]==1){
    cont=cont+1
    Y[cont,]=X[i,]
  }
}
txt='Sampling on C2 and keeping points in B2'
png("d5e2.png")
plot(X[,1],X[,2],main=txt,xlab='x1',yla='x2')
points(Y[,1],Y[,2],col=2)
dev.off()

Vd=area.Cd*suma/n

Vd_true=pi^(d/2)/gamma(1+d/2)

#The result is quite close to the true one :)


#Lets repeat the experiment now we include the variance
   estimator
#from the notes and work on different realizations (j=1,....,B )
# using 10^5 sims in each one.

n=10000
d=3 #change d here to reproduce the table (stoch numbers)
B=100
#some values out of the bucle
area.Cd=2^d
Vd_true=pi^(d/2)/gamma(1+d/2)
###############
Vd_vec=rep(0,B)
for(ii in 1:B){

X=matrix(nr=n,nc=d,data=1) # allocate memory

for(j in 1:d){X[,j]=runif(n,-1,1)} #sampling in Cd
suma=0
for(i in 1:n){
  kaka=I.Bd(X[i,])
```

```
  suma=suma+kaka
}

Vd=area.Cd*suma/n
Vd_vec[ii]=Vd
#b) With respect to the Variance of the estimator, we must say
# that our Xi's are iid variables and therefore
# V[hat_I] = 1/n int(I.Bd(x)-I)^2 f(x) dx
# Lets calculate this but only ones
if(ii==1){
suma2=0
for(i in 1:n){
  kaka=I.Bd(X[i,])*area.Cd
  suma2=suma2+(kaka-Vd)^2
}

Var_estim1sim=suma2/(n^2)
}

}

# numbers For the report
barVd=mean(Vd_vec)
var_Vd_teo=var(Vd_vec)


# Table with the results for part 1
#    d  Vd_true   Vd_stoch_int (Ave on B)  estim of var_Vd 1
   simul
#    2  3.14159    3.139 pm 0.0159            0.0002617024
#    3  4.18879    4.1946 pm  0.0431          0.0015953
#   10  2.55016    2.562048 pm 0.5398877      0.2823511
#Notice that the hipervolume is not strictly increasing with d
   ...


# Table with the results for part 2
#    d        Var_Vd (averaging on 100)
#    2           0.000252653
#    3           0.001860939
#   10           0.2914787
```

```
# Day 5 Exercise 2 part 3 sampling from d-variate normal
#something from previous part we will need
norm2 <- function(x){sum(x^2)}
# Indicator function
I.Bd <-function(x){
  ifelse(norm2(x)<1,return(1),return(0))
}
#d-variate Normal with \Sigma=I_d
```

```
f <- function(x,c){
  d=length(x)
  X=matrix(nr=d,nc=1,data=x)
  sigma <- sm1 <- diag(c,d,d)
  cte1=1/((2*pi)^(d/2))*(1/sqrt(det(sigma)))
  xtsm1x=t(X)%*%sm1%*%X
  Z=cte1*exp(-0.5*xtsm1x)
  return(Z)
}

#Lets repeat the experiment sampling with d-variate normal Sigma
   =Id
# we work on different realizations (j=1,....,B ) using 10^5
   sims
# in each one.

n=10000
d=2  #change d here to reproduce the table (stoch numbers)
B=100;c=1 #B,c and n could be changed
#some values out of the bucle
area.Cd=2^d
Vd_true=pi^(d/2)/gamma(1+d/2)
###############
Vd_vec=rep(0,B)
PB_vec=rep(0,B)
for(ii in 1:B){

  X=matrix(nr=n,nc=d,data=1) # allocate memory

  for(j in 1:d){X[,j]=rnorm(n,0,1)} #sampling in Cd using rnornm
     (0,1)
  #X[i,] must be points distrib as bivariate normal with \Sigma=
     I_d
  suma=0;PointsInBall=0
  for(i in 1:n){
    if(I.Bd(X[i,])==1){PointsInBall=PointsInBall+1}
    kaka=I.Bd(X[i,])/f(X[i,],c)
    suma=suma+kaka
  }

  Vd=suma/n
  Vd_vec[ii]=Vd
  PB_vec[ii]=PointsInBall
  #b) With respect to the Variance of the estimator, we must say
  # that our Xi's are iid variables and therefore
  # V[hat_I] = 1/n int(h(x)-I)^2 f(x) dx
  # where h(x)=I_Bd(x)/f(x) f(x) is the above defined d-variate
     normal
  # pdf. Lets calculate this but only ones
  if(ii==1){
```

