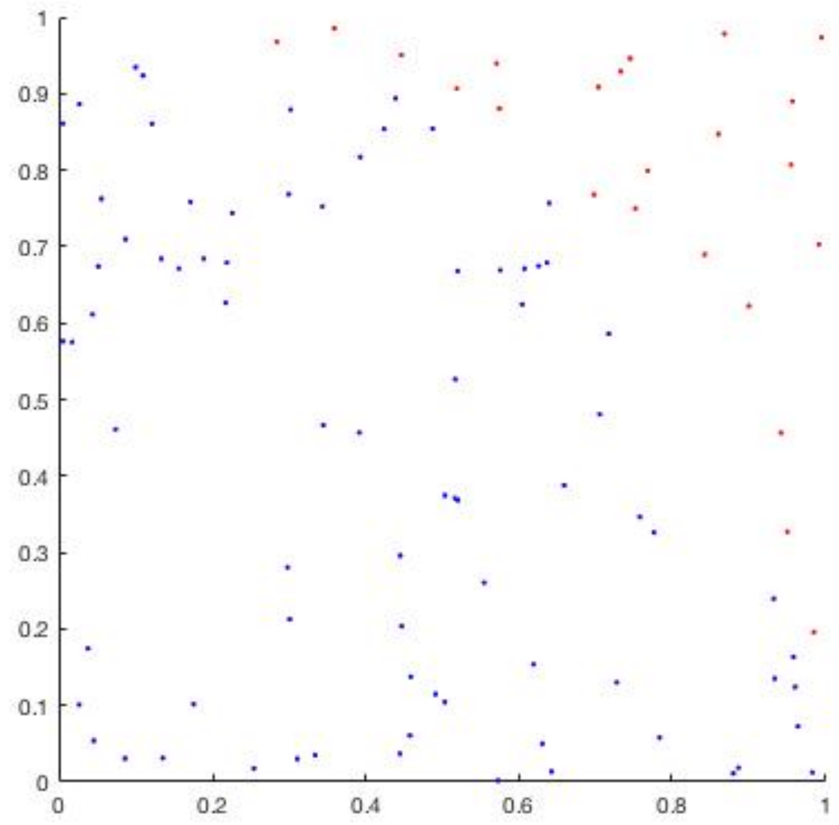
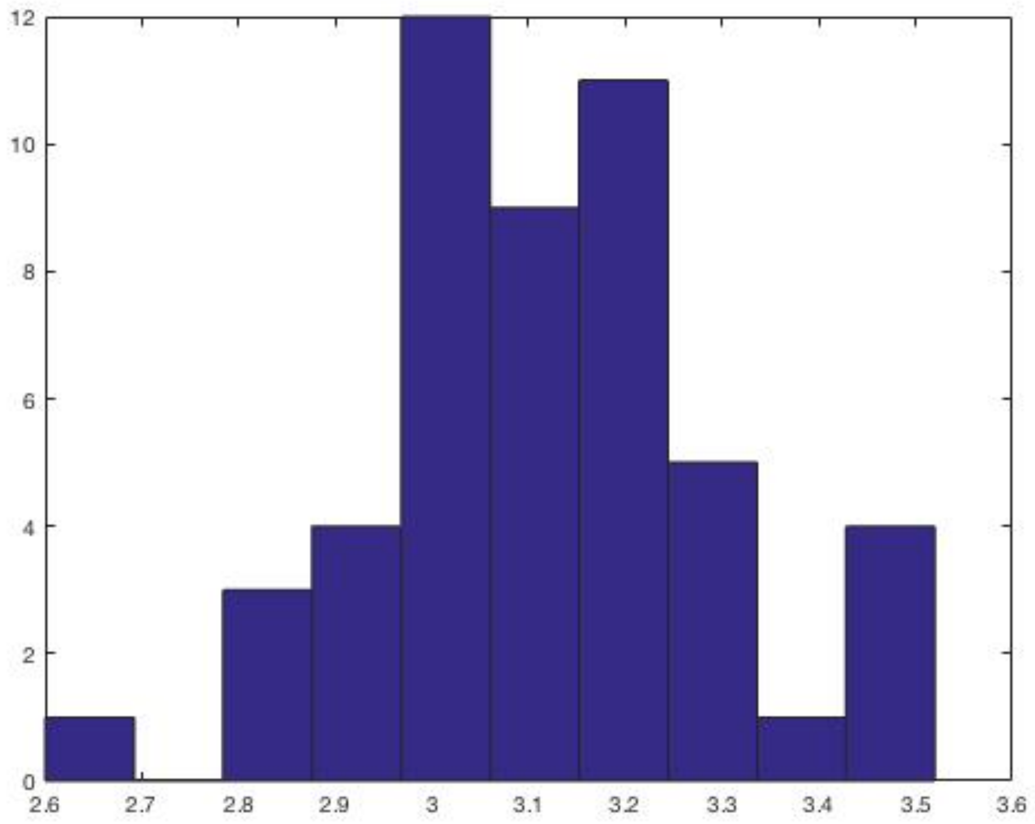
EE 511 final project
Liu, Yang USC ID:5847572002
Problem 1
I used matlab to solve this problem.
i)When k=50 and t=100, a figure of sample points in red and blue during one iteration is shown as following.



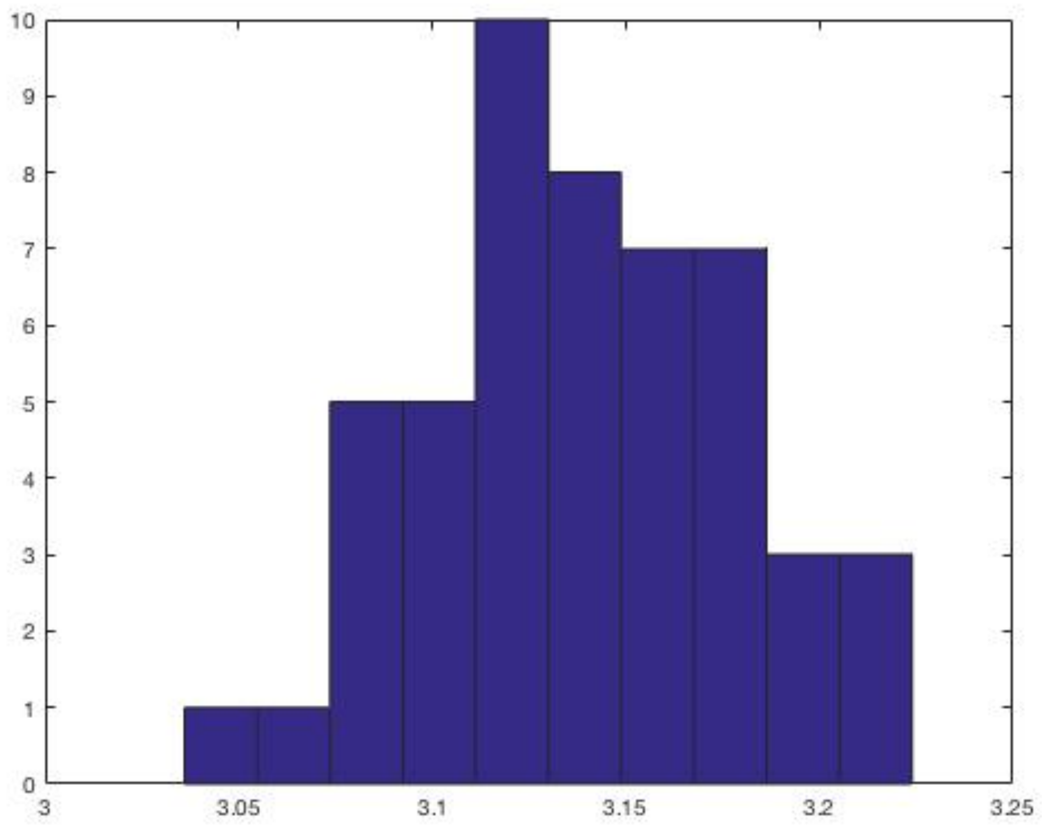
And the histogram of the 50 pi estimates is shown as following.



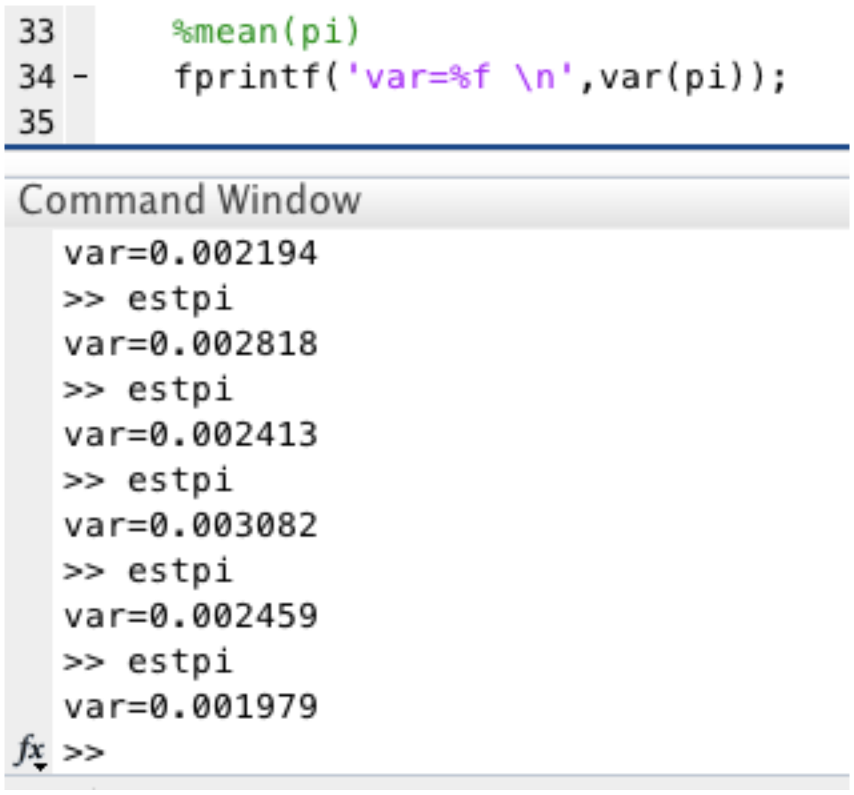
We can see that the outcome is not so good when n is small (100). The smallest estimate of pi is about 2.6 while the largest one is about 3.5, which deviate the real value 3.14

ii)

If I set n to be 1000 with the same value of k=50,the histogram is shown as following.



And the variance when n= 1000 is shown as following.

