28 Mar 2015

AE 8803 (GER)

PROJECT: STEAM LOCOMOTIVE DESIGN OPTIMIZATION

1. Formal Optimization problem statement -

We note that $l \geq 2R+12$ is a constraint equation. However observing the objective function and constraining equations we realize that $l > 2R+12$ provides no design benefit.

∴ We assume $\boxed{l = 2R+12}$ in this design problem.

Problem statement -

$$\min_{\vec{x}} f(\vec{x}) = \min_{\vec{x}} [2lhb + 4(0.4)(4)\pi R^2]$$

by changing $\vec{x} = [h \; b \; R \; r]^T$

subject to $-f_h + (FOS)\frac{RT}{4r} \leq 0$ —— ① where FOS = factor of safety = 2

$$-f_b + (FOS)\frac{RT}{4r} \leq 0 \quad —— ②$$

$$\frac{(FOS)Mh}{2I_h} - S_e \leq 0 \quad —— ③$$ where $M = ml\,a_c/g$,

$a_c = r\omega^2$,

$\omega = V_{max}/R$

$\omega - \omega_{max} \leq 0$ —— ④ where $\omega_{max} = 300$ rpm $= 10\pi$ r/s

$-T + 32000 \leq 0$ —— ⑤ where $T = (0.85)(200\text{ psi})\cdot(24)^2(2r)/(2R)$

$r - R \leq 0$ —— ⑥

$[1, 1, 20, 1] \leq [h, b, R, r] \leq [10, 10, 60, 60]$ —— ⑦

We simplify the problem statement here—

$$\min_{\vec{x}} f(\vec{x}) = \min_{\vec{x}} \left[2(2k+12)\, h\, b + (4)(0.4)(4)\pi R^2 \right]$$

by changing $\vec{x} = [h \quad b \quad R \quad r]^T$

subject to $$\frac{-\pi^2 E I_h}{l^2} + (FOS)\frac{RT}{4r} \leq 0$$

$$\frac{-\pi^2 E I_h}{2l^2} + (FOS)\frac{RT}{4r} \leq 0$$

$$(FOS)\frac{Mh}{2I_h} - Se \leq 0$$

$$w - w_{max} \leq 0$$

$$-T + 32000 \leq 0$$

$$r - R \leq 0$$

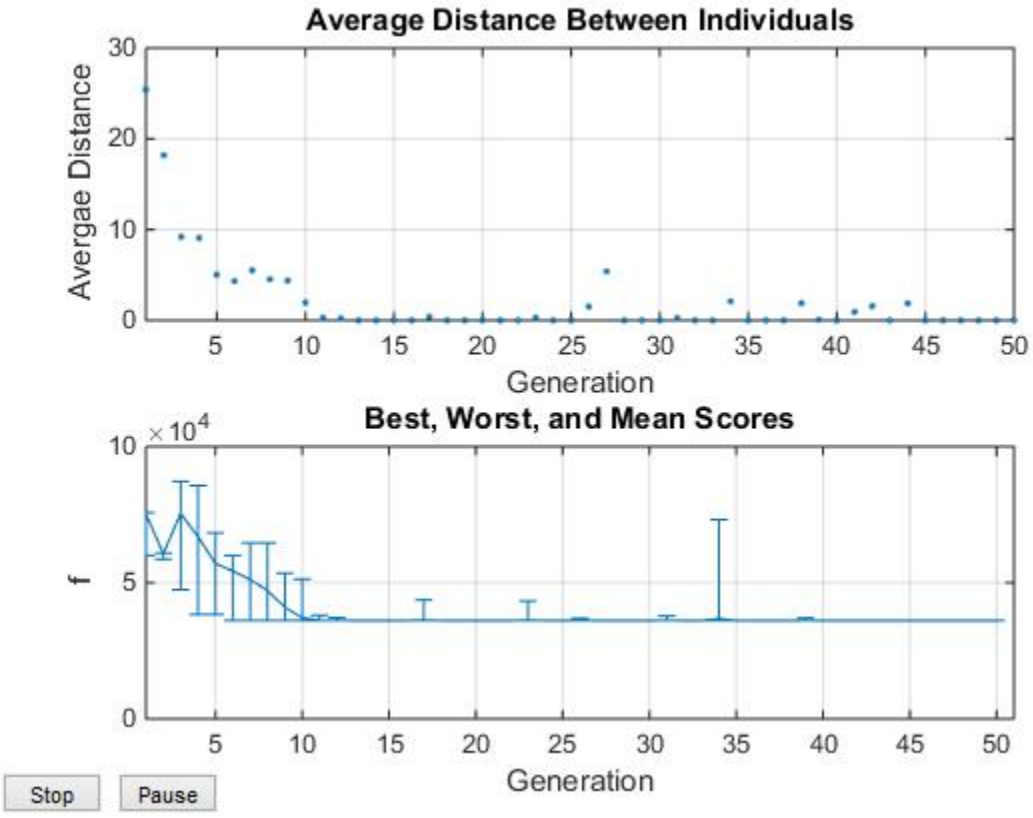
$$[1, 20, 1] \leq [h \quad b \quad R \quad r] \leq [10, 10, 60, 60]$$