```
    suma2=0
    for(i in 1:n){
      kaka=I.Bd(X[i,])/f(X[i,],c)
      suma2=suma2+(kaka-Vd)^2
    }

    Var_estim1sim=suma2/(n^2)
  }

}

# numbers For the report
barVd=mean(Vd_vec)
var_Vd_teo=var(Vd_vec)
barPB=mean(PB_vec)
varPB=var(PB_vec)

#####################################################
#c=1
# Table with the results for part 1
#    d         Vd_true      Vd_stoch_int
#    2         3.14159       3.140442 pm 0.03822546
#    3         4.18879       4.186989 pm 0.08251106
#   10         2.55016       2.637436 pm 2.002591
#Notice that the hipervolume is not strictly increasing with d
   ...

# Table with the results for part 2
#    d         Var_Vd (stoch int)
#    2            0.001461185
#    3            0.006808074
#   10            4.010372

#There must be a mistake very few points inside Bd when d=10 :/
```

```
#Day5exercise3 Stochastic optimization
#part1
#1) Consider the function
h <- function(x){3*dnorm(x,-2,0.2)+6*dnorm(x,1,0.5)+1*dnorm(x
   ,3,0.3)}

xx=seq(-5,5,0.01)
png("d5e3.png",height=300)
par(mfrow=c(1,2))
plot(xx,h(xx),type="l",main='Objective (Opt unif)',
     xlab='x',ylab='h(x)')
#by eye inspection
max.true=-2

#1) Find the maximum of h on [-5,5] using the uniform simulation
```

```
#method and a sample of size 100
n=100
X=runif(n,-5,5)
istar <-which.max(h(X))
max.h.unif=X[istar]

points(X,rep(0,n))
points(max.h.unif,0,col=2,lwd=2)
points(max.true,0,col=3,lwd=2) #looks good


#2)Find the maximum using a non-uniform simul method
#the idea is using a mixture pdf. I know that the area under h
# is 6+3+1=10
integrate(h,-5,5)
Area=10;
barh <- function(x){.3*dnorm(x,-2,0.2)+.6*dnorm(x,1,0.5)+
                      .1*dnorm(x,3,0.3)}
#To sample we need MC methods
n=10000
x=rep(1,n) # allocating x
x[1]=runif(1,-5,5) #initial iteration chosen in [-5,5]
#Metropolis algorithm using runif[-5,5] as proposal
for(i in 1:n) {
  #propose a candidate
  Y <- runif(1,-5,5)
  #compute the ratio
  R <- (barh(Y)*dunif(x[i],-5,5))/
    (barh(x[i])*dunif(Y,-5,5))
  #compute acceptance prob
  p <- min(1,R)
  #We accept or not the candidate randomly
  accept <- rbinom(1,1,p)
  x[i+1] <- ifelse(accept==1,Y,x[i])
}

last=100
X=x[(n-last):n]
#those X might be concentrated in regions of high prob
#(i.e. high values of h)
istar <-which.max(h(X))
max.h.nonunif=X[istar]
#plot
plot(xx,h(xx),type="l",main='Objective (Opt non-unif)',xlab='x',
   ylab='h(x)')
points(X,rep(0,length(X)))
points(max.h.nonunif,0,col=2,lwd=2)
points(-2,0,col=3,lwd=2) #looks good, but those 100 points
                         # are cost to generate
dev.off()
```

```
#This one is looking for the maximizer
optimize(h,c(-5,5),maximum=T)
#But it only finds the local max at x=1 and dont get the global
    :/
```

```
#Day5 Exercise4 Stochastic optimization
#
#Implement the simulated annealing method to maximize h on
    [-3,3]^2
#defined as

h = function(x,y)
  {
  3*dnorm(x,0,0.5)*dnorm(y,0,0.5)+
    dnorm(x,-1,0.5)*dnorm(y,1,0.3)+
    dnorm(x,1,0.5)*dnorm(y,1,0.5)
  }
#visualization
x1<- x2 <-x <-y <-seq(-3,3,0.01)
H=outer(x1,x2,h)
png("d5e4_f1.png")
txt='Contour plot h(x)'
image(x1,x2,H,main=txt)
contour(x1,x2,H,add=T)
dev.off()

#annealing method combine gradient method and tempered
n=10000
#choose determ seq T
T=rep(1,n)
for(i in 1:n){T[i]=1/log(i)}