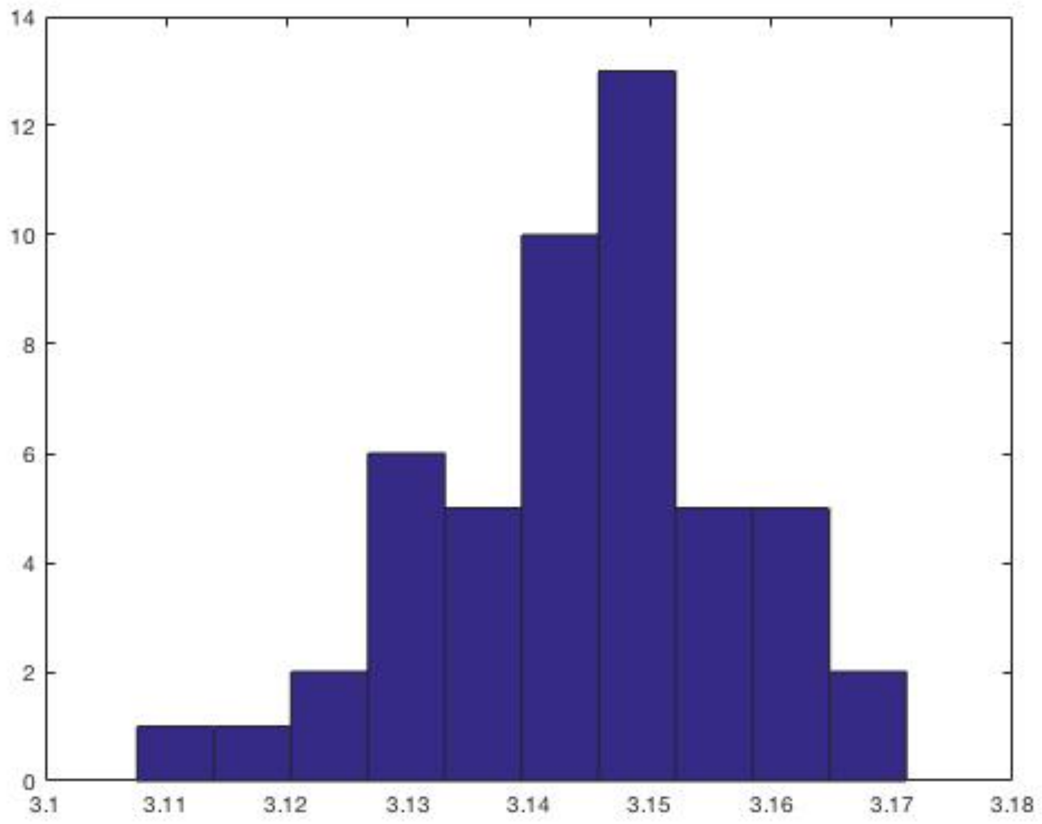


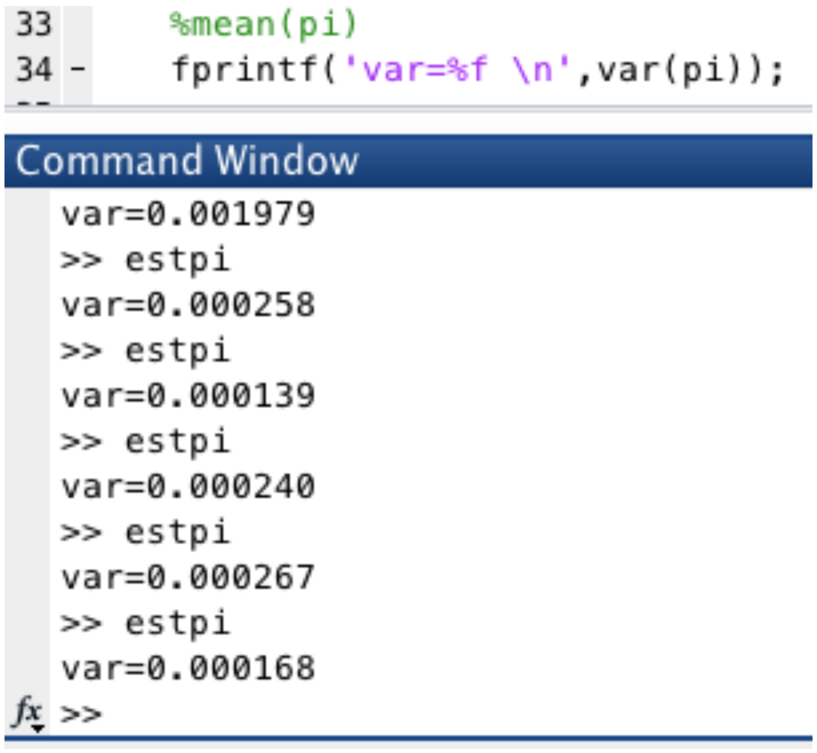
```
    %mean(pi)
    fprintf('var=%f \n',var(pi));

```

Command Window

```
var=0.002194
>> estpi
var=0.002818
>> estpi
var=0.002413
>> estpi
var=0.003082
>> estpi
var=0.002459
>> estpi
var=0.001979
>>
```

When I set n to be 10000, the histogram of pi and variance of them is shown as following.





```
%mean(pi)
fprintf('var=%f \n',var(pi));
```

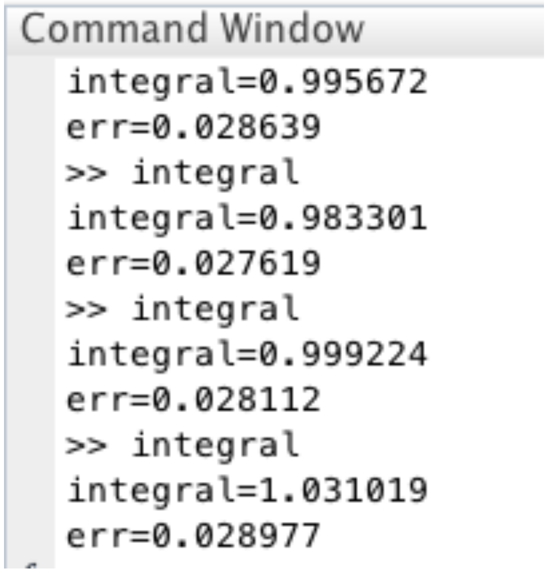
Command Window

```
var=0.001979
>> estpi
var=0.000258
>> estpi
var=0.000139
>> estpi
var=0.000240
>> estpi
var=0.000267
>> estpi
var=0.000168
>>
```

We can conclude that when monte carlo sample size become larger(from 100 to 1000 and 10000), the estimate variance become smaller, which means that we are more likely to find a value of pi estimate closer to the real value 3.14.

iii)

The result of the integral and error estimate for the function is shown as following.



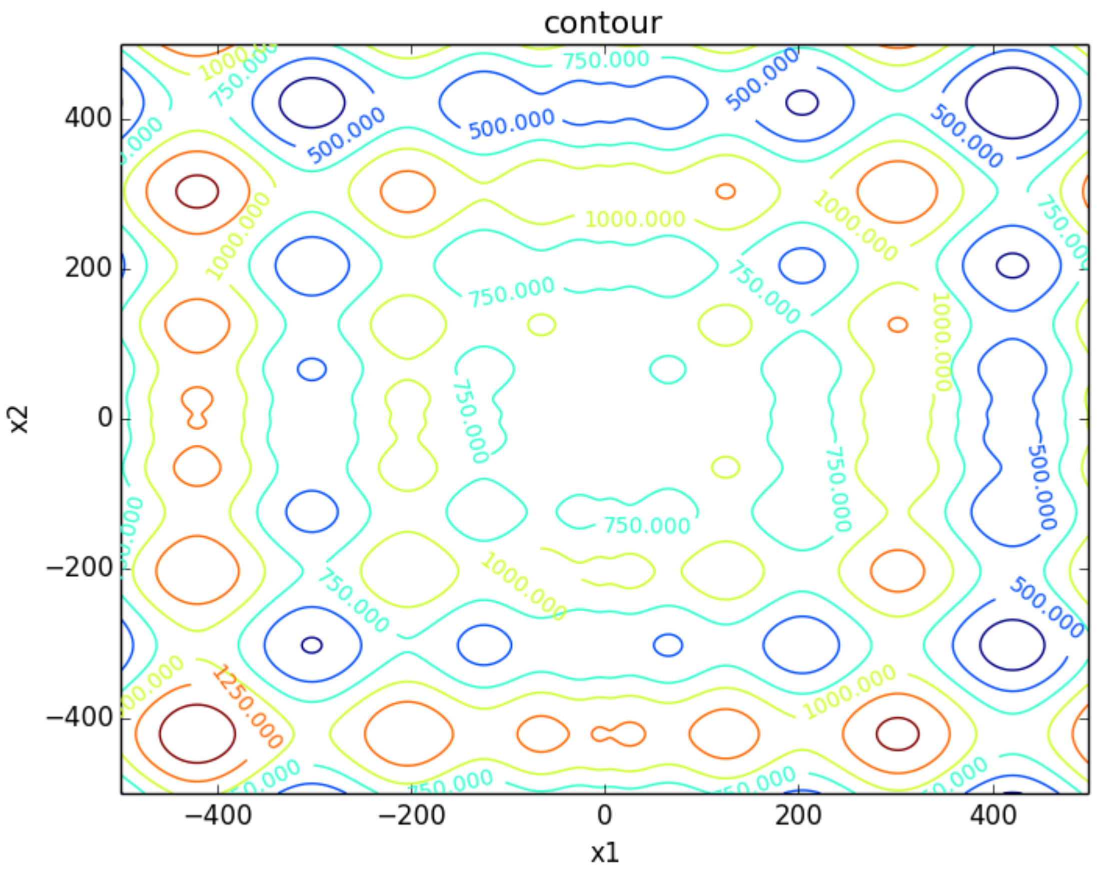
```
integral=0.995672
err=0.028639
>> integral
integral=0.983301
err=0.027619
>> integral
integral=0.999224
err=0.028112
>> integral
integral=1.031019
err=0.028977
```

