EE511: Project 5

Ming-Chang Chiu 6947046287

December 9, 2016

1 [Monte Carlo]

Description: Generate n=100 samples of i.i.d 2-dimensional uniform random variables in the unit-square. Count how many of these samples fall within the quarter unit-circle centered at the origin. This quarter circle inscribes the unit square as shown below: i.) Use these random samples to estimate the area of the inscribed quarter circle. Use this area estimate to estimate the value of pi. Do k=50 runs of these pi-estimations. Plot the histogram of the 50 pi-estimates. ii.) Repeat the experiment with different numbers of uniform samples, n (using k=50 for all these runs). Plot the sample variance of the pi-estimates for these different values of n. What is the relationship the estimate variance and Monte Carlo sample size? iii.) Adapt your Monte Carlo solution to provide integral and error estimates for the function: $g(x, y) = |4x - 2||4y - 2|$ x, y in [0, 1]

In fig.1, we can see the estimation of π, which is close to 3.1 but not very accurate because the sample size is only 100. In fig.2& 3, we can observe the bin size and see that we are closer to the the real value of π because the sample size expands. In addition, judge from fig.4, the variance of the sample decays exponentially with the growth rate of sample size (i.e. sample size grows 10 times leads to error reduces to one-tenth), which means with more samples the error will be smaller. This is again an example of Law of Large Number, the more the sample size, the better the approximation. Using the same method for estimating the integral of g(x,y), the result is roughly 0.9696, very close to the real value, 1, and the sample variance is 0.0266 when n = 100.

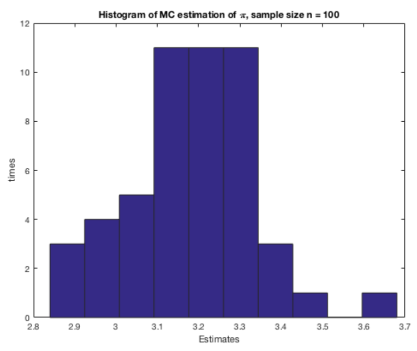


Figure 1

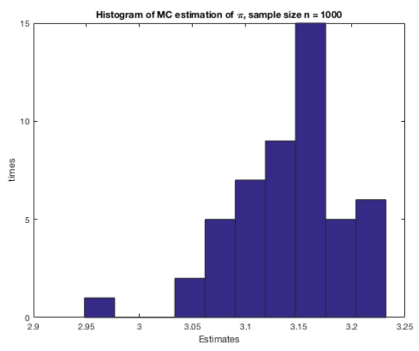


Figure 2

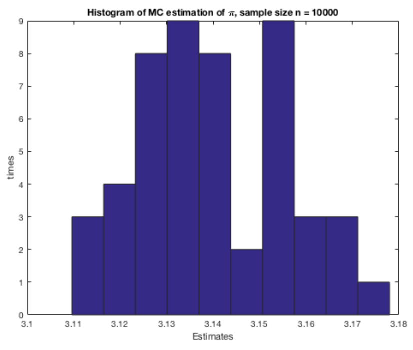


Figure 3

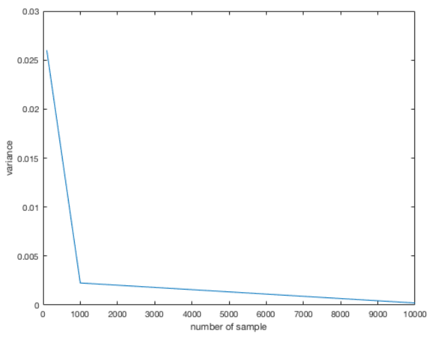


Figure 4

2 [Variance Reduction Methods for Monte Carlo]

Description: Use a total sample budget of n=1000 to obtain Monte Carlo estimates and sample MC estimate variances for the definite integrals in 2 dimensions (x1, x2): (a) $exp(\sum_x 5|x-0.5|)$ for x_i in [0,1] (b) $cos(\pi + \sum_x 5x_i)$ for x_i in [-1,1] (c) $|4x-2||4y-2|$ for x, y in [0,1] Implement stratification and importance sampling (separately) in the Monte Carlo estimation procedures using the same sample budget n=1000. Compare the 3 different Monte Carlo integral estimates and their sample variances. Discuss the quality of the Monte Carlo estimates from each method.

In this problem, I choose another method to implement simple MC by directly using the property of Law of Large Number. And for stratification MC, I choose the number of strata to be 10 for each dimension, so in total 100 strata. For function (a), the actual value is 20; the simple MC gives 20.01 with variance 0.2255; the stratified MC gives 20.016 with variance 0.04; the importance sampling MC gives 20.9 with variance 1.288e-29. For function (b), the actual value is -0.1471; the simple MC gives -0.0366 with variance 0.0004; the stratified MC gives -0.037 with variance 6.4686e-05; the Metropolis-Hastings MC gives -0.0348 with variance 0.0014498. For function (c), the actual value is 1, the simple MC gives: 0.99621 with variance 0.0008; the stratified MC gives 0.99857 with variance 1.9416e-05; the importance sampling gives 0.94941 with variance 1.132e-31.

In sum, the error of importance sampling is massively better than the other two, and stratification MC is also better than simple MC, but the estimation of importance sampling is not necessarily the best. This may be resulted from the choice of proposal distribution. The better the proposal, the better the result.

For function (a)&(c) we can make use of symmetry of the function, for instance, (a) can be factored as $exp(5|x-0.5|) * exp(5|y-0.5|)$. If we do importance sampling MC for 1-D first (i.e. $exp(5|x-0.5|)$) and then square the result to get the result for original function. Also, the shape in 1-D for function (a)&(c) are similar, so I choose the same proposal PDF for them, which is $c(x-0.5)^2, cstandsfornormalizationfactor$. In addition, in Metropolis-Hastings sampling for function (b), I use 2-D normal distribution as proposal distribution and the absolute of function (b) as target distribution; the later is because a probability density function can never be negative. It turns out that Metropolis-Hastings method has similar integral but the variance is not better the other methods.

Among the simple MC, stratified MC, importance sampling and Metropolis Hastings sampling, I prefer using Metropolis Algorithm because to me it is easier and does not have too many constraints. Most importantly, the proposal distribution can be more flexibly chosen.

3 [MCMC for Optimization]

Description: The n-dimensional Schwefel function $f(\vec{x}) = 418.9829n - \sum_{i=1}^{n} x_i sin(\sqrt{|x_i|})$, $x_i in[-500, 500]$. In this problem n = 2. i) Plot a contour plot of the surface for the 2-D surface ii) Implement a simulated annealing procedure to find the global minimum of this surface iii) Explore the behavior of the procedure starting from the origin with an exponential, a polynomial, and a logarithmic cooling schedule. Run the procedure for t=20, 50, 100, 1000 iterations for k=100 runs each. Plot a histogram of the function minima your procedure converges to. iv) Choose your best run and overlay your 2-D sample path on the contour plot of the Schwefel function to visualize the locations your optimization routine explored.

