Exercises in Optimization
(ACM 40990 / ACM41030)

Dr Lennon Ó Náraigh

Exercises #4

Exercises #4 – Global Optimization

1. The Metropolis Algorithm can be used to generate random numbers from an arbitrary distribution. In this exercise, you should write a computer code to generate a sequence of numbers from the exponential distribution,

$$f(x;\lambda) = \begin{cases} \lambda \mathrm{e}^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases} \tag{1}$$

   Here, λ is a positive constant, which you should fix for the exercise.

Code listings are shown below:

- `MHsimple.m`
- `runMH.m`

```
function [X, acc] = MHsimple()

% Computer code to generate numbers drawn from a
% target distribution.  Uses the MH algorithm.
Inspiration from:
%
% https://stephens999.github.io/fiveMinuteStats/...
% MH_intro.html
% http://physics.ujep.cz/~mmaly/vyuka/poc_fyz_1/...
% zdroje/MeropolisAlg_clanek.pdf

% ************************************************
% Numerical parameters:

burnin = 100; % number of burn-in iterations
```

```
nsamp = 10000; % number of samples to draw
sig = 5; % standard deviation of Gaussian proposal
x = 1; % start point

% ************************************************

% Store samples drawn from the Markov chain:
X = zeros(nsamp,1);
% Store the following vector to track
% the acceptance rate:
acc = [0 0];

% ************************************************
% Burn-in:

for i = 1:burnin
    % Update chain:
    [x,a] = MHstep(x,sig);
    % Track accept-reject status
    acc = acc + [a 1];
end

% ************************************************
% MH routine

for i = 1:nsamp
    % Update chain:
    [x,a] = MHstep(x,sig);
    % Track accept-reject status
    acc = acc + [a 1];
    % Store the i-th sample:
    X(i) = x;
end

end

% ************************************************

function [x1,a] = MHstep(x0,sig)
    % Generate candidate from Gaussian:
    xp = normrnd(x0,sig,1);
    % Compute acceptance probability:
    accprob = targetdist(xp) / targetdist(x0);
    u = rand; % uniform random number
    if u <= accprob % if accepted
```

```
        x1 = xp; % new point is the candidate
        a = 1; % note the acceptance
    else % if rejected
        x1 = x0; % new point is the same as the old one
        a = 0; % note the rejection
    end
end

% ****************************************************
% ****************************************************

function probX = targetdist(x)

    lambda=1;

    if(x<0)
        probX=0;
    else
        probX = lambda*exp(-lambda*x);
    end
end
```

The code is run from the following script:

```
% Matlab script to run the code
% "MHsimple.m" and plot the results.

[X,acc] = MHsimple();
xx=0:0.1:10;
lambda=1;
hold on

histogram(X,'Normalization','pdf')
plot(xx,exp(-xx),'linewidth',2,'color','red')
% set(gca,'yscale','log')
grid on
xlim([0 10])
xlabel('x')
ylabel('p(x)')

arate=floor(100*acc(1)/acc(2));
display(strcat('acceptance rate=',num2str(arate)),'%')
```

Results (x-values) are shown in Figure **??**.