problem 2

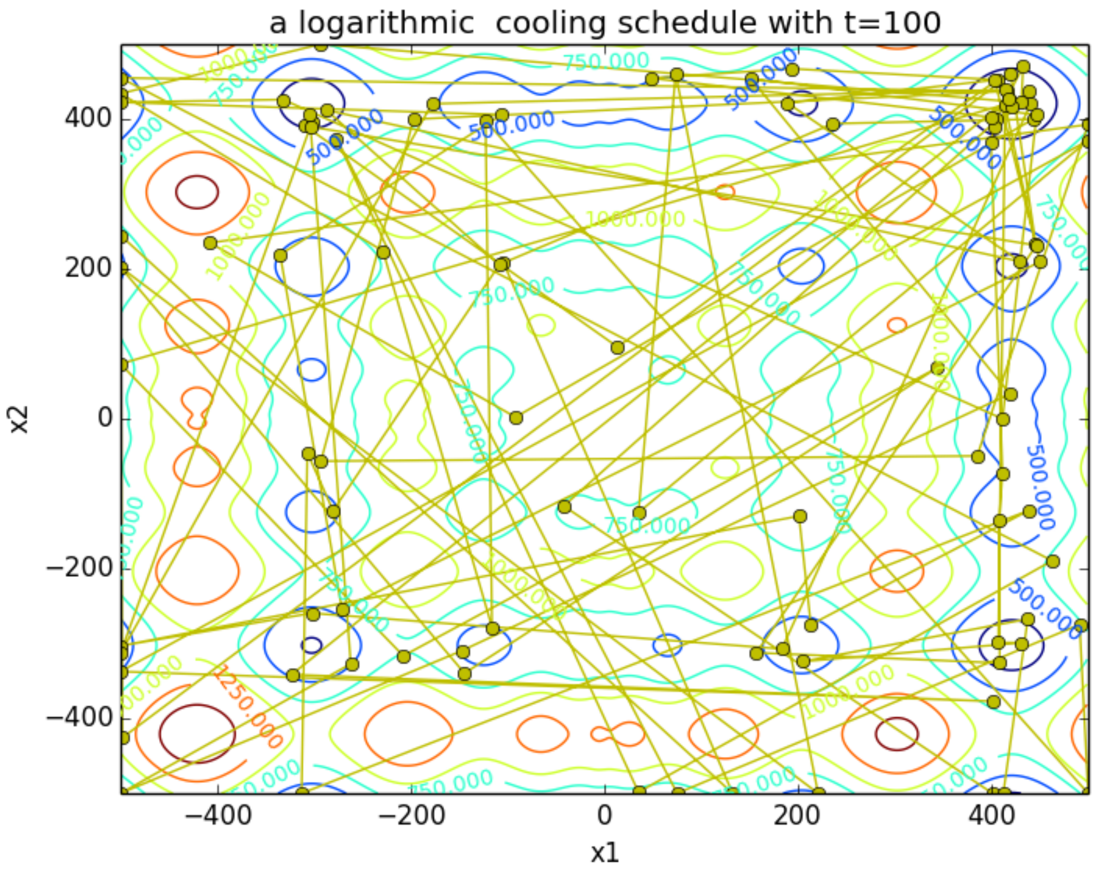
Problem 3

For convenience, I used python for the third problem.

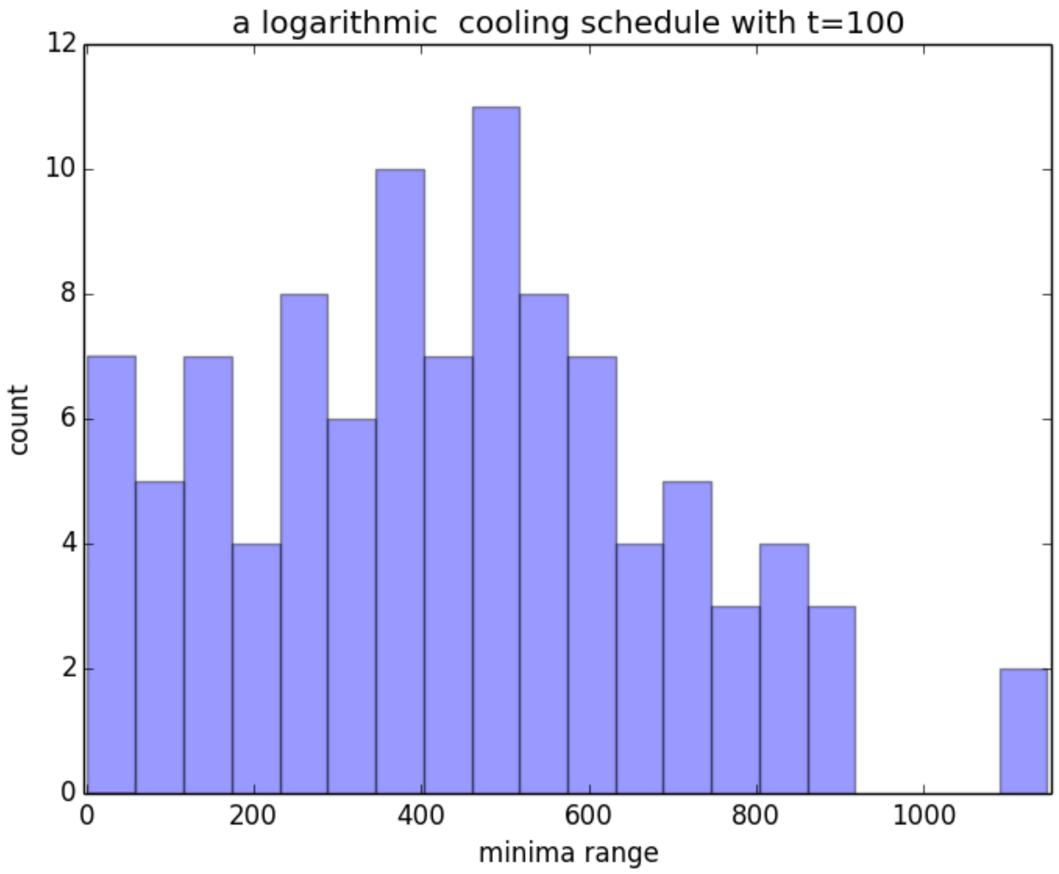
i) The contour plot of the Scwefel function is shown below.



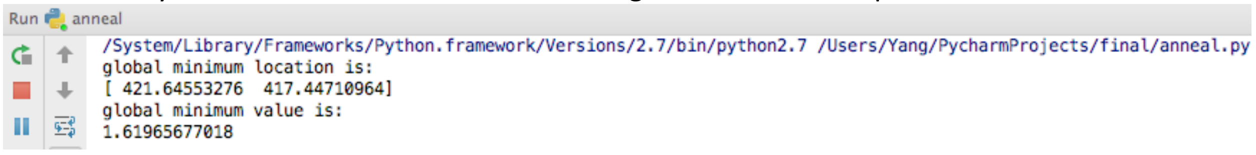
ii) I implemented a simulated annealing procedure to find the global minimum value of the Scwefel function. All the minima with t = 100 are in yellow. The schedule I used is a logarithmic cooling schedule.



The histogram of the minima is also shown in the following figure.



And the global minimum is printed as following. We can see that the minimum is about 1.62 which is very close to 0. And the location of this global minimum is printed.