The global minimum is known as $f(420.9687, 420.9687) = 0$. The global minimum found using MCMC simulated annealing with exponential cooling schedule is $f(422.18, 419.63) = 0.4129$. Note that simulated annealing is not a deterministic method; each run of the method may have different result. The histograms of function minima (fig.5-16) generated by different cooling schedules show that in my implementations of simulated annealing, exponential cooling and polynomial cooling have the best result because they generates more points around the real global minimum, but, again, they all perform well when t is large (i.e. number of iterations during optimization). And the contour–exponential cooling minima overlay plot is fig.17; note that the deeper the contour color is, the smaller the values are. Also, a sample path explored by MCMC simulated cooling is also plotted as fig.18.

Note: The method used to generate new random point is simple: condition on the present coordinate, uniformly generate the new point centered at present value's 600x600 rectangular area. Also, the detail of each cooling schedule can be seen from my code snippet. They all converge to 0 in the end, and so they are all valid cooling schedule.

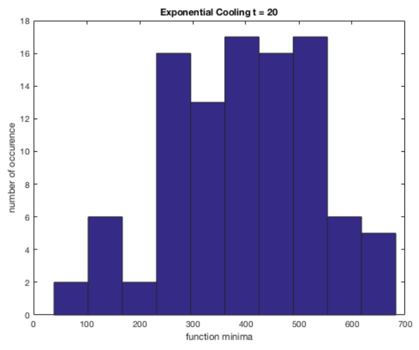


Figure 5

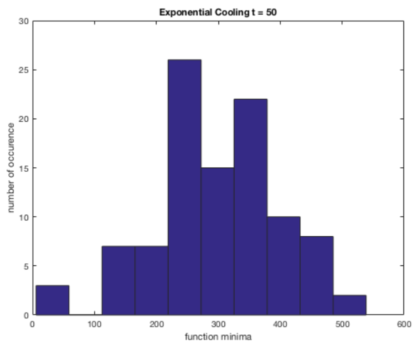


Figure 6

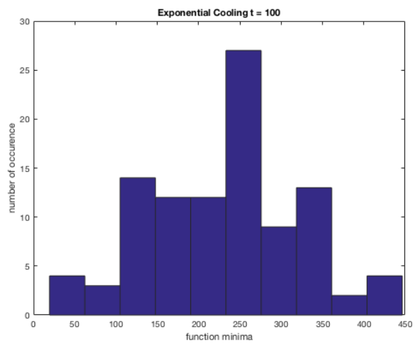


Figure 7

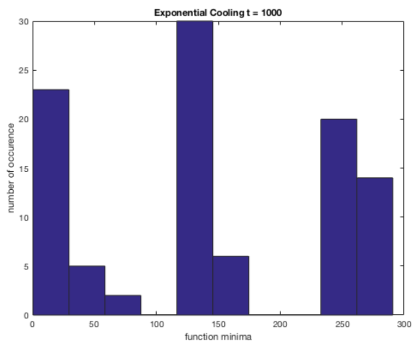


Figure 8

4 Code

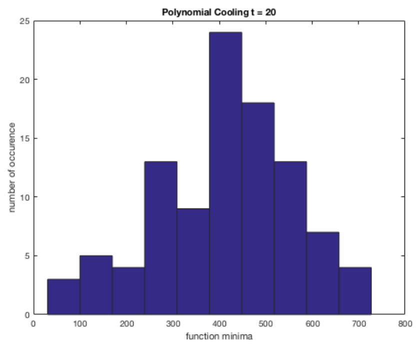


Figure 9

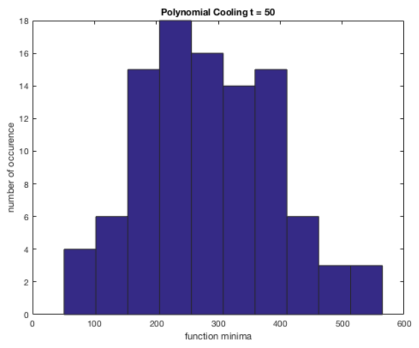


Figure 10

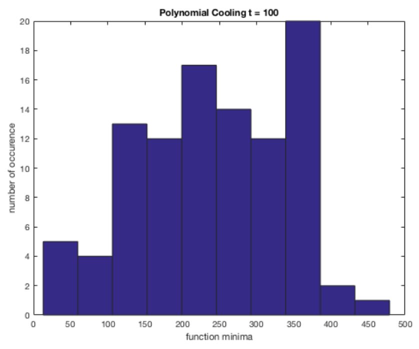


Figure 11

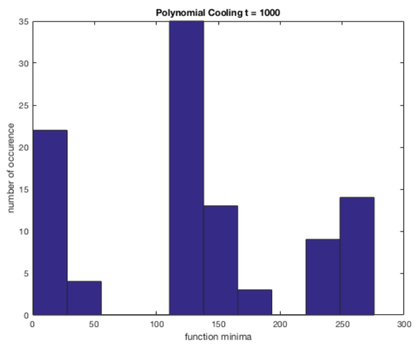


Figure 12

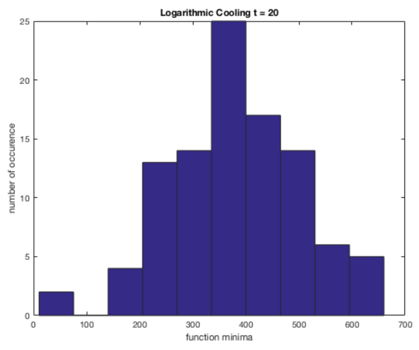


Figure 13

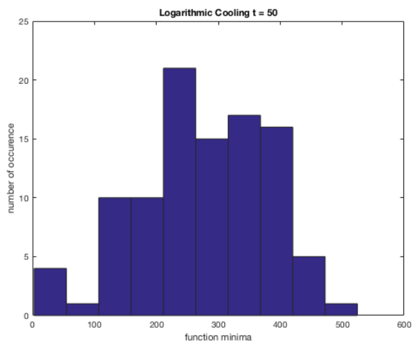


Figure 14

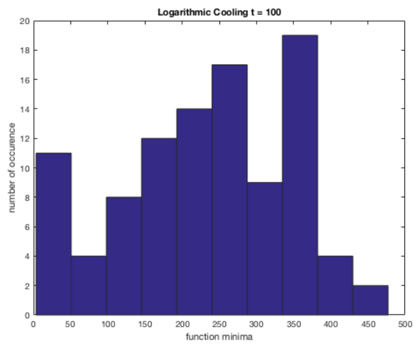


Figure 15

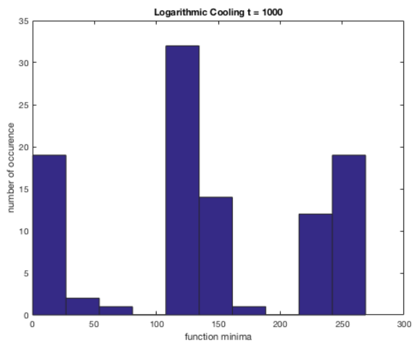


Figure 16

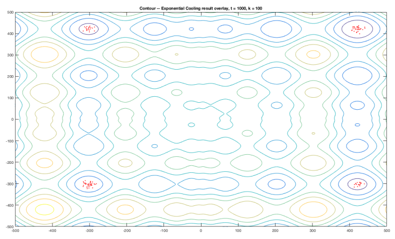


Figure 17

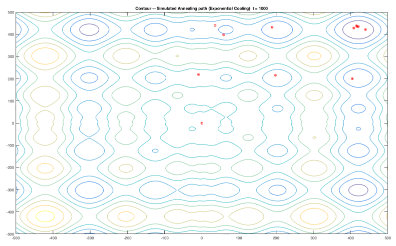
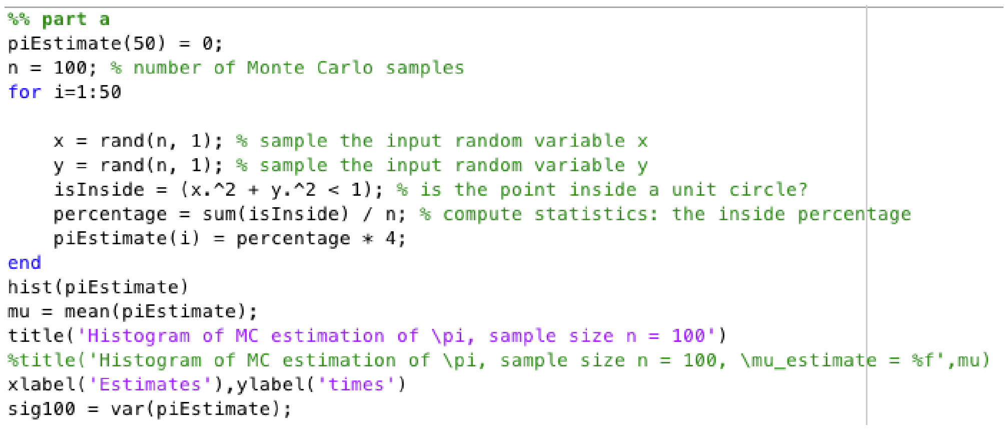
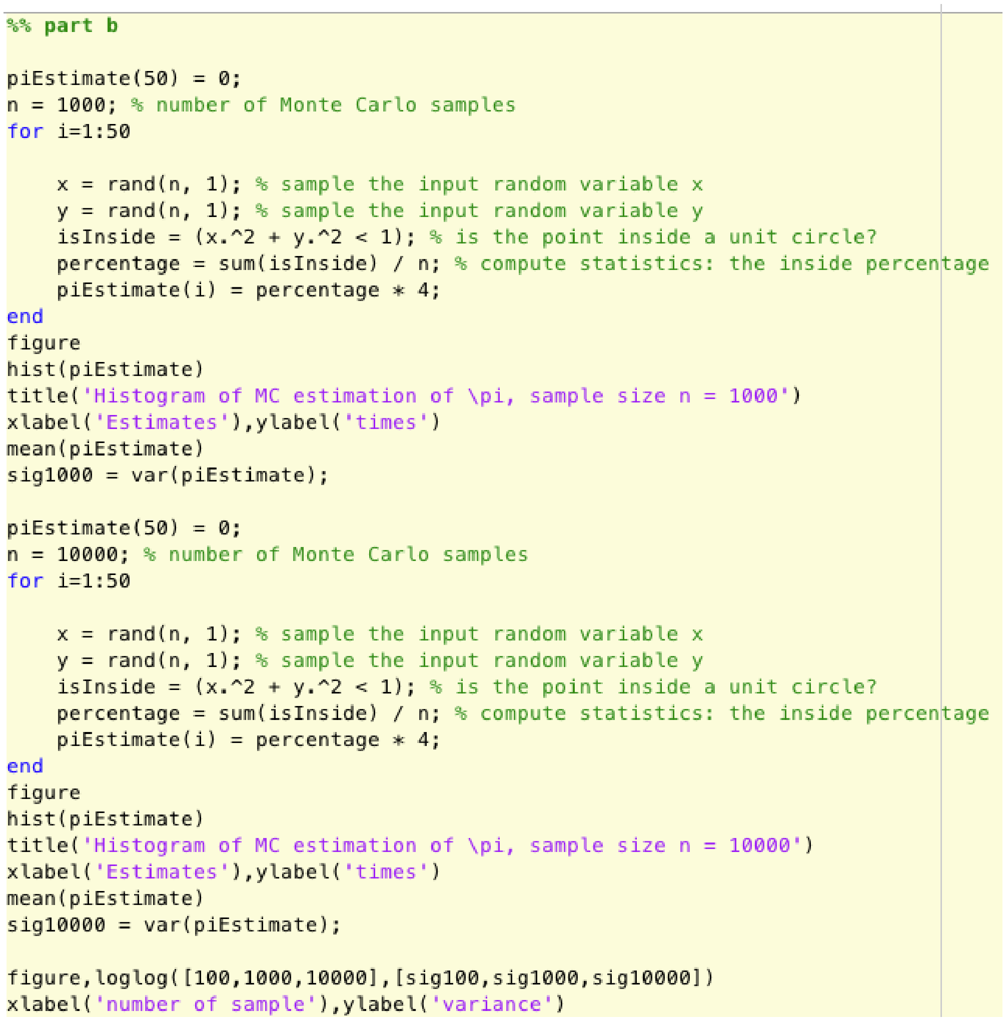