Task 2

For one run of the baseline MATLAB GA function for this problem we obtain the results:

x_opt = [6.7823 3.6505 39.5816 19.5301]' inches.

The value of the objective function at x_opt is f(x_opt) = 36015 in^3.



The code for the nonlinear constrasints function used for the MATLAB GA function is included here.

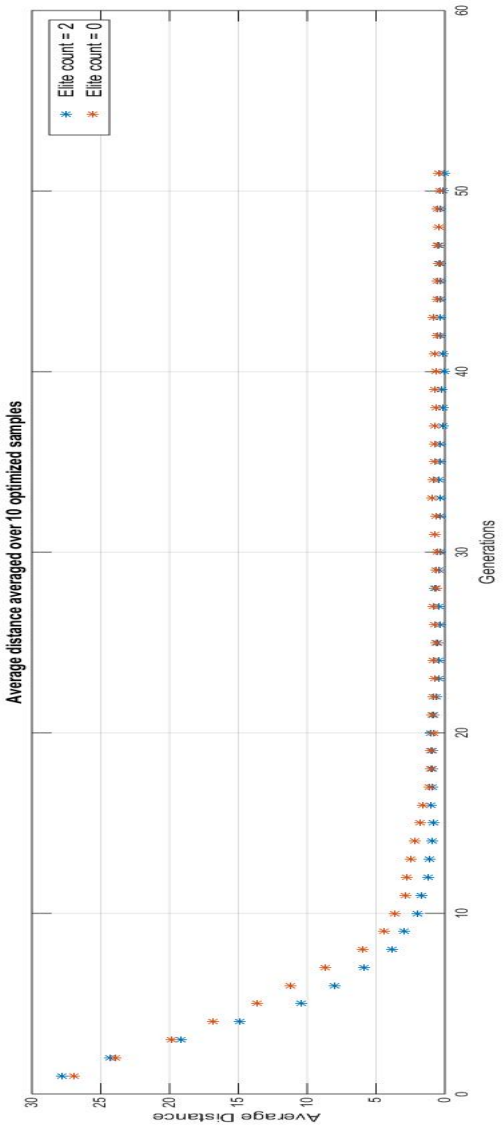
```
function [c,ceq] = funcConstraint_eqns(x)
% AE 8803 (GER) Project
% Function returning constraint equations to use for nonlcon input argument
% to MATLAB's ga() function.


global omega_max
global E
global S_e
global rho_s
global FOS

% Specifying design constants
omega_max = 10*pi; % -> rad/s % Maximum rotational speed of drive wheel
E = 29e6; % ->psi % Young's/Elastic modulus
S_e = 28e3*(32.18*12); % ->psi*(32.18*12)->lb/(in*s^2) % Endurance limit
rho_s = 0.28; % ->lb/in^3
```