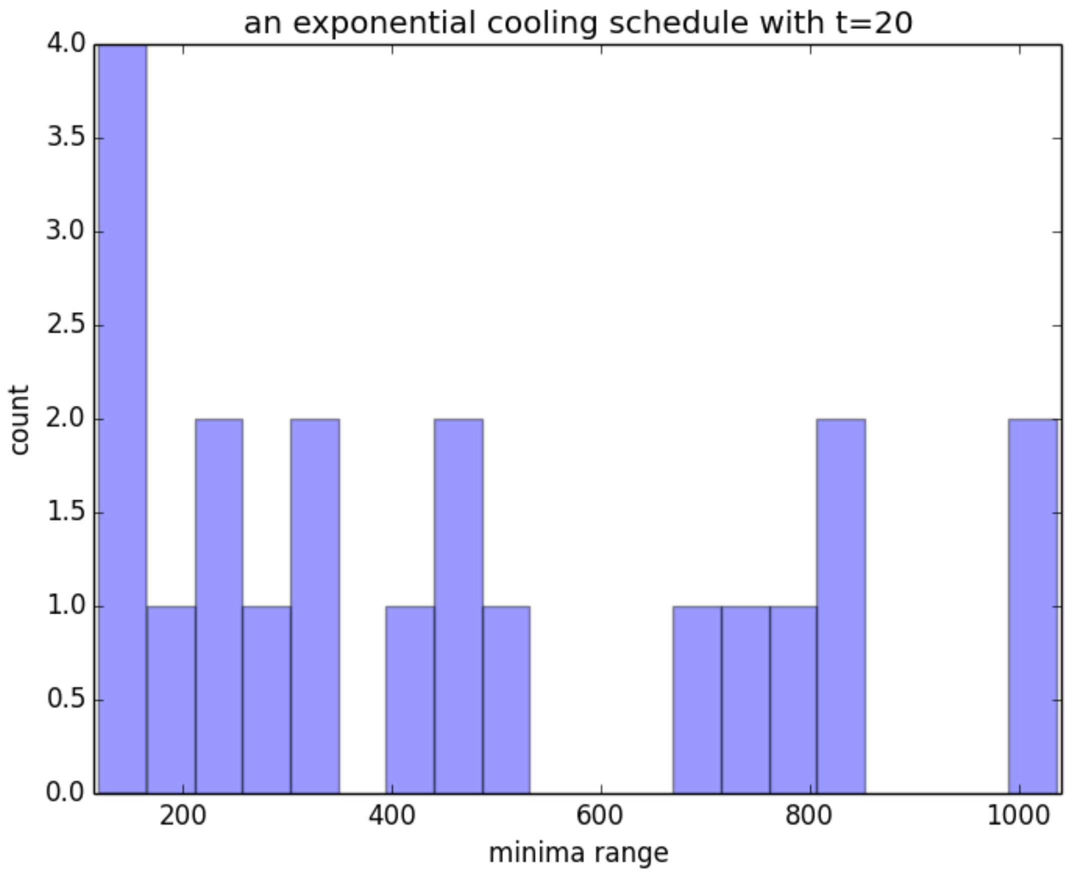
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 421.64553276  417.44710964]
global minimum value is:
1.61965677018
```

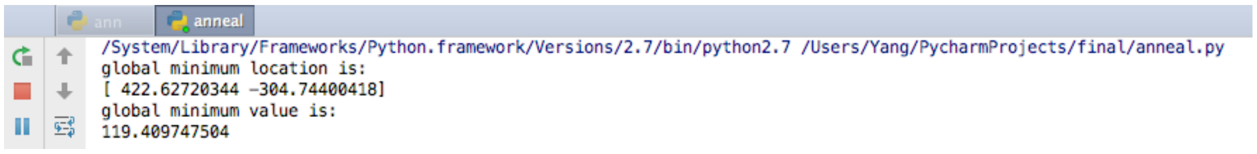
iii)

The outcome of all simulated annealing procedure with different cooling schedule and different t values are shown as following. From these figure I can conclude that with the increase of t, the percentage of smaller minima is increasing, which means that with the increase of the iteration times, we are more likely to find the global minimum.

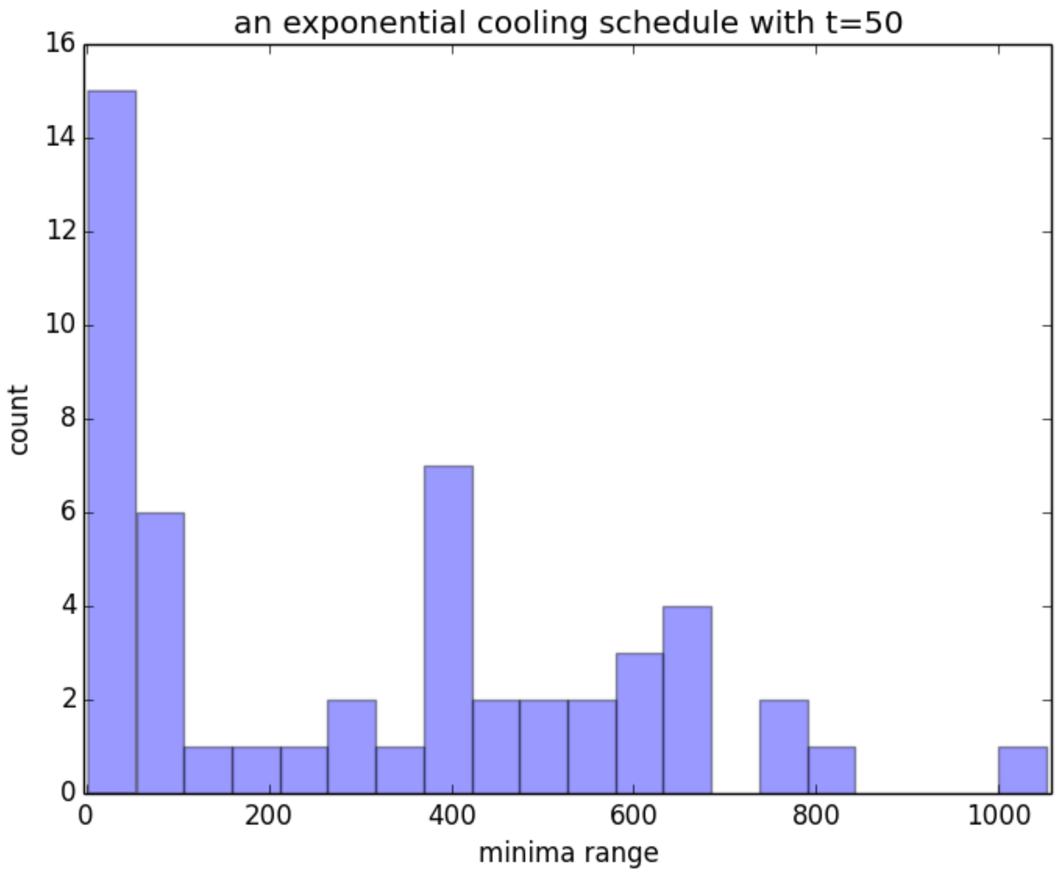
Another finding is that in my algorithm, both the exponential cooling schedule and the logarithmic cooling schedule are very powerful, they show good stability when t become bigger, the global minimum they found are very close to 0 and the percentage of the smaller minima are obviously bigger than others.

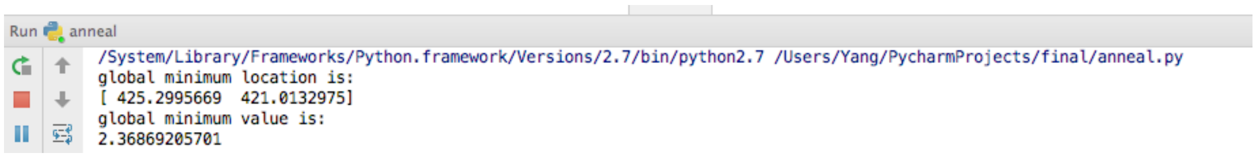
an exponential cooling schedule



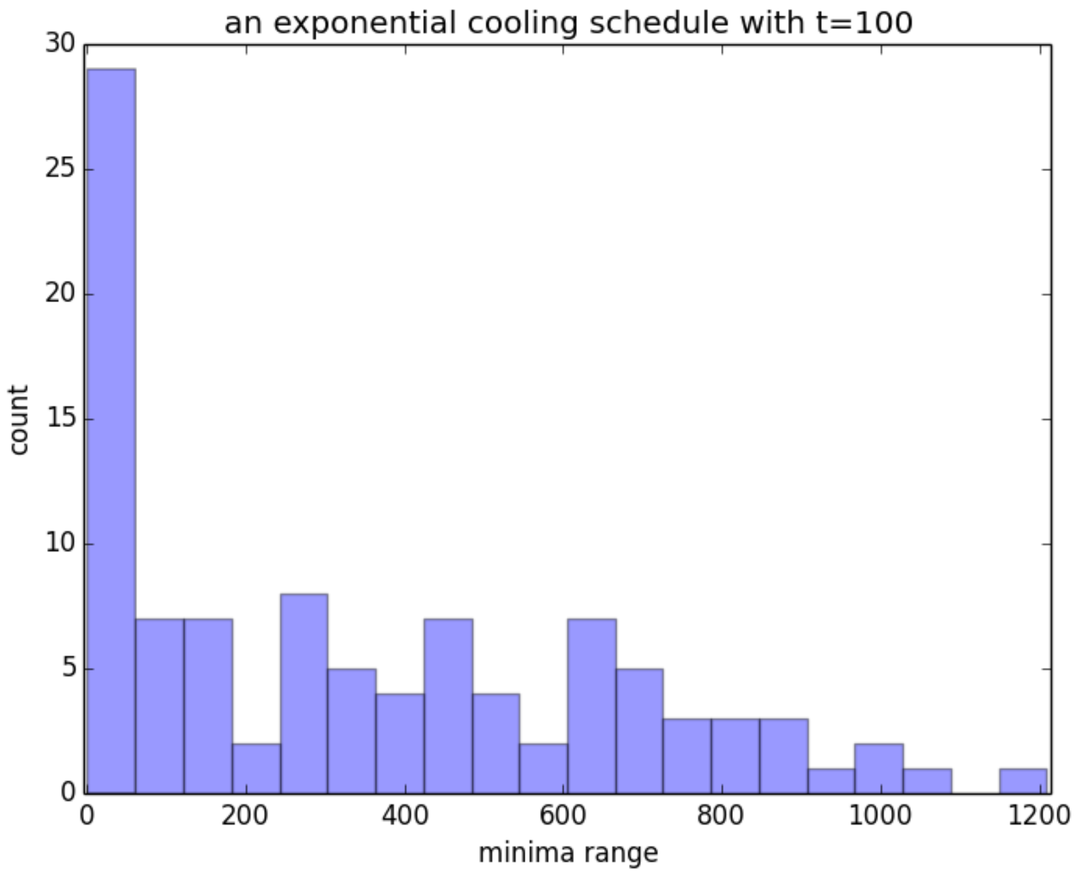


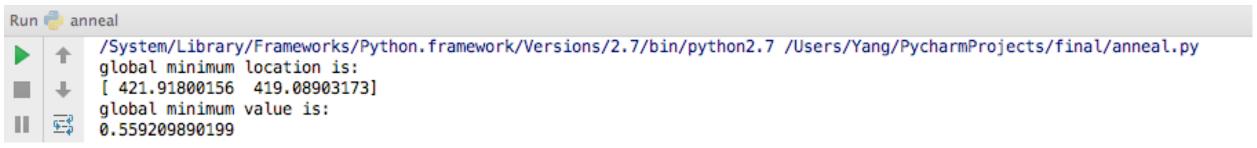
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 422.62720344 -304.74400418]
global minimum value is:
119.409747504
```