Figure 18



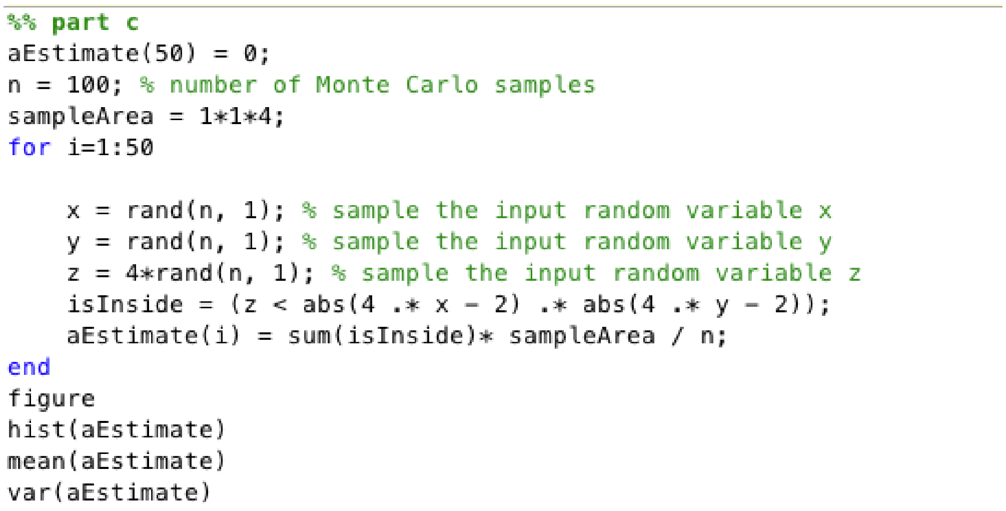
```
%% part a
piEstimate(50) = 0;
n = 100; % number of Monte Carlo samples
for i=1:50

    x = rand(n, 1); % sample the input random variable x
    y = rand(n, 1); % sample the input random variable y
    isInside = (x.^2 + y.^2 < 1); % is the point inside a unit circle?
    percentage = sum(isInside) / n; % compute statistics: the inside percentage
    piEstimate(i) = percentage * 4;
end
hist(piEstimate)
mu = mean(piEstimate);
title('Histogram of MC estimation of \pi, sample size n = 100')
%title('Histogram of MC estimation of \pi, sample size n = 100, \mu_estimate = %f',mu)
xlabel('Estimates'),ylabel('times')
sig100 = var(piEstimate);
```

Figure 19: Q1 part a)



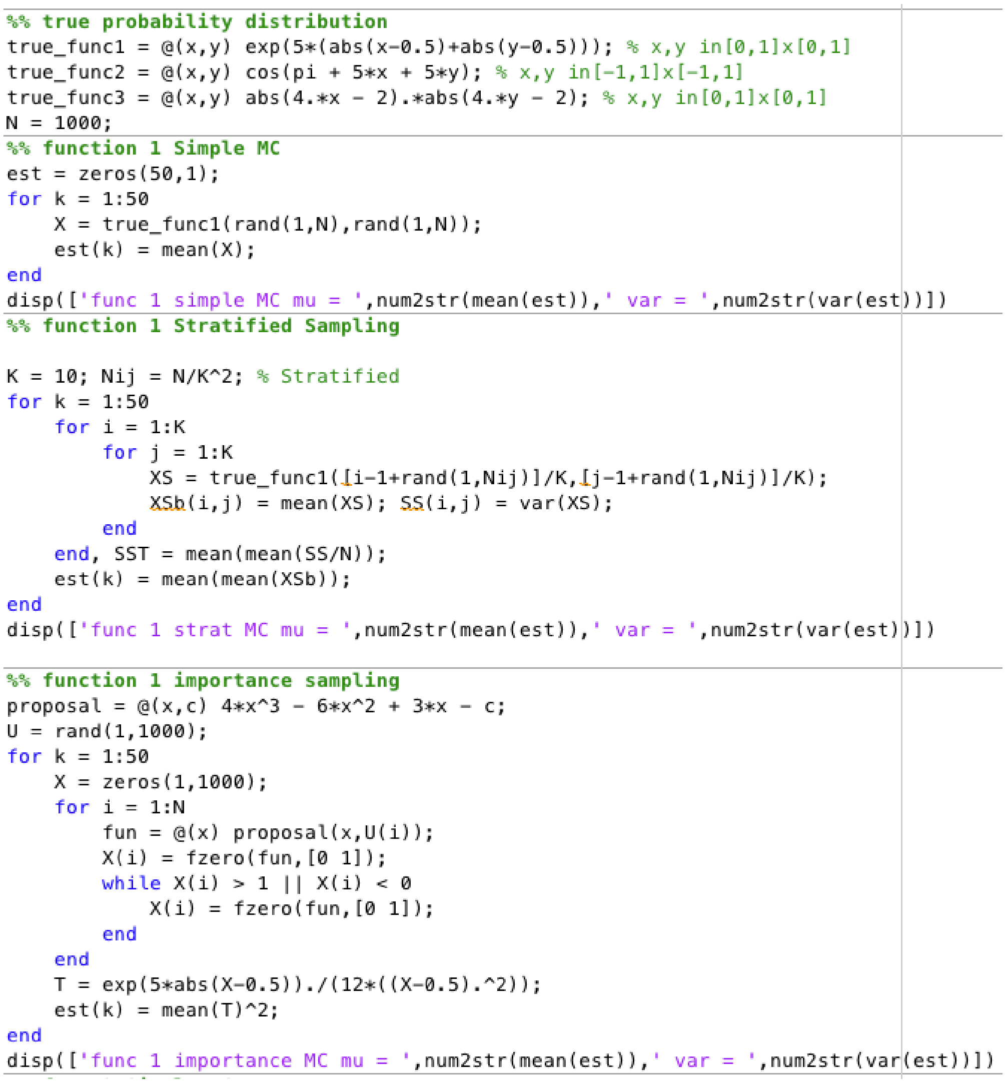
```
%% part b

piEstimate(50) = 0;
n = 1000; % number of Monte Carlo samples
for i=1:50

    x = rand(n, 1); % sample the input random variable x
    y = rand(n, 1); % sample the input random variable y
    isInside = (x.^2 + y.^2 < 1); % is the point inside a unit circle?
    percentage = sum(isInside) / n; % compute statistics: the inside percentage
    piEstimate(i) = percentage * 4;
end
figure
hist(piEstimate)
title('Histogram of MC estimation of \pi, sample size n = 1000')
xlabel('Estimates'),ylabel('times')
mean(piEstimate)
sig1000 = var(piEstimate);

piEstimate(50) = 0;
n = 10000; % number of Monte Carlo samples
for i=1:50

    x = rand(n, 1); % sample the input random variable x
    y = rand(n, 1); % sample the input random variable y
    isInside = (x.^2 + y.^2 < 1); % is the point inside a unit circle?
    percentage = sum(isInside) / n; % compute statistics: the inside percentage
    piEstimate(i) = percentage * 4;
end
figure
hist(piEstimate)
title('Histogram of MC estimation of \pi, sample size n = 10000')
xlabel('Estimates'),ylabel('times')
mean(piEstimate)
sig10000 = var(piEstimate);

figure,loglog([100,1000,10000],[sig100,sig1000,sig10000])
xlabel('number of sample'),ylabel('variance')
```

Figure 20: Q1 part b)