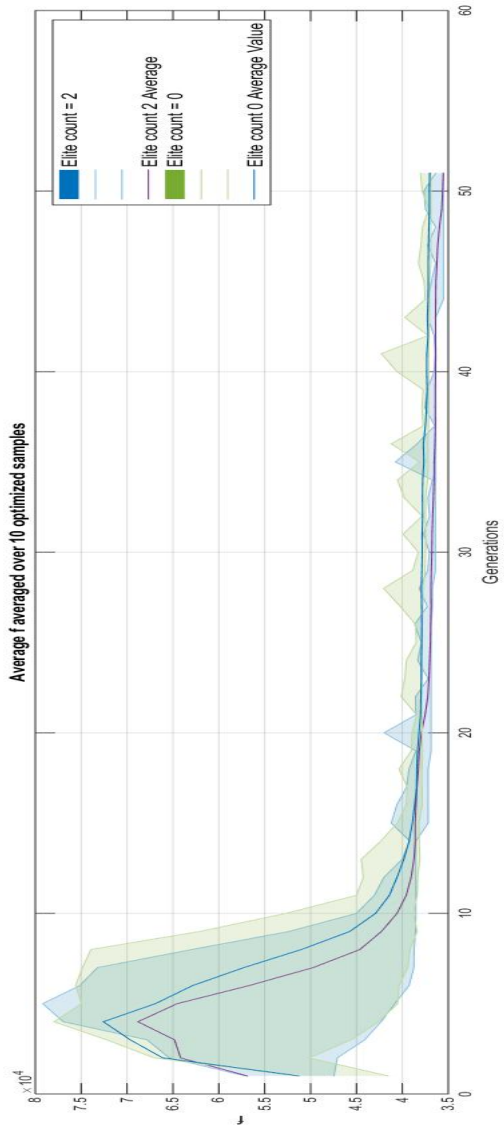
```
FOS = 2; % Factor of safety


% Design state vector: x = [h b R r]

h = x(1);
b = x(2);
R = x(3);
r = x(4);

l = 2*R+12; % Length of connecting rod

% Area moments of inertia
I_h = b*h^3/12;
I_b = h*b^3/12;

T = 0.85*200*24^2*r/R;

f_h = pi^2*E*I_h/l^2;
c(1) = -f_h + FOS*R*T/(4*r);

f_b = pi^2*E*I_b/(2*l^2);
c(2) = -f_b + FOS*R*T/(4*r);

v_max = 70*17.6; % mph*17.6 -> in/s
omega = v_max/R;
a_c = omega^2*r; % Maximum centripetal acceleration of connecting rod
M = rho_s*l*h*b * l * a_c / 8; % Maximum bending moment of connecting rod
c(3) = FOS*M*h/(2*I_h) - S_e;

c(4) = r - R;

c(5) = omega - omega_max;

c(6) = -T + 32000;


ceq = [];


end
```