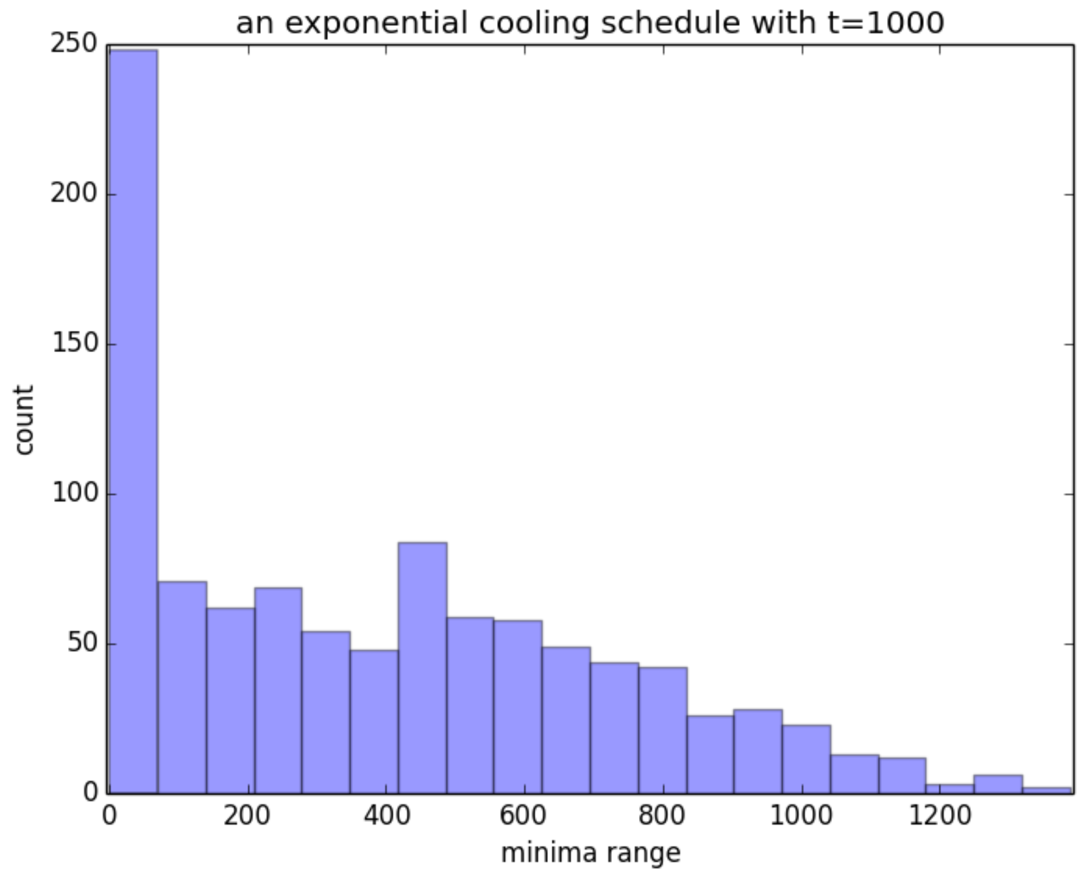


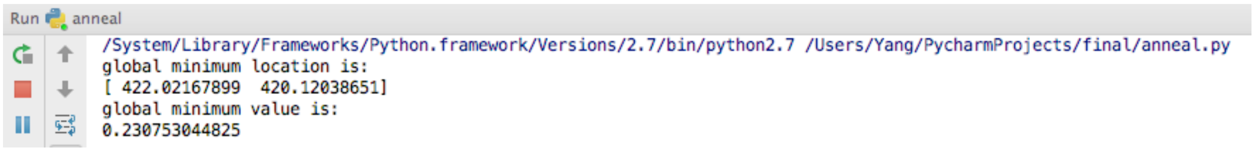
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 425.2995669  421.0132975]
global minimum value is:
2.36869205701
```





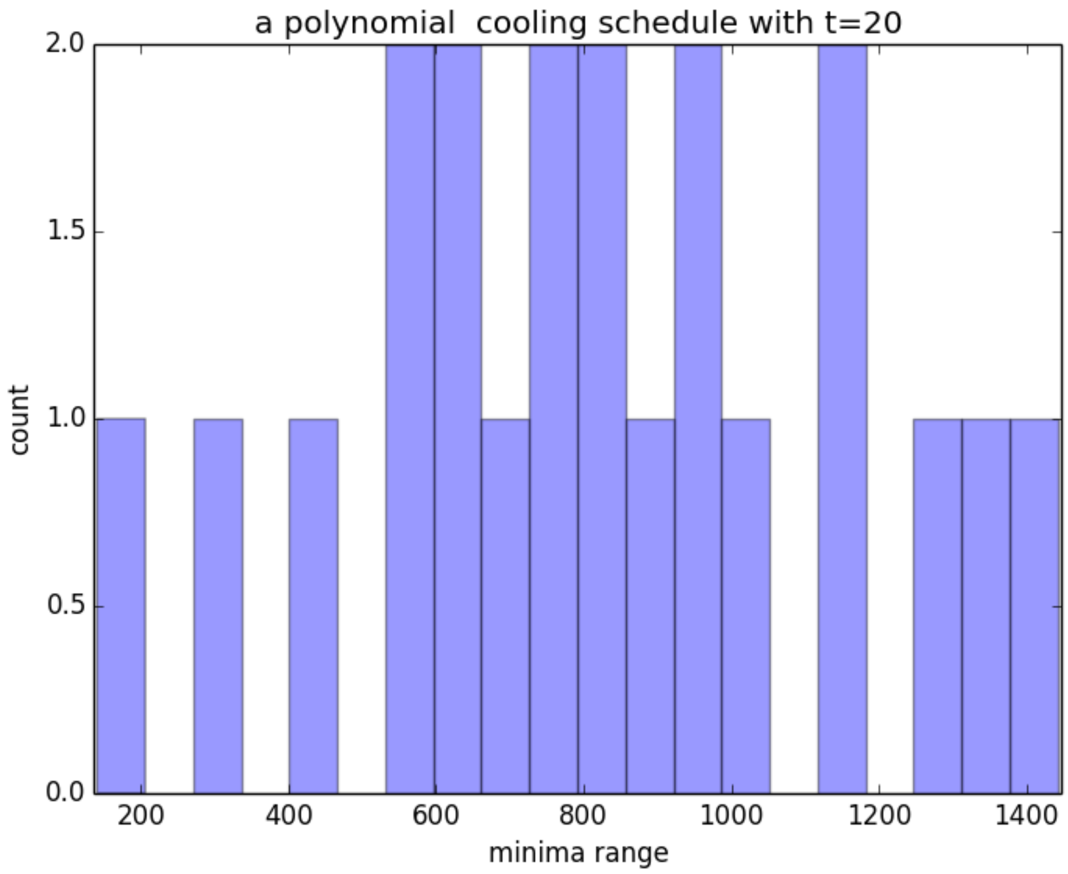
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 421.91800156  419.08903173]
global minimum value is:
0.559209890199
```

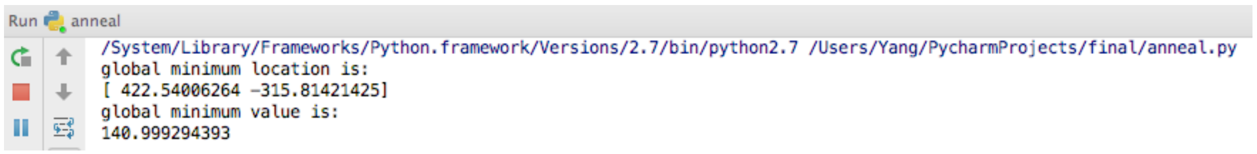




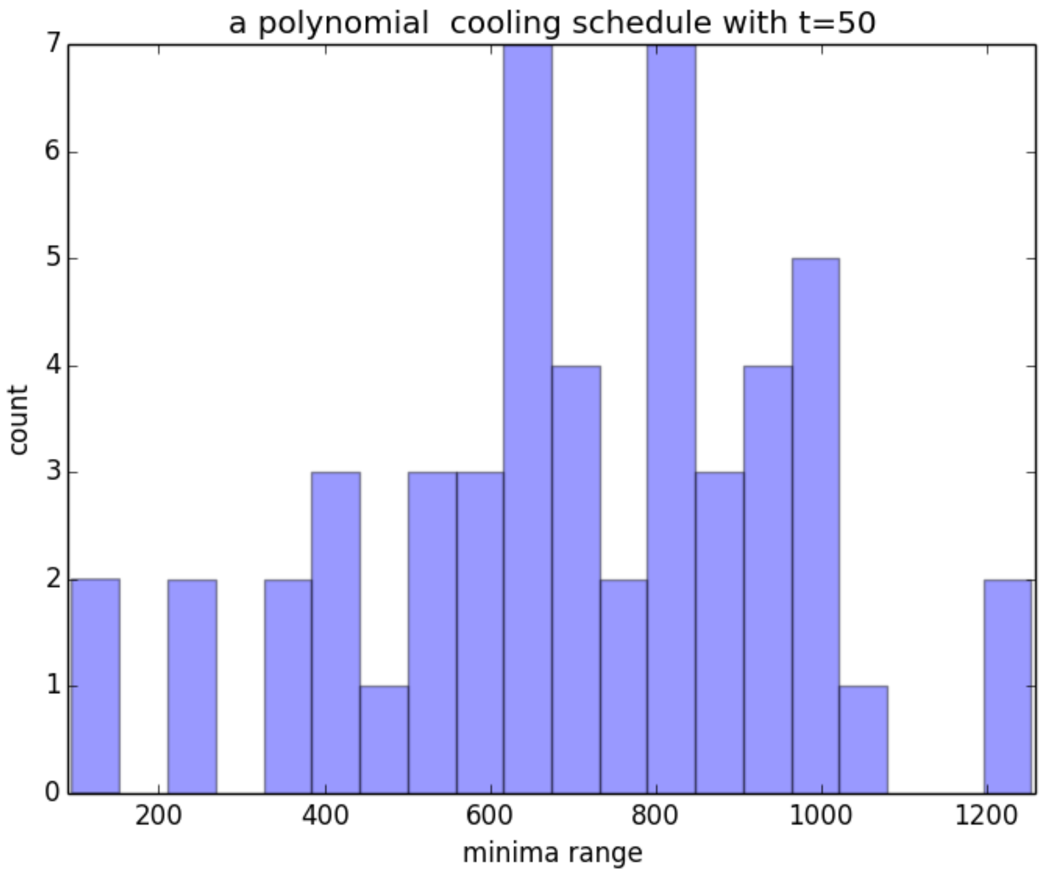
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 422.02167899  420.12038651]
global minimum value is:
0.230753044825
```

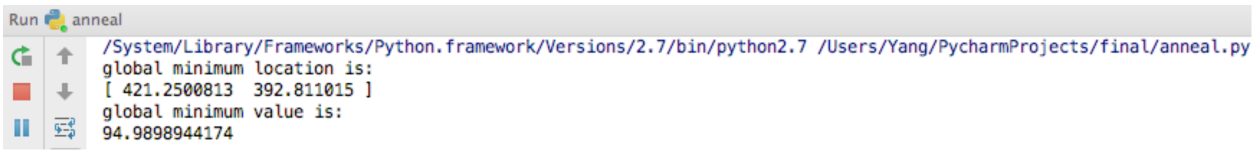
a polynomial cooling schedule



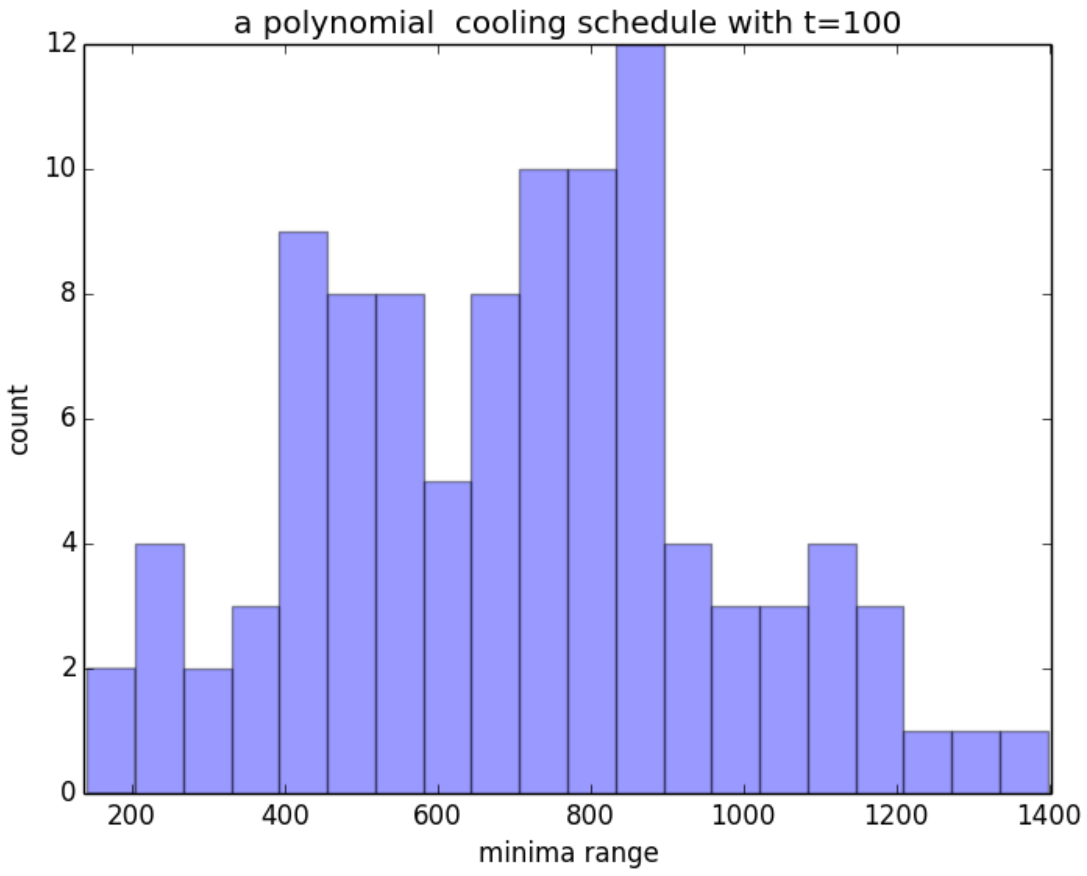


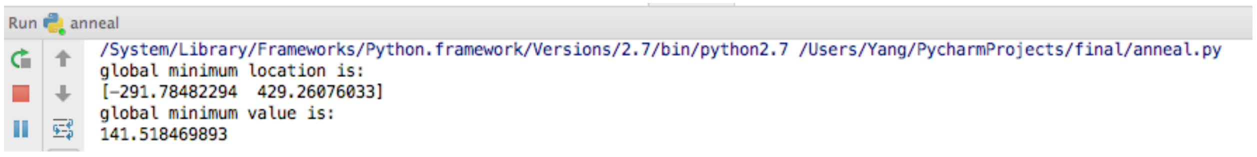
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 422.54006264 -315.81421425]
global minimum value is:
140.999294393
```