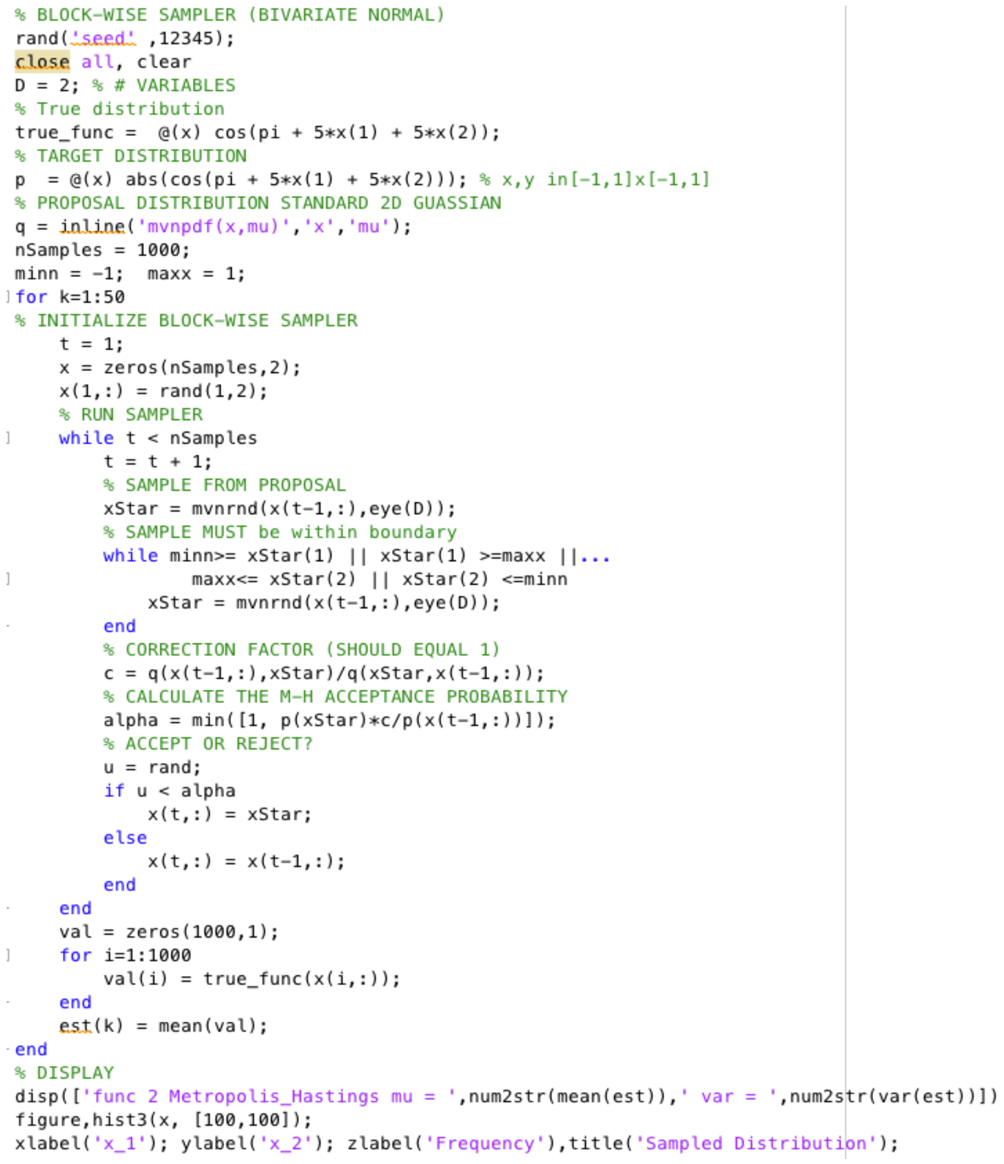
```
%% part c
aEstimate(50) = 0;
n = 100; % number of Monte Carlo samples
sampleArea = 1*1*4;
for i=1:50

    x = rand(n, 1); % sample the input random variable x
    y = rand(n, 1); % sample the input random variable y
    z = 4*rand(n, 1); % sample the input random variable z
    isInside = (z < abs(4 .* x - 2) .* abs(4 .* y - 2));
    aEstimate(i) = sum(isInside)* sampleArea / n;
end
figure
hist(aEstimate)
mean(aEstimate)
var(aEstimate)
```

Figure 21: Q1 part c)



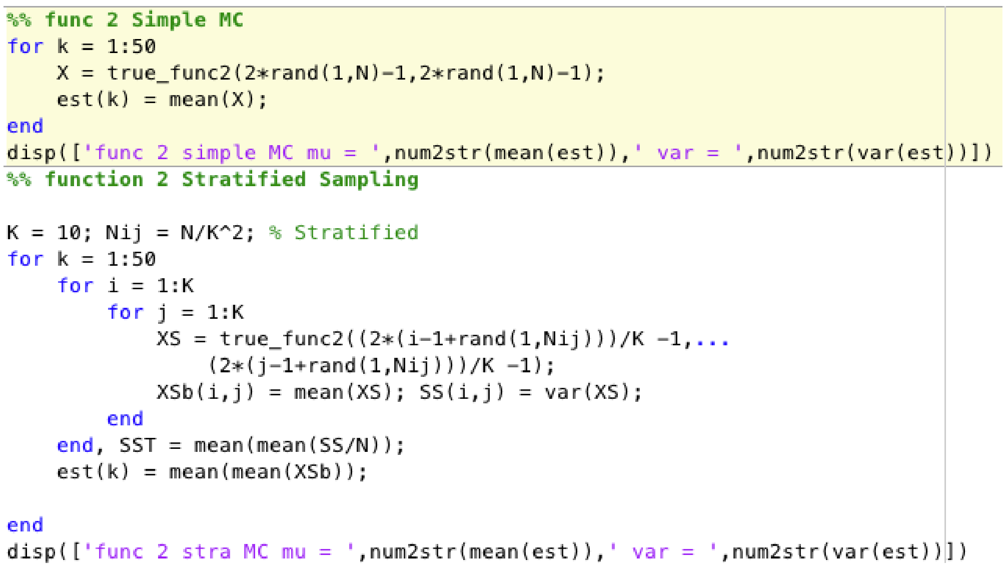
```
%% true probability distribution
true_func1 = @(x,y) exp(5*(abs(x-0.5)+abs(y-0.5))); % x,y in[0,1]x[0,1]
true_func2 = @(x,y) cos(pi + 5*x + 5*y); % x,y in[-1,1]x[-1,1]
true_func3 = @(x,y) abs(4.*x - 2).*abs(4.*y - 2); % x,y in[0,1]x[0,1]
N = 1000;
%% function 1 Simple MC
est = zeros(50,1);
for k = 1:50
    X = true_func1(rand(1,N),rand(1,N));
    est(k) = mean(X);
end
disp(['func 1 simple MC mu = ',num2str(mean(est)),' var = ',num2str(var(est))])
%% function 1 Stratified Sampling

K = 10; Nij = N/K^2; % Stratified
for k = 1:50
    for i = 1:K
        for j = 1:K
            XS = true_func1([i-1+rand(1,Nij)]/K,[j-1+rand(1,Nij)]/K);
            XSb(i,j) = mean(XS); SS(i,j) = var(XS);
        end
    end, SST = mean(mean(SS/N));
    est(k) = mean(mean(XSb));
end
disp(['func 1 strat MC mu = ',num2str(mean(est)),' var = ',num2str(var(est))])

%% function 1 importance sampling
proposal = @(x,c) 4*x^3 - 6*x^2 + 3*x - c;
U = rand(1,1000);
for k = 1:50
    X = zeros(1,1000);
    for i = 1:N
        fun = @(x) proposal(x,U(i));
        X(i) = fzero(fun,[0 1]);
        while X(i) > 1 || X(i) < 0
            X(i) = fzero(fun,[0 1]);
        end
    end
    T = exp(5*abs(X-0.5))./(12*((X-0.5).^2));
    est(k) = mean(T)^2;
end
disp(['func 1 importance MC mu = ',num2str(mean(est)),' var = ',num2str(var(est))])
```

