Research Skills for Computational Applied Mathematics

Matlab documentation

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

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

January 20, 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".

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

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

Otherwise the proposal is refused and we set $X_{n+1} = X_n$. We use it to sample from the Boltzmann distribution with density

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

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

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

The code begins with a common practice in MATLAB, which clears all the objects in the workspace, closes all figures and finally clears 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].

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.

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].

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.