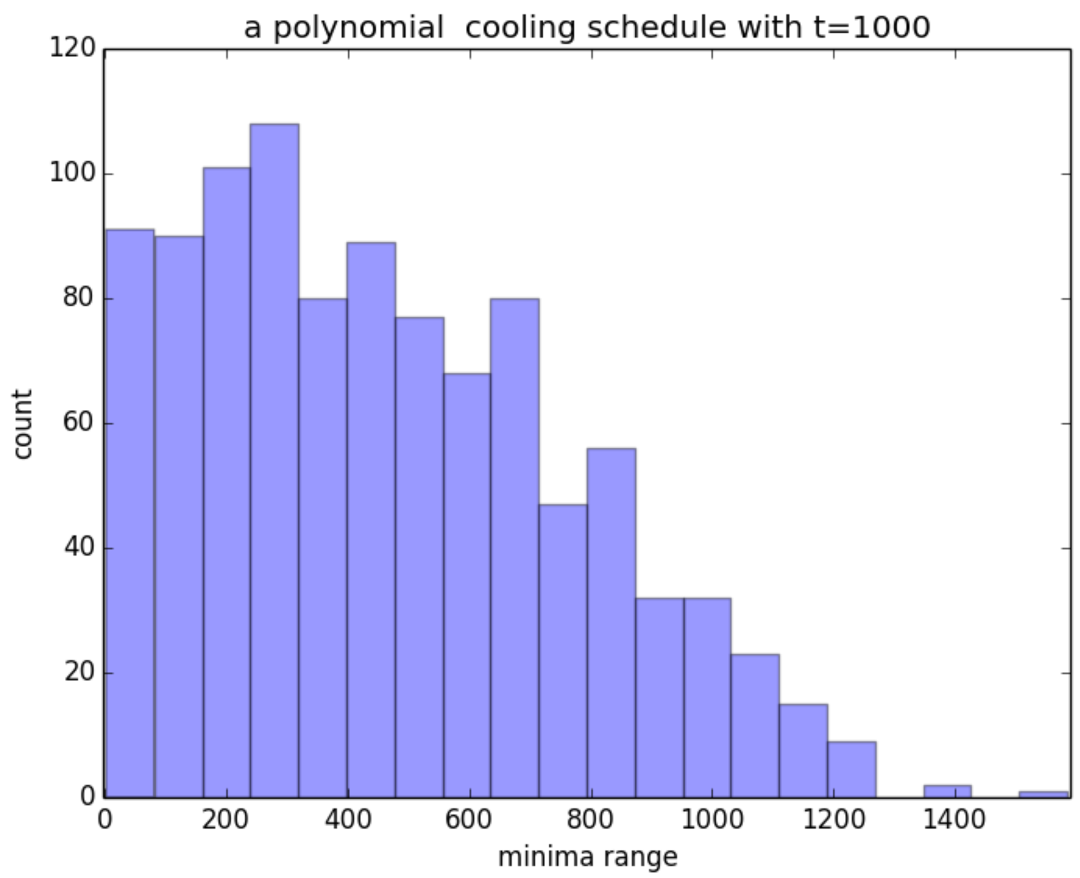


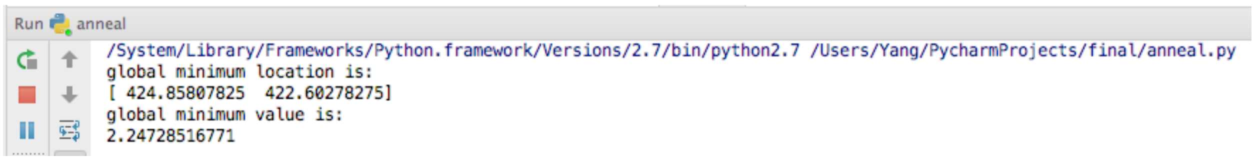
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 421.2500813  392.811015 ]
global minimum value is:
94.9898944174
```





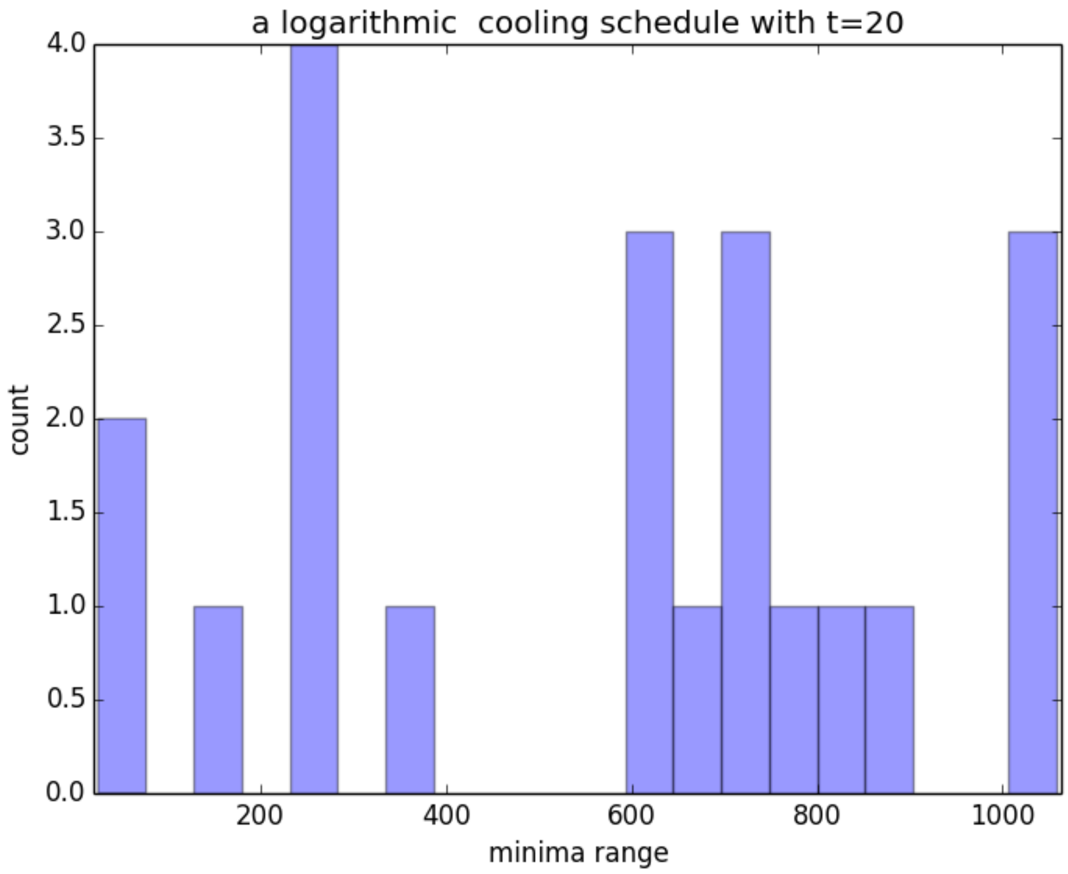
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[-291.78482294  429.26076033]
global minimum value is:
141.518469893
```