Figure 22: Q2 3 types of MC for function (a)



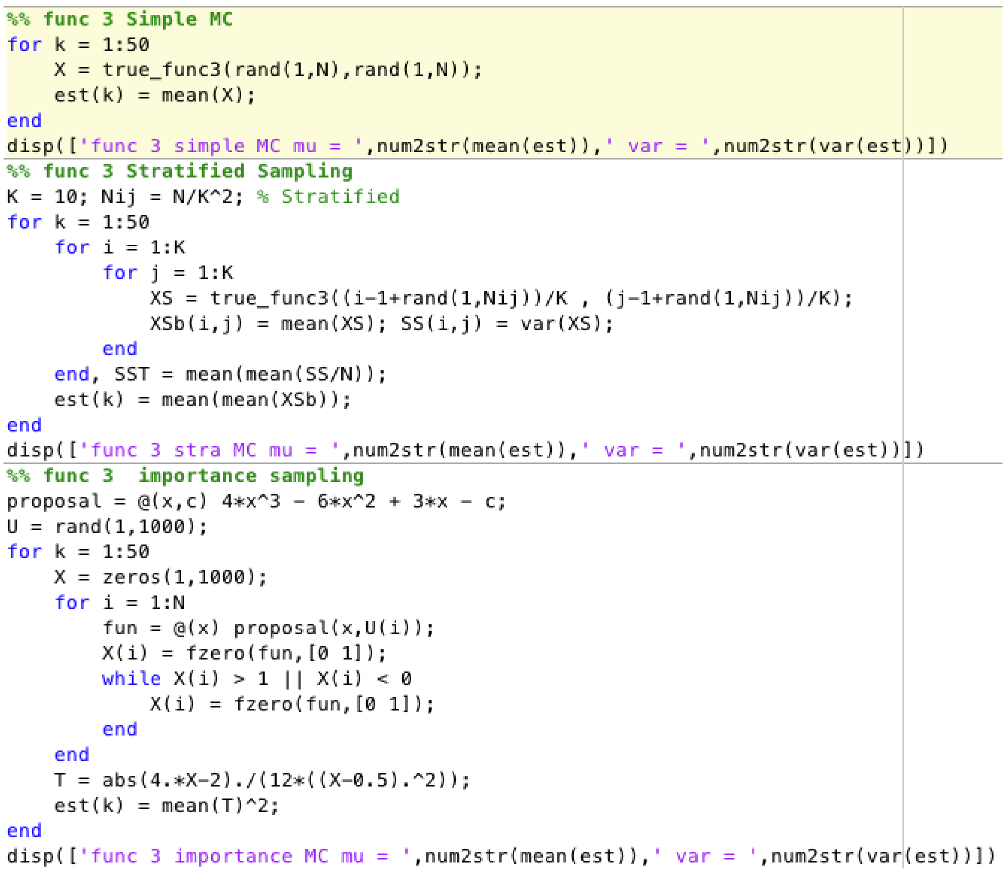
```
% BLOCK-WISE SAMPLER (BIVARIATE NORMAL)
rand('seed' ,12345);
close all, clear
D = 2; % # VARIABLES
% True distribution
true_func =  @(x) cos(pi + 5*x(1) + 5*x(2));
% TARGET DISTRIBUTION
p  = @(x) abs(cos(pi + 5*x(1) + 5*x(2))); % x,y in[-1,1]x[-1,1]
% PROPOSAL DISTRIBUTION STANDARD 2D GUASSIAN
q = inline('mvnpdf(x,mu)','x','mu');
nSamples = 1000;
minn = -1;  maxx = 1;
for k=1:50
% INITIALIZE BLOCK-WISE SAMPLER
    t = 1;
    x = zeros(nSamples,2);
    x(1,:) = rand(1,2);
    % RUN SAMPLER
    while t < nSamples
        t = t + 1;
        % SAMPLE FROM PROPOSAL
        xStar = mvnrnd(x(t-1,:),eye(D));
        % SAMPLE MUST be within boundary
        while minn>= xStar(1) || xStar(1) >=maxx ||...
                maxx<= xStar(2) || xStar(2) <=minn
            xStar = mvnrnd(x(t-1,:),eye(D));
        end
        % CORRECTION FACTOR (SHOULD EQUAL 1)
        c = q(x(t-1,:),xStar)/q(xStar,x(t-1,:));
        % CALCULATE THE M-H ACCEPTANCE PROBABILITY
        alpha = min([1, p(xStar)*c/p(x(t-1,:))]);
        % ACCEPT OR REJECT?
        u = rand;
        if u < alpha
            x(t,:) = xStar;
        else
            x(t,:) = x(t-1,:);
        end
    end
    val = zeros(1000,1);
    for i=1:1000
        val(i) = true_func(x(i,:));
    end
    est(k) = mean(val);
end
% DISPLAY
disp(['func 2 Metropolis_Hastings mu = ',num2str(mean(est)),' var = ',num2str(var(est))])
figure,hist3(x, [100,100]);
xlabel('x_1'); ylabel('x_2'); zlabel('Frequency'),title('Sampled Distribution');
```

Figure 23: Q2 Metropolis-Hastings MC for function (b)