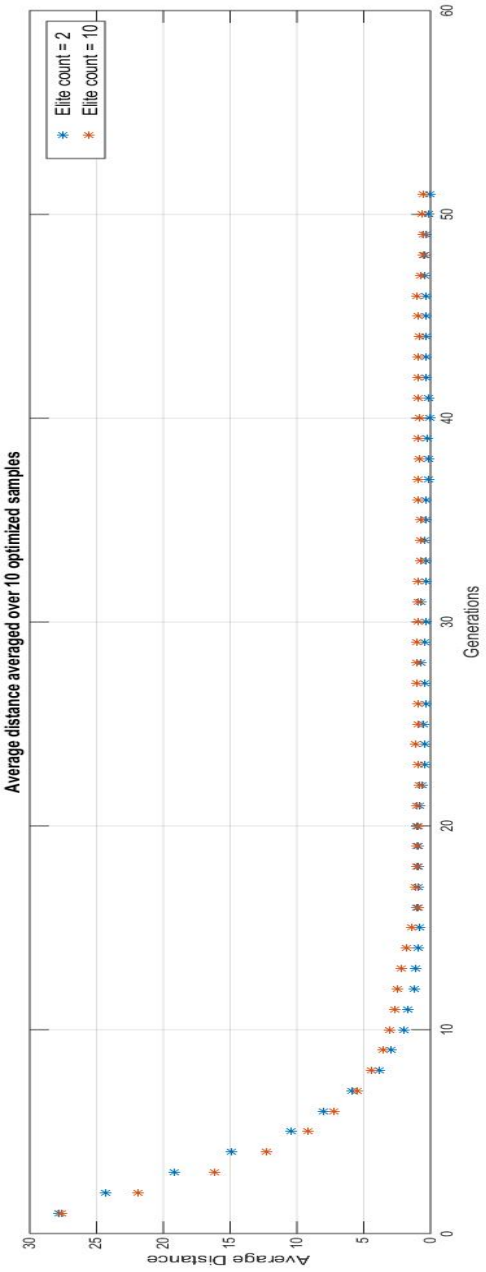
Task 3

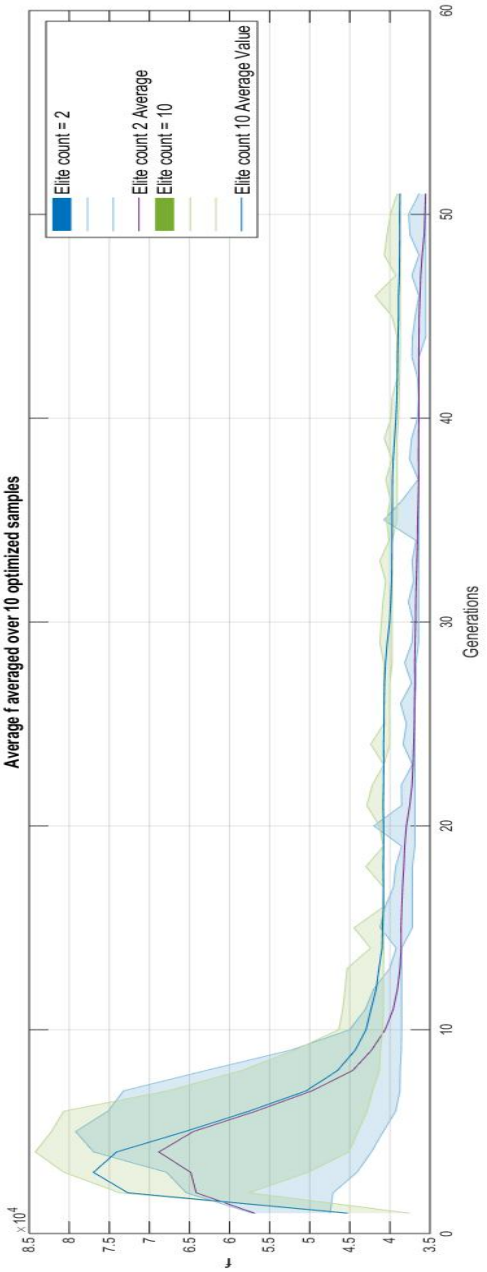
We plot the required graphs here for the comparing each case of varying the elite count, crossover fraction, mutation rate and the population size with the baseline settings of these parameters. This is followed by a discussion on the optimizer behaviour. The baseline parameters are elite count = 2, crossover fraction = 0.75, mutation rate = 0.01, population size = 50.

Elite count variation: Figures 1A and 1B here show the required plots for the 10 samples averaged cases of variation of the elite count being 0 and 10 as compared with an averaged 10 samples averaged plots for the baseline case.