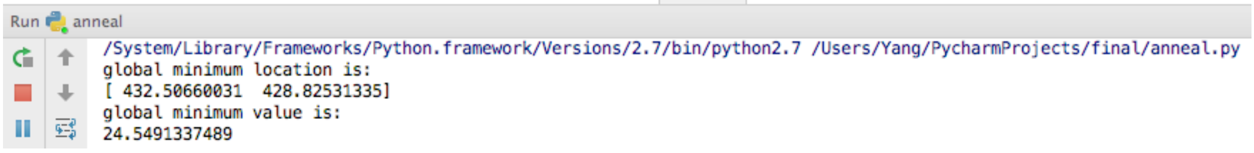




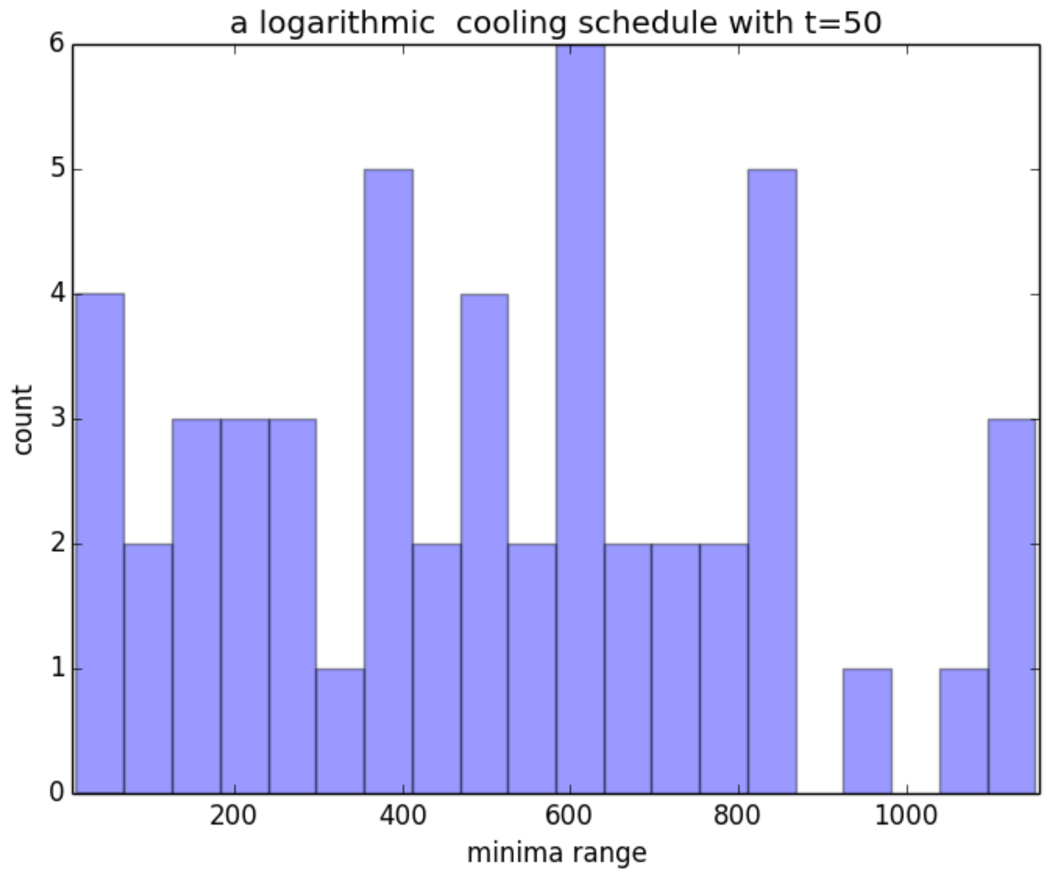
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 424.85807825  422.60278275]
global minimum value is:
2.24728516771
```

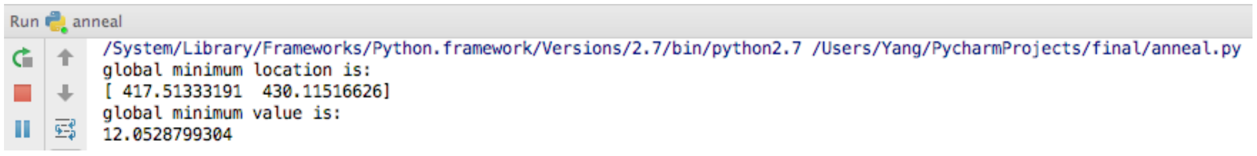
a logarithmic cooling schedule



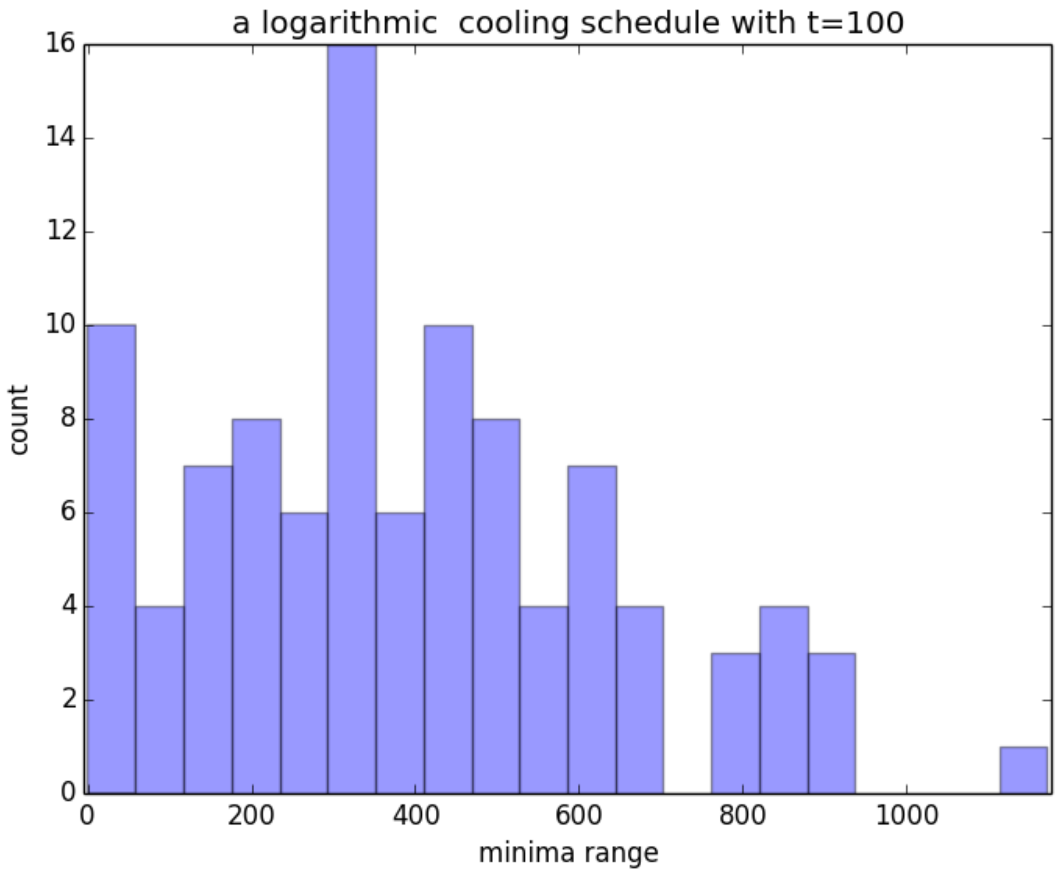


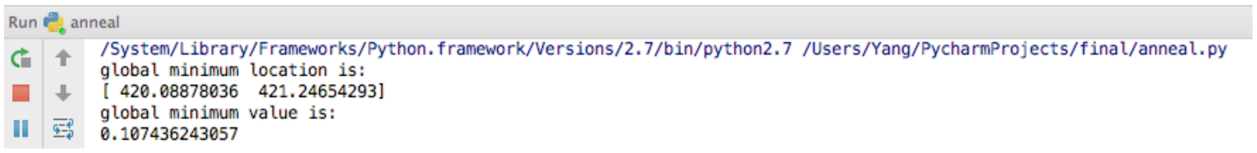
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 432.50660031  428.82531335]
global minimum value is:
24.5491337489
```



