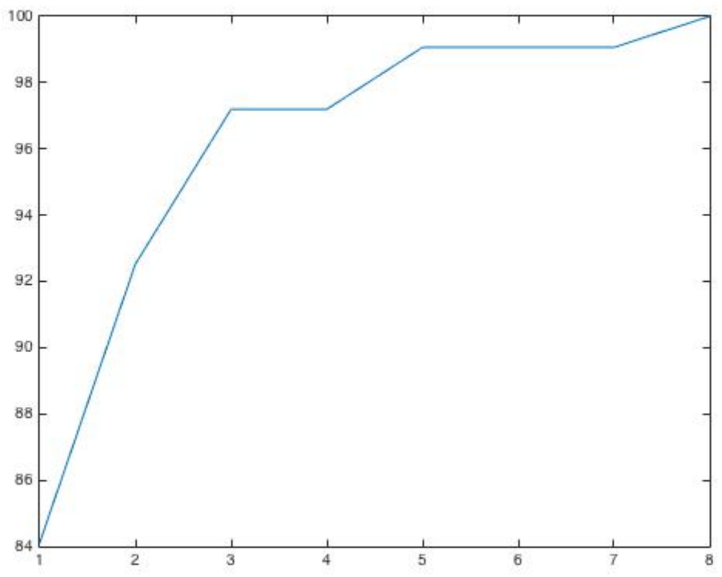
1. Genetic Algorithm

(a)

I ran my algorithm multiple times and got the same result every time. The average best fitness is 100%. In other words, the average accuracy is also 100%.

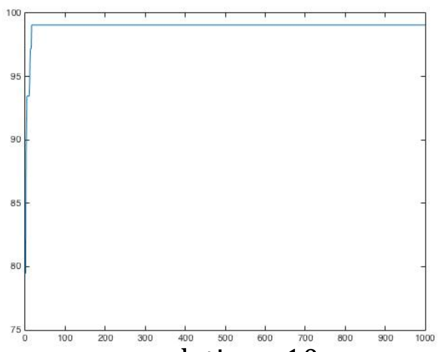
(b)

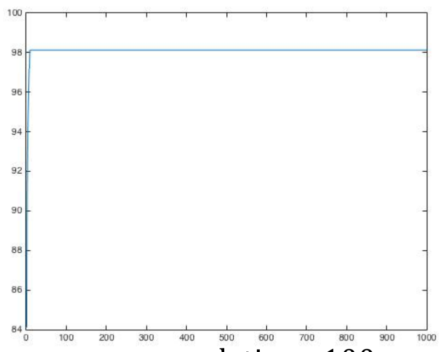
Using the same parameter from problem (a), the plot displays the fitness value as a function of iterations. It quickly goes to the 100% fitness value. In the beginning, fitness value goes up as the iteration increases. It makes sense because the more iterations it processes, the bigger chance it will reach the goal.

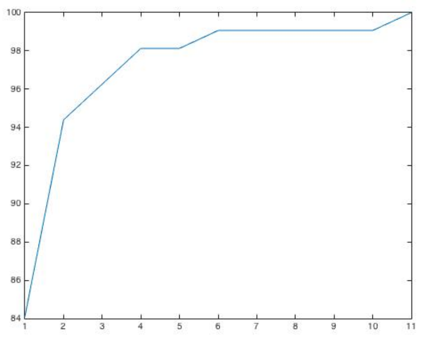


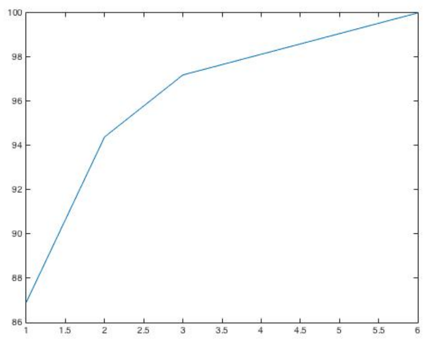
(c)

Population size	10	100	1000	10000
Mean accuracy	98.32%	98.13%	100%	100%
Average number of iteration	1000(sec)	1000(sec)	7(sec)	4(sec)
Average run time	3(sec)	35(sec)	6(sec)	58(sec)


