Research Skills for Computational Applied Mathematics

Matlab documentation

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

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

January 24, 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 also many templates 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

$$\tilde{X}_{n+1} = X_n + Z_n,$$

where 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

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

Otherwise the proposal is refused and 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, their description is within 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')
```

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. Fit the target density on the histogram, create plot of Y_n over n steps.
2. Comment out the Metropolis step, and rerun the sampling only with the random walk proposal, create plot of Y_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 at the bottom 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].
2. Compare with the correlation obtained with Matlab's "autocorr" function and plot the results together.
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].

5. Euler-Maruyama

The source code is in "main4_sample_EulerMaruyama". The simulation reproduces [1][Figure 9]. The method is described in [1][Section 6.3].

5.a. Tasks

1. Turn off the noise and compare with the exact solution for the corresponding ODE.
2. Following the simulation setting described in [1][Section 6.3], reproduce [1][Figure 10].

6. Comparison of Random-Walk Metropolis and MALA

The source code is in "main5_sample_comparisonMALAandRW". The simulation reproduces [1][Figures 11 and 12].

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].

7.a. Tasks

1. Modify the code for HMC, so that the time-step Δt can be random.

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.

MATH11197: Research Skills in Computational Applied Mathematics.

Individual report

First part: You are asked to complete the tasks outlined in the matlab documentation.In particular, marks associated with each task are

- Task 2.b (1) 5% (2) 5%
- Task 3.a (1) 5% (2) 5% (3) 10%
- Task 5.a (1) 5% (2) 5%
- Task 7.a (1) 5%

Second part: *Application to sampling in high dimensions*
Consider the problem of sampling from a high dimensional Gaussian with covariance matrix Σ given by

$$\Sigma = \text{diag}(\sigma_1^2, \cdots, \sigma_{100}^2), \quad \sigma_i = (i/100)^2.$$

1. Write down the Hamiltonian equations of motion using a quadratic kinetic energy function. In particular your Hamiltonian should be of the form

   $$H = \frac{p^T p}{2} + V(x),$$

   where $V(x)$ needs to be chosen in the appropriate way in order to ensure that one is sampling from the desired distribution. [5%]

2. Write down the discretization of the Hamiltonian equations using the leapfrog method [5%]

3. By studying the corresponding one step map, deduce what is the maximum Δt that one is allowed to use when discretising the Hamiltonian equations. [10%]

4. Compare the performance between the Random Walk Metropolis and the Hamiltonian Monte Carlo (with deterministic and random choice of Δt). Possible diagnostic tests include [35%]

   a) Plot the outputs for different co-ordinates for the RWM and HMC

   b) Studying the autocorrelation function for different co-ordinates and different methods

   c) Plotting the sample mean as a function of the different co-ordinates

   d) Plotting the standard deviation as a function of the different co-ordinates

   e) Plotting the distribution of the MCMC method against the true distribution (you can do this component wise since every component is independent of each other)

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.