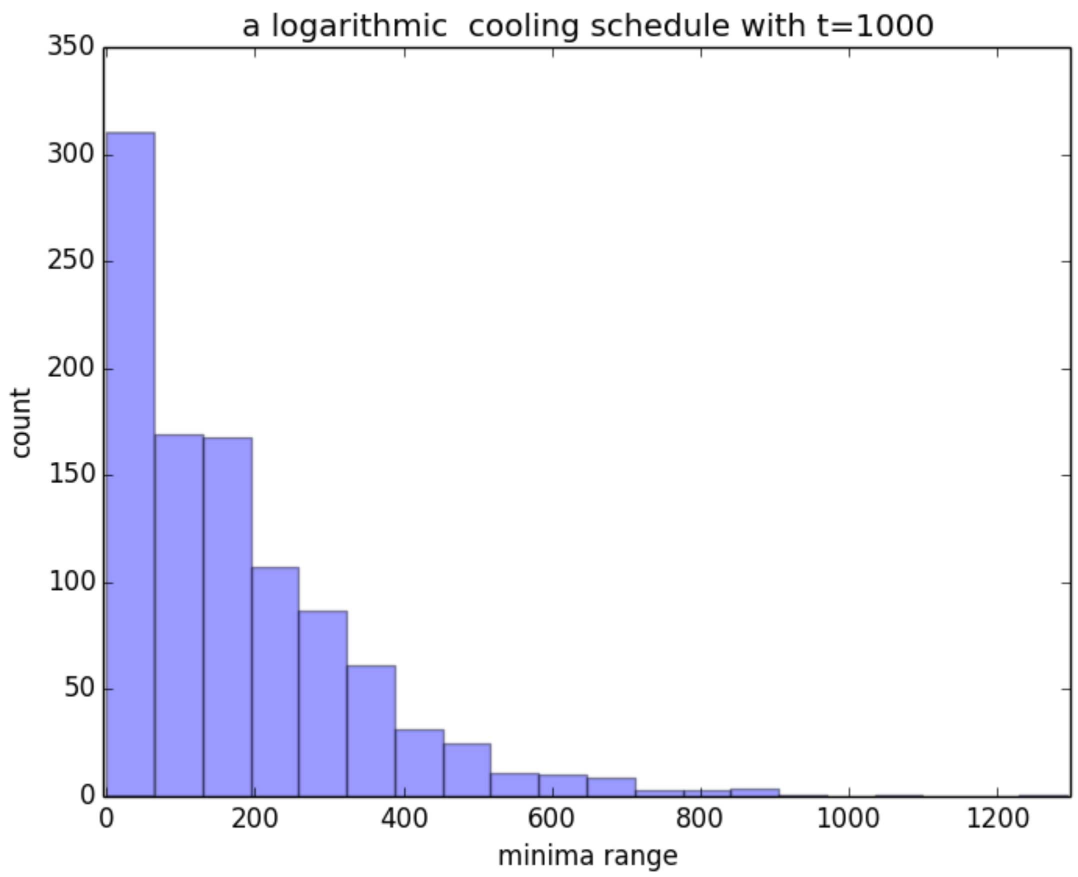


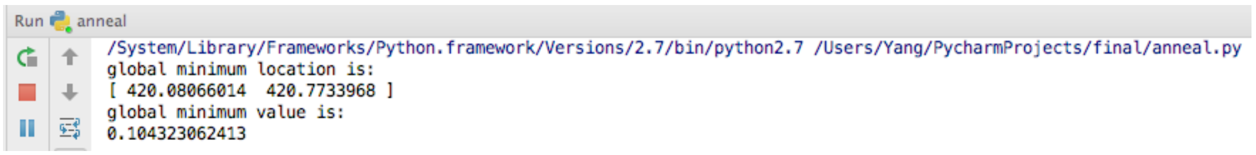
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 417.51333191  430.11516626]
global minimum value is:
12.0528799304
```





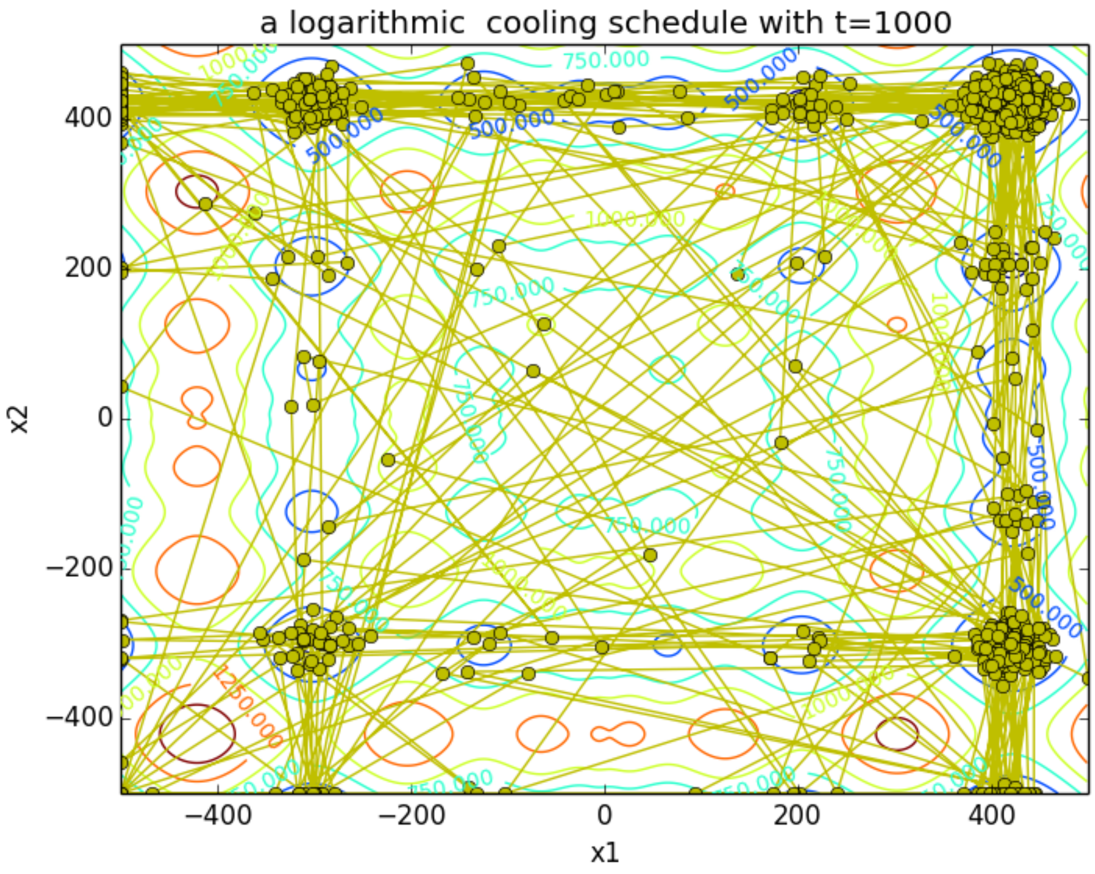
```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 420.08878036  421.24654293]
global minimum value is:
0.107436243057
```





```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/Yang/PycharmProjects/final/anneal.py
global minimum location is:
[ 420.08066014  420.7733968 ]
global minimum value is:
0.104323062413
```

iv) I chose logarithmic cooling schedule to run the procedure of simulated annealing, we can see that all of the blue areas, which means that the exact values of the function are smaller than 500, have been searched, and it finally converges to the smallest area in the upper right corner, and the global minimum it found is close to 0.



summary
The final project is harder than the previous ones, and it mainly focus on the Monte Carlo Method. The 1st problem is comparatively easy, I can generate 2-dimensional uniform random variables by combine the uniform random variable respectively in x and y. We can see that the mean ratio of points in the circle to the total sample points equals to the ratio of area of the ¼ circle to the unit square. We can use this equation to estimate pi. The second problem mainly use Monte Carlo Method to calculate finite integral. The 3rd problem is to use simulated annealing procedure with different iterations and different cooling schedules. In my experiment, the logarithmic cooling schedule shows the best performance.

source code
problem 1
(i)and (ii)

```
clear
```

```
n=10000;
k=50;
pi=ones(k,1);
for j=1:k
    x=rand(n,1);
    y=rand(n,1);
    %figure('color','white');
    %hold all
    %axis square;
    r=x.^2+y.^2;
    m=0;
    i=1;
    for i=1:n
        if r(i)<=1
            m=m+1;

           % plot(x(i),y(i),'b.');
        %else

            %plot(x(i),y(i),'r.');
        end
    end
    %fprintf('m=%d',m) ;fprintf('j=%d\n',j) ;

    pi(j,1)=m/(0.25*n);
    %fprintf('pi=%f\n',pi(j,1)) ;

end
figure; hist(pi,10);
%mean(pi)
fprintf('var=%f \n',var(pi));
```

(iii)

```
n = 1000;
f = zeros(n, 1);
f_sum = 0;
vf_sum = 0;
x = rand(n, 1);
y = rand(n, 1);
q = [x, y];

for j = 1 : n
    f(j) = abs(4*q(j,1)-2) * abs(4*q(j,2)-2);
    f_sum = f_sum + f(j);
end

ii = f_sum/n;

for m = 1 : n
    est_Varf = power((f(m) - ii), 2);
    vf_sum = vf_sum + est_Varf;
end

est_Varf = vf_sum / (n - 1);
est_Var = est_Varf / n;
err = sqrt(est_Var);
```

```
fprintf('integral=%f \n',ii);
fprintf('err=%f \n',err);
```

problem 2

problem 3

```
## Generate a contour plot
# Import some other libraries that we'll need
# matplotlib and numpy packages must also be installed
from __future__ import division
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
import random
import math
import warnings
warnings.filterwarnings("ignore")


# define objective function
def f(x):
    x1 = x[0]
    x2 = x[1]
   # obj = 0.2 + x1**2 + x2**2 - 0.1*math.cos(6.0*3.1415*x1) -
0.1*math.cos(6.0*3.1415*x2)

    obj = 418.9829 * 2 - x1 * math.sin(math.sqrt(abs(x1))) - x2*
math.sin(math.sqrt(abs(x2)));
    return obj

# Start location
x_start = [0.8, -0.5]

# Design variables at mesh points
i1 = np.arange(-500, 500, 1)
i2 = np.arange(-500, 500, 1)
x1m, x2m = np.meshgrid(i1, i2)
fm = np.zeros(x1m.shape)
for i in range(x1m.shape[0]):
    for j in range(x1m.shape[1]):
        #fm[i][j] = 0.2 + x1m[i][j]**2 + x2m[i][j]**2 \
        #     - 0.1*math.cos(6.0*3.1415*x1m[i][j]) \
        #     - 0.1*math.cos(6.0*3.1415*x2m[i][j])
        fm[i][j] = 418.9829 * 2 - x1m[i][j] * math.sin(math.sqrt(abs(x1m[i][j]))) -
x2m[i][j] * math.sin(math.sqrt(abs(x2m[i][j])));
# Create a contour plot
plt.figure()
# Specify contour lines
#lines = range(2,52,2)
# Plot contours
CS = plt.contour(x1m, x2m, fm)#,lines)
# Label contours
plt.clabel(CS, inline=1, fontsize=10)
# Add some text to the plot
plt.title('a logarithmic  cooling schedule with t=100')
plt.xlabel('x1')
plt.ylabel('x2')

##################################################
# Simulated Annealing
```

```
##################################################
# Number of cycles
n = 100 #n=20,50,100,1000
# Number of trials per cycle
m = 100
# Number of accepted solutions
na = 0.0
# Probability of accepting worse solution at the start
p1 = 0.7
# Probability of accepting worse solution at the end
p50 = 0.001
# Initial temperature
t1 = -1.0/math.log(p1)
# Final temperature
t50 = -1.0/math.log(p50)
# Fractional reduction every cycle
frac = (t50/t1)**(1.0/(n-1.0))#exponential

# Initialize x
x = np.zeros((n+1,2))
x[0] = x_start
xi = np.zeros(2)
xi = x_start
na = na + 1.0
# Current best results so far
xc = np.zeros(2)
xc = x[0]
fc = f(xi)
fs = np.zeros(n+1)
fs[0] = fc
# Current temperature
t = t1
# DeltaE Average
DeltaE_avg = 0.0
for i in range(n):
   # print 'Cycle: ' + str(i) + ' with Temperature: ' + str(t)
    for j in range(m):
        # Generate new trial points
        xi[0] = xc[0] + 500*random.random() - 250
        xi[1] = xc[1] + 500*random.random() - 250
        # Clip to upper and lower bounds
        xi[0] = max(min(xi[0],500),-500)
        xi[1] = max(min(xi[1],500),-500)
        DeltaE = abs(f(xi)-fc)
        if (f(xi)>fc):
            # Initialize DeltaE_avg if a worse solution was found
            #   on the first iteration
            if (i==0 and j==0): DeltaE_avg = DeltaE
            # objective function is worse
            # generate probability of acceptance
            p = math.exp(-DeltaE/(DeltaE_avg * t))
            # determine whether to accept worse point
            if (random.random()<p):
                # accept the worse solution
                accept = True
            else:
                # don't accept the worse solution
                accept = False
        else:
            # objective function is lower, automatically accept
            accept = True
        if (accept==True):
            # update currently accepted solution
            xc[0] = xi[0]
            xc[1] = xi[1]
            fc = f(xc)
            # increment number of accepted solutions
            na = na + 1.0
            # update DeltaE_avg
            DeltaE_avg = (DeltaE_avg * (na-1.0) +  DeltaE) / na
    # Record the best x values at the end of every cycle
    x[i+1][0] = xc[0]
    x[i+1][1] = xc[1]
    fs[i+1] = fc
    # Lower the temperature for next cycle
    #t = frac * t#expo
    t = t1/(math.log(3+i))
    #t = t1/(1+1e-5*i*i)


plt.plot(x[:,0],x[:,1],'y-o')
#Save the figure as a PNG
plt.savefig('final.png')

# print solution
print 'global minimum location is: '
print x[np.argmin(fs)]
print 'global minimum value is:'
print min(fs)

fig = plt.figure()

bins = np.linspace(min(fs),max(fs),21)  # fixed number of bins

plt.xlim([min(fs) - 5, max(fs) + 5])

plt.hist(fs, bins=bins, alpha=0.4)
plt.title('a logarithmic  cooling schedule with t=100')
plt.xlabel('minima range')
plt.ylabel('count')

#Save the figure as a PNG
plt.savefig('final2.png')

plt.show()
```