population =10


population =100

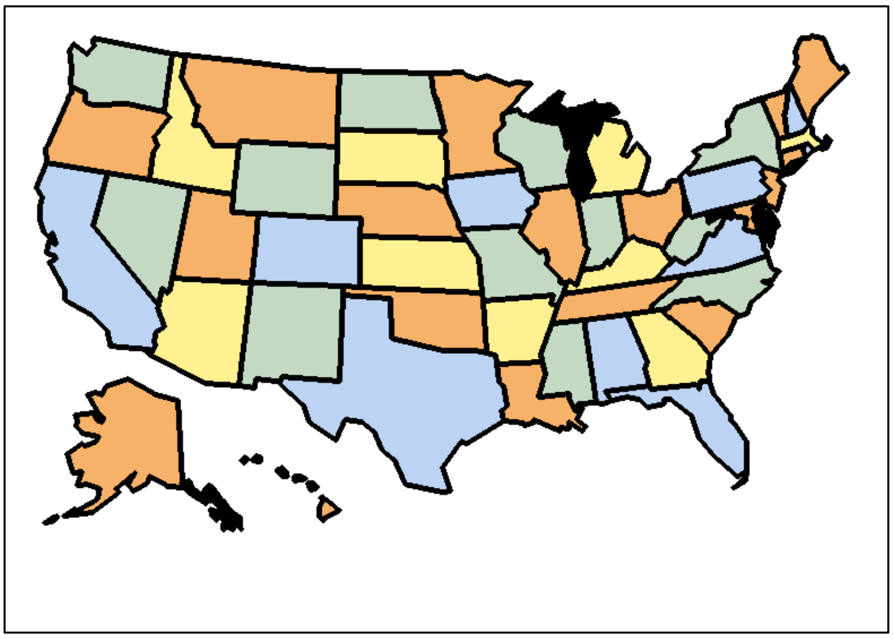

population =1000


population=10000

In the table shown above, we can see only when population size are 1000 or 10000 can it reach 100% accuracy, while less population couldn't. Also, with 1000 and 10000 population, it only needs less then 10 iterations to reach the goal. What's more, as to the run time needed, the time consumed with 10000 populations is much longer than others. Although it only runs less then 10 iterations, it takes too much time on each iteration, which is not very efficient. The four plot shown above shows the fitness value varying with iterations.

(d)

I gave 33rd and 40th genes the value of 1 in the beginning. And during my algorithm, I made crossover and mutation methods avoid changing the value of 33rd and 40th genes, so that I can keep the color of these two states staying the same.



Extra credit:

The method I used in my code is that whenever I extract two best offspring from all, I test their colors and know which colors appears most frequently and which is the least. Then I changed some of the most frequent color into the least one in order to balance the amount of color being used.

(see the code from line 27 to line 47)

However, there are 50 states and 4 colors. 50/4=12.5. In other words, there's no chances that each color is used an equal number of times. I can only balance them as much as I can.

2.

Using Matlab’s quagprog function:

X=4.9000

Y=3.3667

Objective: 65.49

Using Matlab’s fmincon function:

X=4.9000

Y=3.3667

Objective: 65.49

Parameters:

```
H=[1 0;0 9];
f=[3.2;-6];
lb=[0,0];
ub=[];
Aeq=[];
Beq=[];
A=[-1 -3;2 5; 3 4];
b=[-15; 100; 80];
```

3.

(a)

objective function: 4W+2d

constraint:

σ-σb<=0

σ-σy<=0

d-d2<=0

d1-d<=0

t-t2<=0

t1-t<=0

(b)

resulting thickness: 6.12cm

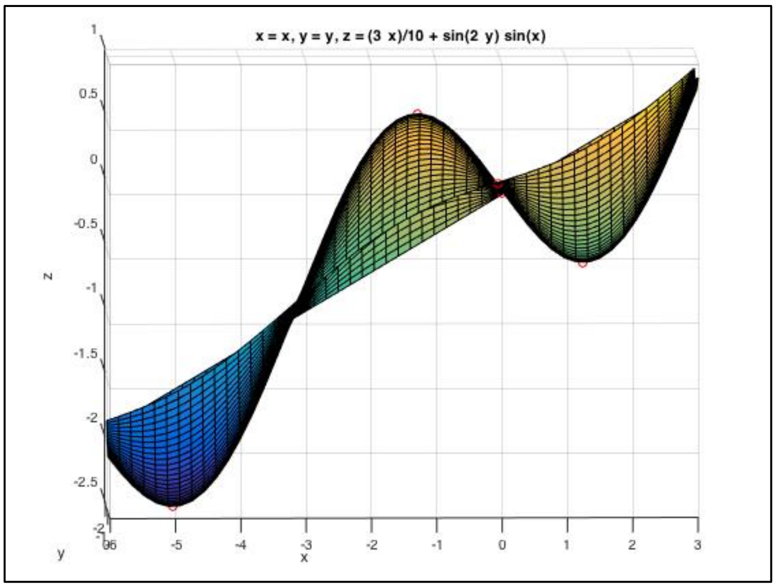
resulting diameters: 0.19 cm

solution file: as attachment

(c)
resulting thickness: 6.12cm
resulting diameters: 0.19 cm
solution file: as attachment

