Research Skills for Computational Applied Mathematics

Matlab documentation

Dr. Zofia Trstanova
zofia.trstanova@ed.ac.uk

Dr. Kostas Zygalakis
k.zygalakis@ed.ac.uk

February 3, 2018

1. Introduction

This project strictly follows [1]. The programming exercises are written in Matlab code available at `https://github.com/ZofiaTr/MCMC_SanzSerna`.

The main source code is located in folder "Matlab_code". The exercise tasks $1-6$ correspond to functions called "main+task_number+task_name ". Functions used in the main files are saved in folder "functions", which is linked by "addpath" command at the beginning of each file. Figures are saved in folder "figures".

For MATLAB documentation, see `https://uk.mathworks.com/help/matlab/` or type

```
help commandYouWantToKnowAbout
```

in MATLAB.

Other useful tools for writing the report and presentations are LaTeXand beamer package, see `https://en.wikibooks.org/wiki/LaTeX/Presentations`. You can create a project on `overleaf.com`, there are many LaTeXtemplates available.

2. Metropolis Random Walk

2.a. Histogram

The source code is in "main1_sample_MetropolisRW.m". The simulation is described in [1][Section 5.3] and it reproduces [1][Figure 6].

The following code implements one dimensional Metropolis random walk, i.e. for N simulation steps, the new state X^{n+1} is obtained from the previous state X^n using the proposal

$$\tilde{X}^{n+1} = X^n + hZ^n,$$

where $h > 0$ is the step size, Z_n are independent identically distributed continuous random variables in $\mathbb{R}^d$ with $d = 1$ according to a normal (Gaussian) distribution $\mathcal{N}(0, 1)$. The new state is then obtained by Metropolis rule, which defines the acceptance probability a for $\tilde{X}^{n+1}$ as

$$a = \min\left(1, \frac{\rho(\tilde{X}^{n+1})}{\rho(X^n)}\right).$$

If the proposal is accepted, we set $X^{n+1} = \tilde{X}^{n+1}$, otherwise if the proposal is refused, we set $X^{n+1} = X^n$.

In the following example, we use this algorithm to sample from the Boltzmann distribution with density

$$\rho(x) = Z^{-1}\mathrm{e}^{-\beta x^4}, \tag{1}$$

where β is the inverse temperature and Z is the normalization constant such that

$$\int_{\mathbb{R}} \rho(x)dx = 1.$$

The code begins with a common practice in MATLAB to clear all the objects in the workspace, close all figures and clear the command window.

```
clear all;
close all;
clc;
```

The simulation parameters and the distribution are defined in this section, see their description inside the MATLAB comments.

```
% number of steps
N = 1000000;
% step size
h = 1.0;
%inverse  temperature
beta = 1.0;

% potential
V = @(x) x.^4;
% target probability density
rho = @(x) exp(-beta .*V(x));
```

It is possible to define the random number generator by fixing the seed. The code then produces the same sequence of random numbers at every execution. This can be useful for example for debugging.

```
seed=0;
rng(seed);
```

Once we have initialized the arrays to store the samples, we can start the main iteration of N steps to perform the sampling. Note that for small systems, it is more efficient to precompute the array of random numbers.

```
% initialize array of samples
X = zeros(1, N);
% intialize randon numbers, see help randn, help rand
Z = randn(1, N);
U = rand(1, N);

fprintf('Sampling RW\n')

for n = 1 : N - 1
    % proposal
    X(n+1) = X(n) + h * Z(n);

    %Metropolis step: acceptance ratio
    acceptanceRatio = rho(X(n+1)) / rho(X(n));

    % metropolis rule
```

```
    if (acceptanceRatio < U(n+1))
        % refuse
        X(n+1) = X(n);
    end

end
fprintf('sampling done\n')
```

Note that the Metropolis rule tests the condition that the proposal is rejected. Since we have already assigned the proposal to the array X at index $n+1$ in the line before, if the proposal is accepted, there is nothing to be done. Testing the refusal condition is more efficient, because in the case of refusal we only assign the value from the previous step to X_{n+1}.