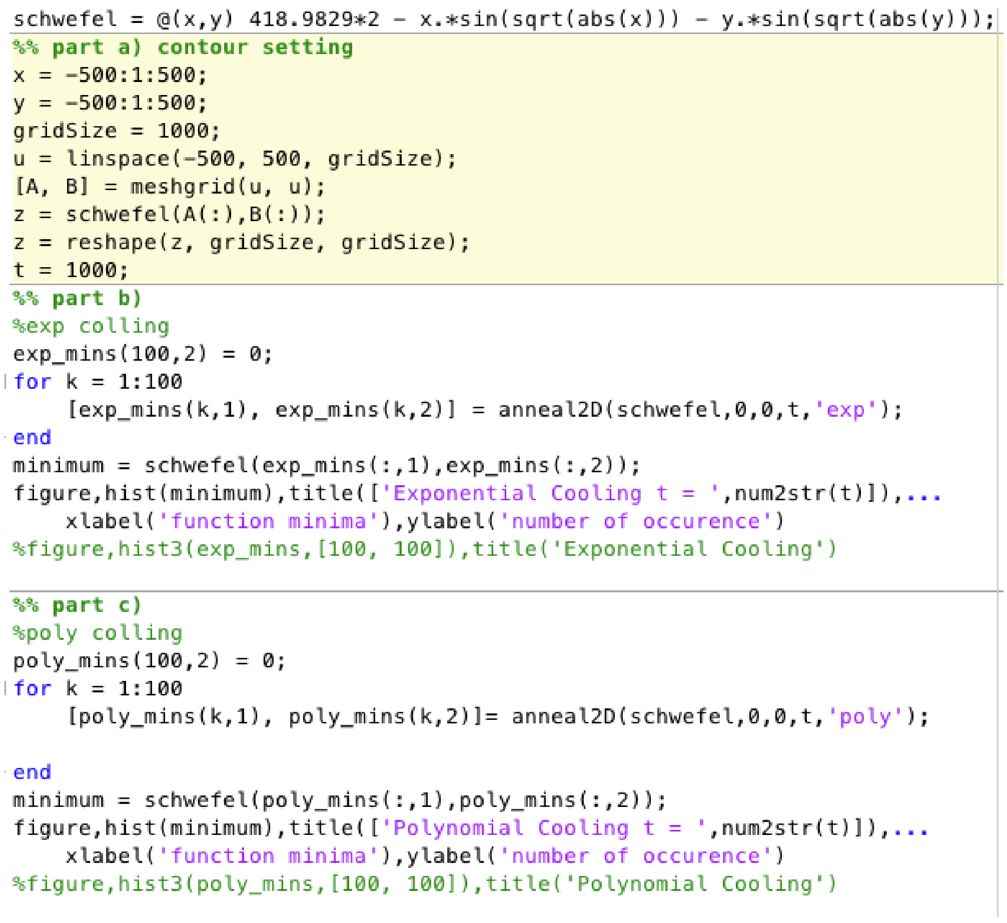
```
%% func 2 Simple MC
for k = 1:50
    X = true_func2(2*rand(1,N)-1,2*rand(1,N)-1);
    est(k) = mean(X);
end
disp(['func 2 simple MC mu = ',num2str(mean(est)),' var = ',num2str(var(est))])
%% function 2 Stratified Sampling

K = 10; Nij = N/K^2; % Stratified
for k = 1:50
    for i = 1:K
        for j = 1:K
            XS = true_func2((2*(i-1+rand(1,Nij)))/K -1,...
                (2*(j-1+rand(1,Nij)))/K -1);
            XSb(i,j) = mean(XS); SS(i,j) = var(XS);
        end
    end, SST = mean(mean(SS/N));
    est(k) = mean(mean(XSb));

end
disp(['func 2 stra MC mu = ',num2str(mean(est)),' var = ',num2str(var(est))])
```

Figure 24: Q2 2 types of MC for function (b)



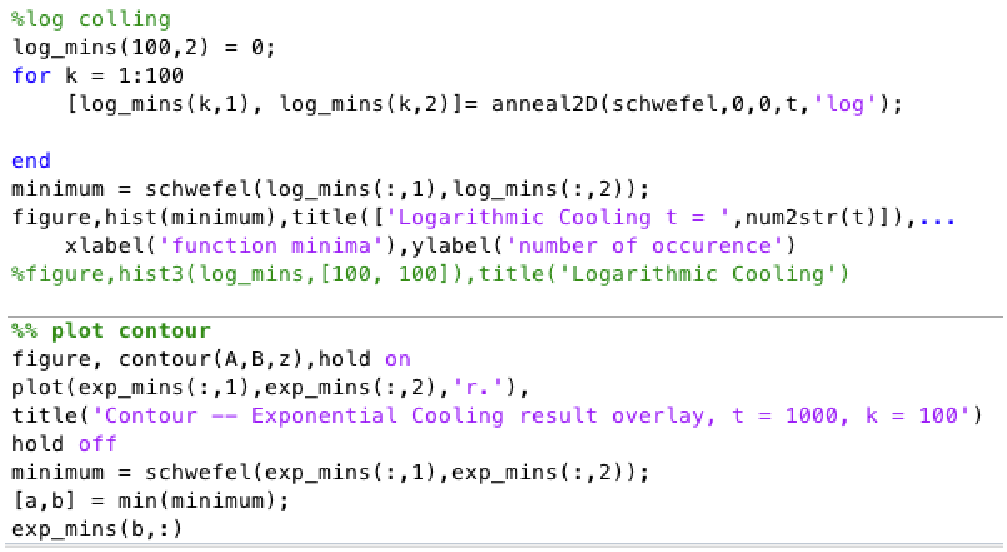
```
%% func 3 Simple MC
for k = 1:50
    X = true_func3(rand(1,N),rand(1,N));
    est(k) = mean(X);
end
disp(['func 3 simple MC mu = ',num2str(mean(est)),' var = ',num2str(var(est))])
%% func 3 Stratified Sampling
K = 10; Nij = N/K^2; % Stratified
for k = 1:50
    for i = 1:K
        for j = 1:K
            XS = true_func3((i-1+rand(1,Nij))/K , (j-1+rand(1,Nij))/K);
            XSb(i,j) = mean(XS); SS(i,j) = var(XS);
        end
    end, SST = mean(mean(SS/N));
    est(k) = mean(mean(XSb));
end
disp(['func 3 stra MC mu = ',num2str(mean(est)),' var = ',num2str(var(est))])
%% func 3  importance sampling
proposal = @(x,c) 4*x^3 - 6*x^2 + 3*x - c;
U = rand(1,1000);
for k = 1:50
    X = zeros(1,1000);
    for i = 1:N
        fun = @(x) proposal(x,U(i));
        X(i) = fzero(fun,[0 1]);
        while X(i) > 1 || X(i) < 0
            X(i) = fzero(fun,[0 1]);
        end
    end
    T = abs(4.*X-2)./(12*((X-0.5).^2));
    est(k) = mean(T)^2;
end
disp(['func 3 importance MC mu = ',num2str(mean(est)),' var = ',num2str(var(est))])
```

Figure 25: Q2 3 types of MC for function (c)