(d)
resulting thickness: 6.12cm
resulting diameters: 0.19 cm
solution file: as attachment

(e)
in this question I test the consistency of these methods with changing the value of E from 900000 to 9000. It turned out that the results are the same. The results are shown below.
Excel solver:
Cost : 115
d = 10
t = 1
Matlab function:
Cost : 115
d = 10
t = 1

4. Genetic Algorithm and Minimization
(a)
visualization:


(b)

I used "solve" function to solve x and y and also their periodic parameter. And arbitrarily set those periodic parameters to find the feasible x and y. Then I plot the points mapping from these x and y on the surface to see which are maximums and minimums. The results are shown below.

stationary points:

[-1.2661 -0.7854]	→ local maximum →	0.5741
[-5.0171 -0.7854]	→ global minimum →	-2.4591
[0 -1.4184]	→ local maximum →	0
[1.2661 -0.7854]	→local minimum →	-0.5741
[0 -0.1523]	→local maximum →	0

(c)

- Genome design + initial population

  I randomly generated several points of x and y in the given range. And combine each x and y to become a candidate X.
- Fitness function

  The fitness function is defined by the difference between the z value and the global minimum which is -2.459.
- Implement different cases
  1. Crossover, no mutation:

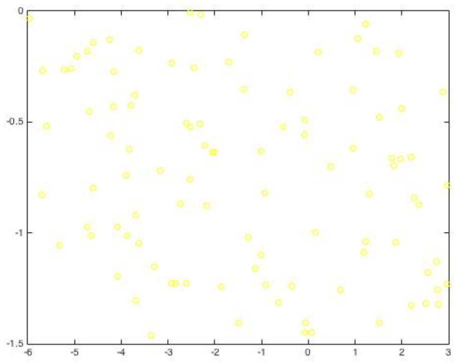
     In my code, if you set "cRate" equals to 0 and "mRate" equals to 1, it would only crossover the candidates. If that's the case, it barely reaches the global minimum. The reason is that crossover can only swap x and y value with other candidates, meaning that if you didn't get the right points from the initialization, then it will never get the global minimum.
  2. Mutation, no crossover:

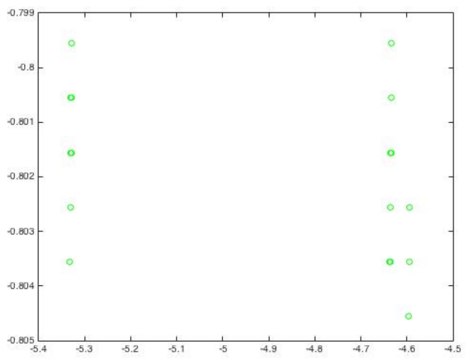
     On the other hand, if you set "cRate" as 1 and "mRate" as 0, it would only mutate the candidates. Only by mutation, it takes way more time to reach the global minimum. The reason is that in my mutation function, I add or minus 0.0001 to x and y. Without crossover, it's too slow to get closed to the global minimum.
  3. Both:

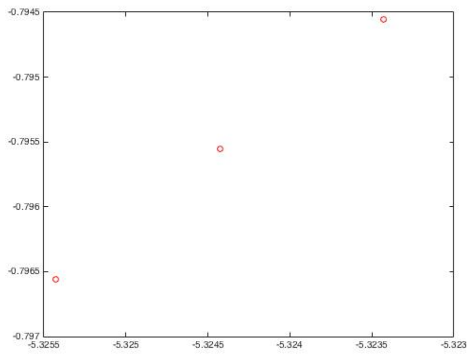
     If you set "cRate" between 0 and 1, it runs both crossover and mutation.

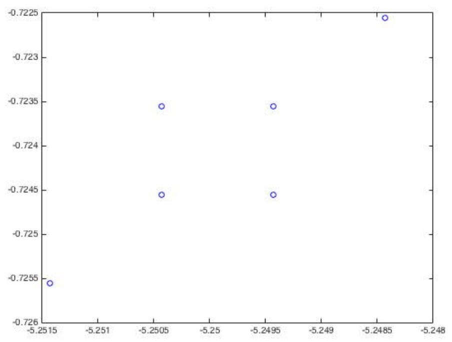
It is the best algorithm among these three. Because crossover can speed up the process approaching the minimum, and mutation can slight adjust the x and y value to reach the global minimum.

- 4% elitism vs. no elitism
  In my code, if you set "e" as 0, meaning that you don't use elitism method. Without elitism, there is no way to ensure the offspring is better than parents. In my genetic algorithm, I randomly chose parents among all, so it will never reach global minimum without elitism.
- plot the population at various stage


initialization


iteration=5


Iteration=10


iteration=100