In the postprocessing part, we create a histogram of samples X^n. Note that in order to make all figures consistent, it is a good idea to define parameters, which, for example, fix the font size.

```
%% show histogram
myFontSize = 14;

f6 = figure(6);
h = histogram(X, 20, 'Normalization','probability');
xlabel('X', 'FontSize', myFontSize)
ylabel('Probability', 'FontSize', myFontSize)
set(gca, 'FontSize', myFontSize)

%save the figure
print(f6,'figures/figure6','-dpng')
```

2.b. Tasks

1. Plot the trajectory X^n over n steps.
2. Fit the target density into the histogram.
3. Comment out the Metropolis step, and rerun the sampling only with the random walk proposal, create plot of X^n over n steps.

3. Autocorrelation

The source code is in "main2_sample_autocorrelation.m". The simulation reproduces [1][Figure 7].

This code performs the sampling of the distribution (1) by Random Walk Metropolis Algorithm (see previous section). The main difference with the structure of "main1_sample_MetropolisRW.m" is that the sampling part is wrapped up as a function

called "sample_MetropolisRW" which is saved in folder "functions". This function takes number of steps to be performed N, steps size h, distribution that should be sampled ρ and initial state X_0 and returns an array of N samples X and an array of rejections of size N.

```
function [X, rejections] = sample_MetropolisRW(N, h, rho, X0)
% parameters :
%
%   number of steps N
%   stepsize h
%   rho is target density
%   initial condition X_0, size(1, d)
% return : trajectory and number of rejections

% determine dimension from the initial condition
d = length(X0);
% initialize array of samples
X = zeros(d, N);
X(:,1) = X0;
rejections = 0;

% intialize randon numbers, see help randn, help rand
Z = randn(d, N);
U = rand(1, N);

for n = 1 : N - 1
    % proposal
    X(:,n+1) = X(:,n) + h * Z(:,n);
    %Metropolis step: acceptance ratio
    acceptanceRatio = rho(X(:,n+1)) / rho(X(:,n));
    % random number
    if (acceptanceRatio < U(n+1))
        % refuse
        X(:,n+1) = X(:,n);
        rejections = rejections+1;
    end
end
end
```

The main code "main2_sample_autocorrelation.m" then calls this function to generate samples. The first part of the code is as before:

```
% clean the working space
clear all;
close all;
```

```
clc;

addpath([pwd,'/functions']);

% choose initial seed, comment out to turn off, see help rng
seed=0;
rng(seed);
myFontSize = 14;

%inverse  temperature
beta = 1.0;
% potential
V = @(x) x.^4;
% target probability density
rho = @(x) exp(-beta .*V(x));
```

We want to investigate several step sizes, we therefore have an array of step sizes h, which will be iterated over later on.

```
% step size
h = [0.5, 1, 2, 4];
% number of various stepsizes
nrh = length(h);
% number of steps
N = 1000000;
%initial condition
X0=0;
```

In order to compute the empirical auto-covariance, we choose the lag array. We also initialize a figure to allow writing inside during the iterations over the step sizes.

```
lag  = 0:19;

f7 = figure(7);
hold on
% initialize for loop counter
i=0;
```

What follows is the main for-loop over the step sizes: for every step size h, we perform the sampling and compute the empirical auto-correlation. The implementation of the function *compute_empirical_auto_correlation_coeff* is one of the tasks (see the end of this section). Finally, within each iteration, we plot the auto-covariance in a subplot belonging to the pre-intialized figure.

```
for stepSize = h
    fprintf('Sampling with step size h = %f\n', stepSize);
```

```
    % increase the loop counter
    i = i+1;
   % perform the sampling with given parameters
    [X, rejections] = sample_MetropolisRW(N, stepSize, rho, X0);
    %%%%% compute_empirical_auto_correlation_coeff
    rho_nu = compute_empirical_auto_correlation_coeff(X, lag);
    %%%%%%%%%%%%%%%%%%%%%%%%

    subplot(2,2,i)
    plot(lag, rho_nu, 'LineWidth', 2)
    xlabel('lag', 'FontSize', myFontSize)
    ylabel('Correlation', 'FontSize', myFontSize)
    ylim([0,1])
    xlim([0,max(lag)])
    legend(['h = ', num2str(stepSize)])
    title([num2str(rejections), ' rejections, rate ', num2str(rejections / length(X))])
    set(gca, 'FontSize', myFontSize)

end
%% save figure
print(f7,'figures/figure7','-dpng')
```