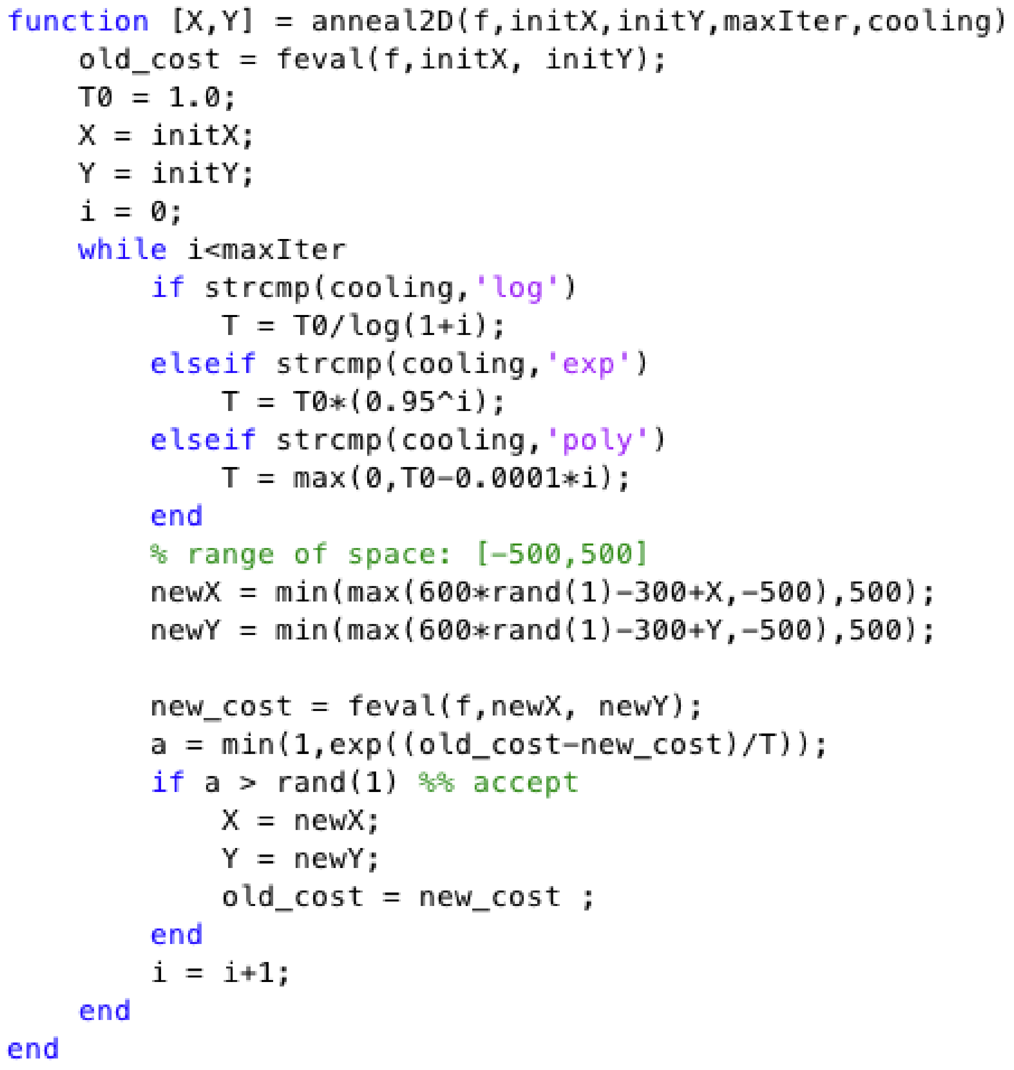
```
schwefel = @(x,y) 418.9829*2 - x.*sin(sqrt(abs(x))) - y.*sin(sqrt(abs(y)));
%% part a) contour setting
x = -500:1:500;
y = -500:1:500;
gridSize = 1000;
u = linspace(-500, 500, gridSize);
[A, B] = meshgrid(u, u);
z = schwefel(A(:),B(:));
z = reshape(z, gridSize, gridSize);
t = 1000;
%% part b)
%exp colling
exp_mins(100,2) = 0;
for k = 1:100
    [exp_mins(k,1), exp_mins(k,2)] = anneal2D(schwefel,0,0,t,'exp');
end
minimum = schwefel(exp_mins(:,1),exp_mins(:,2));
figure,hist(minimum),title(['Exponential Cooling t = ',num2str(t)]),...
    xlabel('function minima'),ylabel('number of occurence')
%figure,hist3(exp_mins,[100, 100]),title('Exponential Cooling')

%% part c)
%poly colling
poly_mins(100,2) = 0;
for k = 1:100
    [poly_mins(k,1), poly_mins(k,2)]= anneal2D(schwefel,0,0,t,'poly');

end
minimum = schwefel(poly_mins(:,1),poly_mins(:,2));
figure,hist(minimum),title(['Polynomial Cooling t = ',num2str(t)]),...
    xlabel('function minima'),ylabel('number of occurence')
%figure,hist3(poly_mins,[100, 100]),title('Polynomial Cooling')
```

Figure 26: Q3 implementation of Exponential and Polynomial cooling



```
%log colling
log_mins(100,2) = 0;
for k = 1:100
    [log_mins(k,1), log_mins(k,2)]= anneal2D(schwefel,0,0,t,'log');

end
minimum = schwefel(log_mins(:,1),log_mins(:,2));
figure,hist(minimum),title(['Logarithmic Cooling t = ',num2str(t)]),...
    xlabel('function minima'),ylabel('number of occurence')
%figure,hist3(log_mins,[100, 100]),title('Logarithmic Cooling')

%% plot contour
figure, contour(A,B,z),hold on
plot(exp_mins(:,1),exp_mins(:,2),'r.'),
title('Contour -- Exponential Cooling result overlay, t = 1000, k = 100')
hold off
minimum = schwefel(exp_mins(:,1),exp_mins(:,2));
[a,b] = min(minimum);
exp_mins(b,:)
```

Figure 27: Q3 implementation of Logarithmic cooling and contour plot



```
function [X,Y] = anneal2D(f,initX,initY,maxIter,cooling)
    old_cost = feval(f,initX, initY);
    T0 = 1.0;
    X = initX;
    Y = initY;
    i = 0;
    while i<maxIter
        if strcmp(cooling,'log')
            T = T0/log(1+i);
        elseif strcmp(cooling,'exp')
            T = T0*(0.95^i);
        elseif strcmp(cooling,'poly')
            T = max(0,T0-0.0001*i);
        end
        % range of space: [-500,500]
        newX = min(max(600*rand(1)-300+X,-500),500);
        newY = min(max(600*rand(1)-300+Y,-500),500);

        new_cost = feval(f,newX, newY);
        a = min(1,exp((old_cost-new_cost)/T));
        if a > rand(1) %% accept
            X = newX;
            Y = newY;
            old_cost = new_cost ;
        end
        i = i+1;
    end
end
```

Figure 28: Simulated Annealing implementation