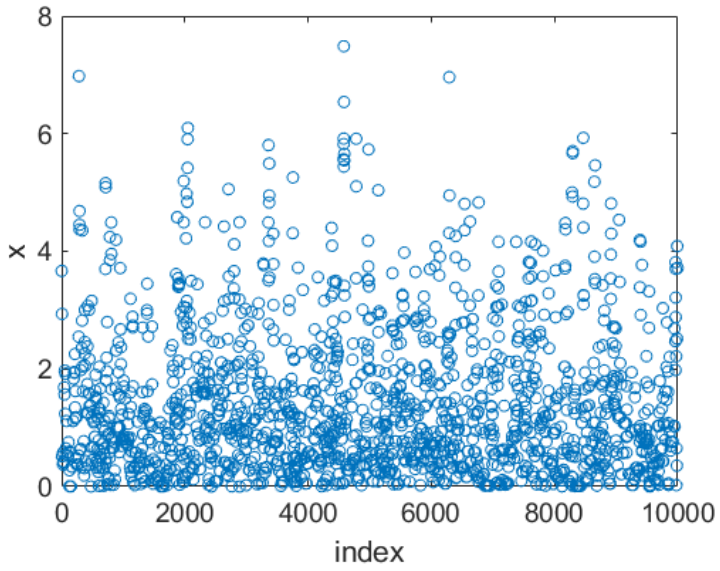

Figure 1: x-values from the exponential distribution, generated from the MH algorithm. Parameter: $\lambda = 1$.

A histogram of the x-values is generated and shown in Figure **??**.

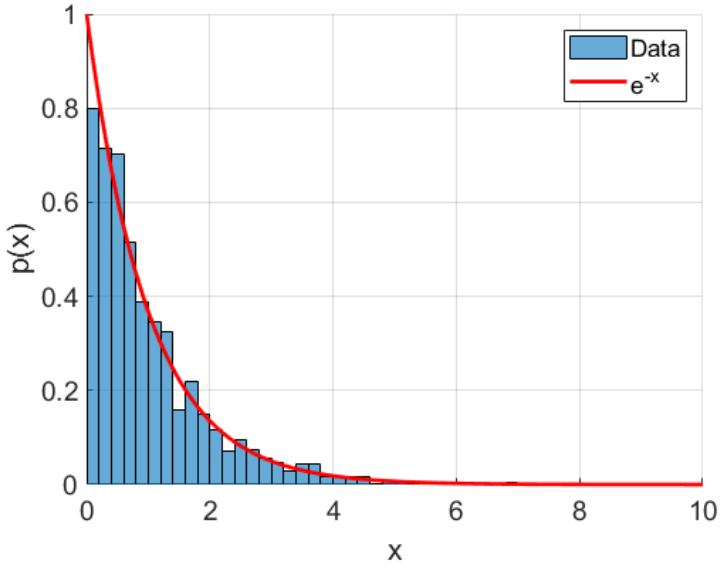

Figure 2: Histogram of x-values showing comparison with the exponential distribution.

2. Write a computer code to compute the global minimum of the cost function

$$f = \frac{\sin(x)}{x^2 + 10}.$$

Compare your result with the built-in SA algorithm in Python / Matlab.

The computer code has already been developed in the typed notes and the global minimum estimated. There is no need to repeat this calculation here. Instead, we show how to implement the built-in SA algorithm in Matlab, this requires the installation of the 'global optimization' toolbox. Code listings are shown below, and are also available in the online repository (`SA_matlab.m`).

```
function [x_star,fval]=SA_matlab()

% For reproducibility:
rng default

% Function handle for cost function:
CostFunction = @myfun;

% Starting point
x0 = 3;

% ************************************************
% Call SA solver, use verbose mode:

options = optimoptions('simulannealbnd',...
    'FunctionTolerance',1e-8,'PlotFcns',...
     {@saplotbestx,@saplotbestf,@saplotx,@saplotf});

[x_star,~] = simulannealbnd(CostFunction,x0,[],[],options);

% After getting a rough idea of the position of
% the global minimum using SA, I now refine the
% estimate using local optimization methods:

[x_star,fval] = fminunc(CostFunction,x_star);

% ************************************************

    function y=myfun(x)
        y=sin(x)/(x*x+10);
    end
```

```
end
```

Some coaxing is required to produce the correct final answer. In particular, the function tolerance needs to be set at 10^{-8}, to avoid premature termination of the algorithm at a local minimum. Also, SA really just gives a good estimate of the global minimum, and the present example is no exception. Hence, in this example, once a good estimate of the global minimum is found, this is taken as the initial guess for a local optimization, which then produces a more precise estimate of the global minimum, $x \approx -1.3466$.