3.a. Tasks

1. Implement auto-covariance function called "compute_empirical_auto_correlation_coeff" which takes array of samples X and lag and returns empirical auto-covariance computed according to the formula at the end of [1][Section 5.3]:

$$\rho_\nu = \frac{\gamma_\nu}{\gamma_0},$$

   where

$$\gamma_\nu = \frac{1}{N+1}\sum_{i=0}^{N-\nu}(x_i - \hat{m})(x_{i+\nu} - \hat{m}), \quad \hat{m} = \frac{1}{N+1}\sum_{i=0}^{N} x_i.$$

2. Compare the empirical auto-covariance with the values obtained with Matlab's "autocorr" function and plot the results together in one figure.

3. Try out more values for the step size h. Comment on what you observe.

4. Brownian motion

The source code is in "main3_BrownianMotion". The simulation reproduces [1][Figure 8]. There are no tasks associated with this file.

5. Euler-Maruyama

The source code is in "main4_sample_EulerMaruyama". It contains an implementation of the Euler-Maruyama method and the simulation reproduces [1][Figure 9].

In this example, we consider a one dimensional stochastic differential equation of a form

$$dX_t = X_t dt + dB_t, \quad t > 0, \quad X_0 = 1,$$

where B_t is one dimensional Wiener process.

The Euler-Maruyama discretization is

$$X^{n+1} = X^n + \Delta t X^n + \sqrt{\Delta t} Z^n,$$

where $Z^n \sim \mathcal{N}(0, 1)$.

Euler-Maruyama is implemented in function "functions/sample_EulerMaruyama_linearDrift.m".

```
function X = sample_EulerMaruyama_linearDrift(N, dt, X0)
% Euler-Maruyama discretization
% parameters : int N, number of steps
%              double dt, time step size
%              initial condition X0 of size (d,1)
% return : trajectory X, array (1,N)

  % dimesion
  d = length(X0);
  % initialize array of samples
  X = zeros(d, N);
  % initial condition
  X(1,:) = X0;
  % intialize randon numbers, see help randn, help rand
  Z = randn(d, N);

  for n = 1 : N - 1

        X(:,n+1) = X(:,n)* (1 + dt) + sqrt(dt) * Z(:,n);
  end
end
```

5.a. Tasks

1. Turn off the noise and compare the numerical trajectory with the exact solution for the corresponding ODE.

2. Using the Euler-Maruyama discretization, simulate $100,000$ trajectories in $0 \leq t \leq 1$ with step-size $\Delta t = 0.001$, and record the value of $X_t = 1$ for each trajectory. Produce an histogram by distributing those $100,000$ values into 15 bins centered

at $-4, -3, \ldots, 10$. Fit a Gaussian density with mean e and variance $(e^2 - 1)/2$ to the distribution of $X_t = 1$, resulting into Figure **??**.

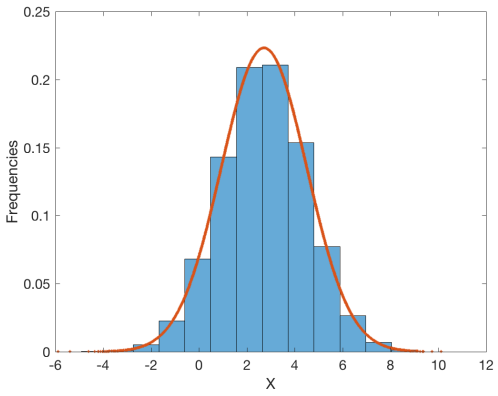

Figure 1: Histogram of $X_t = 1$.

6. Comparison of Random-Walk Metropolis and MALA

The source code is in "main5_sample_comparisonMALAandRW". The simulation reproduces [1][Figures 11 and 12]. This section should provide an introduction on the MALA algorithm and on the efficiency comparison of two methods.