#aux distrib unif in [-3,3]^2
#init X
X=matrix(nr=n,nc=2,data=0)
X[1,]=c(runif(1,-3,3),runif(1,-3,3))
#iterate
for(i in 1:(n-1)){
  #gi=c(runif(1,-3,3),runif(1,-3,3))
  gi=c(rnorm(1,0,1),rnorm(1,0,1))
  Y=X[i,]+gi
  #restriction to [-3,3]^2
  if(Y[1]<(-3)){Y[1]=-3}; if(Y[2]<(-3)){Y[2]=-3}
  if(Y[1]>3){Y[1]=3};if(Y[2]>3){Y[2]=3}

  dh=h(Y[1],Y[2])-h(X[i,1],X[i,2])
  p <- min(exp(dh/T[i]),1)
  #We accept or not the candidate randomly
  accept <- rbinom(1,1,p)
  X[i+1,1] <- ifelse(accept==1,Y[1],X[i,1])
```

```
  X[i+1,2] <- ifelse(accept==1,Y[2],X[i,2])
}

#suggested visualization
#image(x,y,H,main='Stochastic Sequence')
#lines(X,type='l',lwd=.5)
#points(X[1,1],X[1,2],col=4,pch=16)
#points(X[n,1],X[n,2],col=3,pch=16)

# n=10000 but many rejections we save in z the non repeated
# entries of x cont its elements and plot that one
norm2 <- function(x){sum(x^2)}
cont=1;eps=1.e-6
index=rep(TRUE,n)
for(i in 2:n){
  if(norm2(X[i,]-X[i-1,])>eps) cont=cont+1
  ifelse(norm2(X[i,]-X[i-1,])>eps, index[i] <-TRUE, index[i]<-
     FALSE)
}
print(cont)
z=X[index,]

#Lets plot z points instead

png("d5e4_fig2.png",height=300)
par(mfrow=c(1,2))
image(x1,x2,H,main='Stoch Seq g=bivar Sigma=I_2')
#image(x1,x2,H,main='Stoch Seq g=U[-3,3]^2')
lines(z,type='p',col=1,pch=8)
points(z[1,1],z[1,2],col=4,pch=16)
points(z[length(z[,1]),1],z[length(z[,1]),2],col=3,pch=16)
dev.off()
#I dont think we are outperforming
#standard methods as Newton, steepest descent. Other alternative
    are
#trust region methods. Indeed there are points in the sequence
   that
# are closer to the minim than the last one. That's very sad.
#we can also construct linear constraints in [-3,3]^2 and use
#fmincon, SQP can be applyed to the new problem

#Lets accept the candidate only if dh >0 meaning than the
   sequence
#is s.t. h(X[i+1,])>h(X[i]). There will be even more rejections

X=matrix(nr=n,nc=2,data=0)
X[1,]=c(runif(1,-3,3),runif(1,-3,3))
#iterate
for(i in 1:(n-1)){
  #gi=c(runif(1,-3,3),runif(1,-3,3))
```

```
  gi=c(rnorm(1,0,1),rnorm(1,0,1))
  Y=X[i,]+gi
  #restriction to [-3,3]^2
  if(Y[1]<(-3)){Y[1]=-3}; if(Y[2]<(-3)){Y[2]=-3}
  if(Y[1]>3){Y[1]=3};if(Y[2]>3){Y[2]=3}

  dh=h(Y[1],Y[2])-h(X[i,1],X[i,2])
  X[i+1,1] <- ifelse(dh>0,Y[1],X[i,1])
  X[i+1,2] <- ifelse(dh>0,Y[2],X[i,2])
}


#Looks like a gradient method, many rejections in the algorithm
# n=1000 but many rejections we save in z the non repeated
# entries of x cont its elements and plot that one
norm2 <- function(x){sum(x^2)}
cont=1;eps=1.e-6
index=rep(TRUE,n)
for(i in 2:n){
  if(norm2(X[i,]-X[i-1,])>eps) cont=cont+1
  ifelse(norm2(X[i,]-X[i-1,])>eps, index[i] <-TRUE, index[i]<-
     FALSE)
}
print(cont)
z=X[index,]

#Lets plot z points
png("d5e4_fig3.png",height=300)
par(mfrow=c(1,2))
image(x1,x2,H,main='g=bivar accept if dh>0')
#image(x1,x2,H,main='g=Unif accept if dh>0')
lines(z,type='p',col=1,pch=8)
points(z[1,1],z[1,2],col=4,pch=16)
points(z[length(z[,1]),1],z[length(z[,1]),2],col=3,pch=16)
dev.off()

print(z[length(z[,1]),])
```