We seek to sample the Boltzmann distribution with density $\mathrm{e}^{-V(x)}$ in d dimensions with energy

$$V(x) = \frac{k}{2}(r(x) - 2)^2, \quad r(x) = \|x\|_2, \quad k = 100.$$

The Metropolis adjusted Langevin algorithm (MALA) takes as a proposal the Euler-Maruyama discretization of (under this distribution) invariant the SDE

$$dX_t = -\frac{1}{2}\nabla V(X_t)dt + dB_t,$$

and corrects the sample by Metropolis-Hastings rule. More precisely, the proposal is

$$\tilde{X}^{n+1} = X^n - \frac{\Delta t}{2}\nabla V(X^n) + \sqrt{\Delta t}Z^n,$$

which is accepted with probability

$$A_{\Delta t} = \min\left(\frac{\rho(\tilde{X}^{n+1})T(\tilde{X}^{n+1}, X^n)}{\rho(X^n)T(X^n, \tilde{X}^{n+1})}, 1\right),$$

with

$$T(X, \tilde{X}) = \left(\frac{1}{2\pi\Delta t}\right)^{d/2} \exp\left(-\frac{\|\tilde{X} - X + \frac{\Delta t}{2}\nabla V(X)\|^2}{2\Delta t}\right)$$

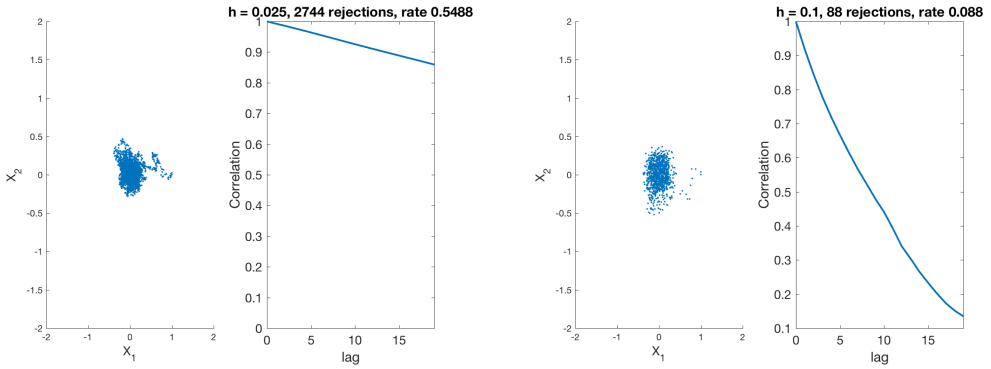

Figure 2: Comparison of Metropolis Random Walk (Left) and MALA (right). MALA outperforms Metropolis RW.

In order to compare the efficiency of the MALA algorithm with the random walk Metropolis-Hastings, we compare their empirical auto-covariance coefficients. The results in Figure 1 suggest that Random walk Metropolis performs worse than MALA. Note that we have been tracking the average rejection, which is a very useful observable for Metropolized schemes and provides an information on the efficiency of generating successful proposals and can be tuned by changing the time step size. The bigger the time step, the bigger the rejection rate, i.e. the more proposals are refused and the correlations between the samples increase. On the other hand, a very small rejection rate might not be desirable neither, because it might require very small step sizes and therefore slow down the exploration again. Therefore, one should think about a good compromise, which will provide the best exploration.

6.a. Tasks

1. Explain why can you see from Figure 1 that MALA is better than Metropolis Random Walk.

2. Find the most efficient time step size for MALA (you might want to create a separate MATLAB main file, use "main5_sample_comparisonMALAandRW" as template) and name it main_MALA_findBestStepSize:
   - Write a "for-loop" iteration over time step sizes $h = 0.01 : 0.01 : 0.2$ and simulate 10000 steps of MALA.
   - Fix 20 lag values between 0 and 19 and compute the empirical auto-covariance coefficient ρ_ν for every trajectory with step size h and save the end values, i.e. ρ_ν at $\nu = 19$.
   - Generate a plot of the average rejection rate and ρ_{19} over the step sizes h. Which h is optimal?

7. Comparison of Random-Walk Metropolis, MALA and HMC

The source code is in "main6_sample_comparison_RW_MALA_HMC". The simulation reproduces [1][Figures 14, 15 and 16].

The Hybrid Monte Carlo algorithm (also called Hamiltonian Monte Carlo or in short HMC) is described in [1][Section 9].

```
function [X, rejections] = sample_HMC(N, dt, T, V, dV , X0)
% HMC
% parameters : int N, number of steps
%              double dt, time step size
%              potential V, function
%              dV gradient of the potential V, function
%              initial condition X0 of size (d,1)
% return : trajectory X, array (d,N)

rejections =0 ;
% determine dimension from the initial condition
d = length(X0);
% initialize array of samples
X = zeros(d, N);
X(:,1) = X0;
% initialize array of momenta
p = zeros(d, N);

% Hamiltonian defined for X, p \in R^d
H = @(q,p) V(q) + 0.5*p'*p;

U = rand(1, N);

% number of inner steps
L = floor(T/dt);
if (L ==0)
    L=1;
end

%sample N steps
for n = 1 : N - 1

    % resample momentum
    pRand = normrnd(0,1, size(X0));
    %save previous values of positions and momenta
    xn = X(:,n);
    pn = pRand;
```

```
    % save previous state
    xnOld = xn;
    pnOld = pn;

    % perform L steps of deterministic dynamics
    for ni = 1: L
        % proposal
        [X_proposal, p_proposal] = sample_Verlet(2, dt, dV, xn, pn);
        xn = X_proposal(:,end);
        pn = p_proposal(:,end);
    end

    %compute acceptance probability
    acceptanceProba =  min(1, exp(-H(xn,pn) + H(xnOld,pnOld)));

    %Metropolis rule
    if ( acceptanceProba < U(n+1))
        % refuse
        X(:,n+1) = xnOld;
        % reverse momentum
        p(:,n+1) = -pnOld;
        rejections = rejections+1;
    else
        % accept
        X(:,n+1) = xn;
        p(:,n+1) = pn;

    end
end
end
```

7.a. Tasks

1. Modify the code for HMC, so that the time-step Δt can be random.
2. Prove that for $\pi(dx, dp) = \mathrm{e}^{-\beta V(x)}\mathrm{e}^{-\beta\frac{1}{2}p^T M^{-1}p}dxdp$ with a potential energy V such that
$$\int_\Omega \mathrm{e}^{-\beta V(x)}dx < \infty$$
and a constant mass matrix M, it holds that
$$\mathbb{E}_\pi[p^T M^{-1}p] = \beta^{-1}.$$
3. In the code "main6_sample_comparison_RW_MALA_HMC" use the by HMC generated momenta trajectory $(p^n)_{n=1,\dots,N}$ to compute the mean value of the kinetic

energy, i.e.

$$T_{\text{kin}}^N = \frac{1}{dN} \sum_{n=0}^{N-1} (p^n)^T (p^n), \quad p^n \in \mathbb{R}^d.$$

To what value does T_{kin}^N converge as N goes to infinity?

MATH11197: Research Skills in Computational Applied Mathematics.

Individual report This assignment counts for 25% of the total assessment for the module. The general aim of this project is to illustrate that you have understood the material of the paper [1] through completing the individual tasks described in the `MATLAB` documentation. You should typeset your work using LaTeX. Your report should not be more than 12 pages long including references (11pt fonts). Your report should also contain an introduction to the topic and should explain clearly the different steps you followed in order to complete the different tasks. A conclusions section summarising your findings and discussing some other possible studies you could perform should also be included.

Your work will be assessed according to the following criteria:

- **Exposition** (mathematical accuracy, clarity, literary presentation, degree of coverage of the topic) 40%;
- **Literature** (understanding, relating different sources, finding new sources) 20%;
- **Originality** (examples cited or constructed, new treatments and proofs of standard results, simple generalisations, original researches) 20%;
- **Scope of topic** (conceptual and technical difficulty, relationship with previous studies, relevance of material included) 20%.

Please refer to the grade descriptors for more information

Deadline The assignment is to be handed in to MTO by Friday the 23rd of February at 12am. You will also need to submit your `MATLAB` code online through learn.

References

[1] JM Sanz-Serna. Markov chain monte carlo and numerical differential equations. In *Current challenges in stability issues for numerical differential equations*, pages 39–88. Springer, 2014.