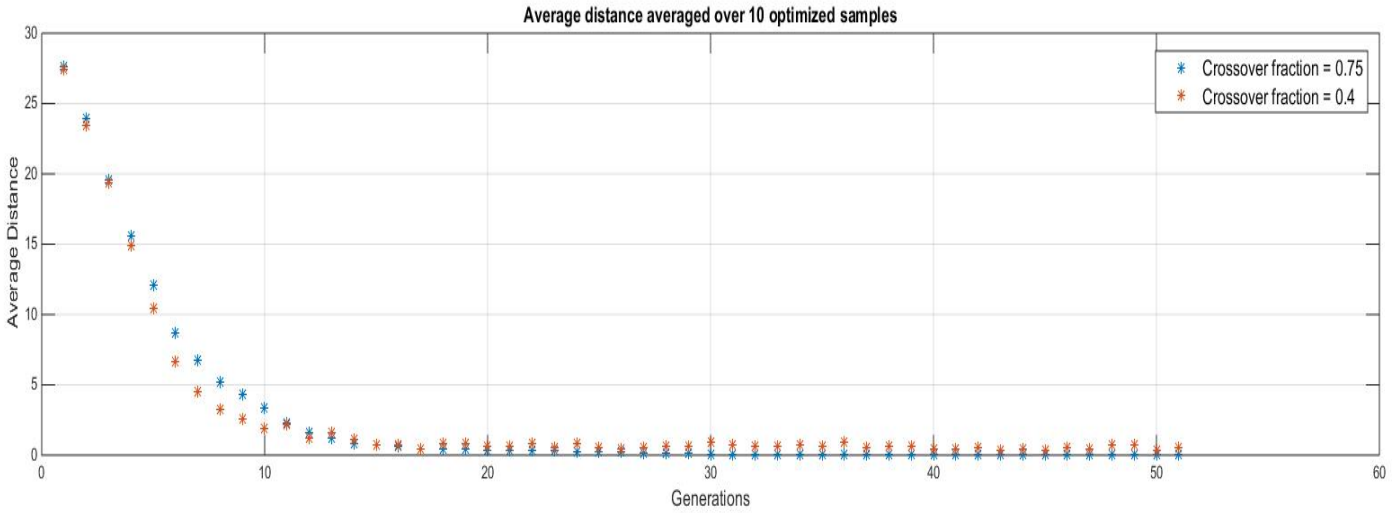


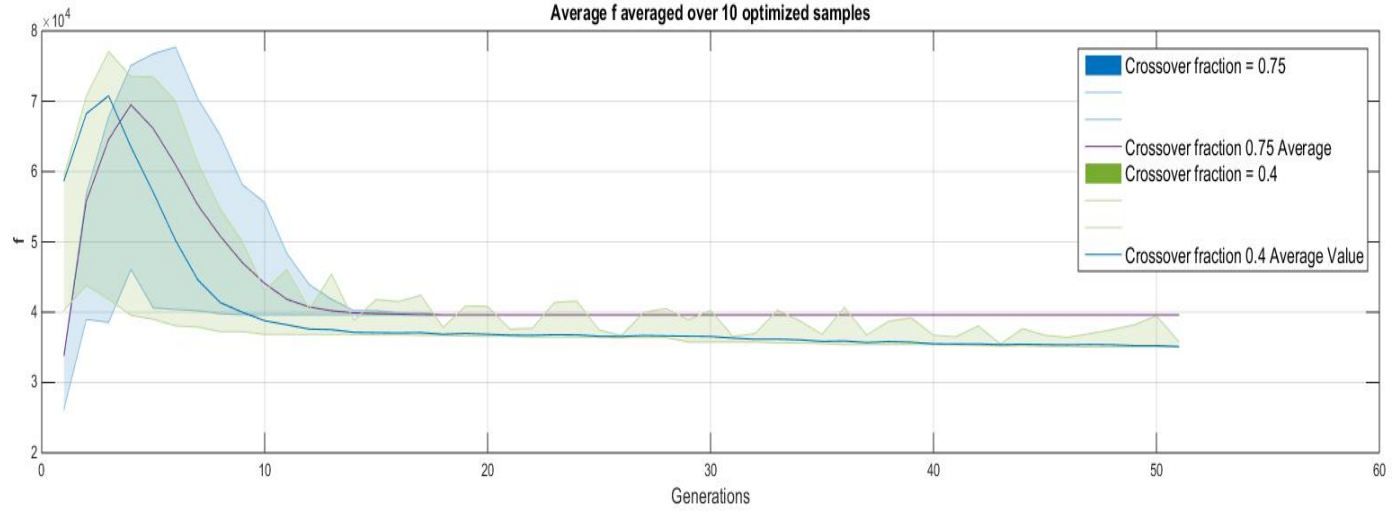


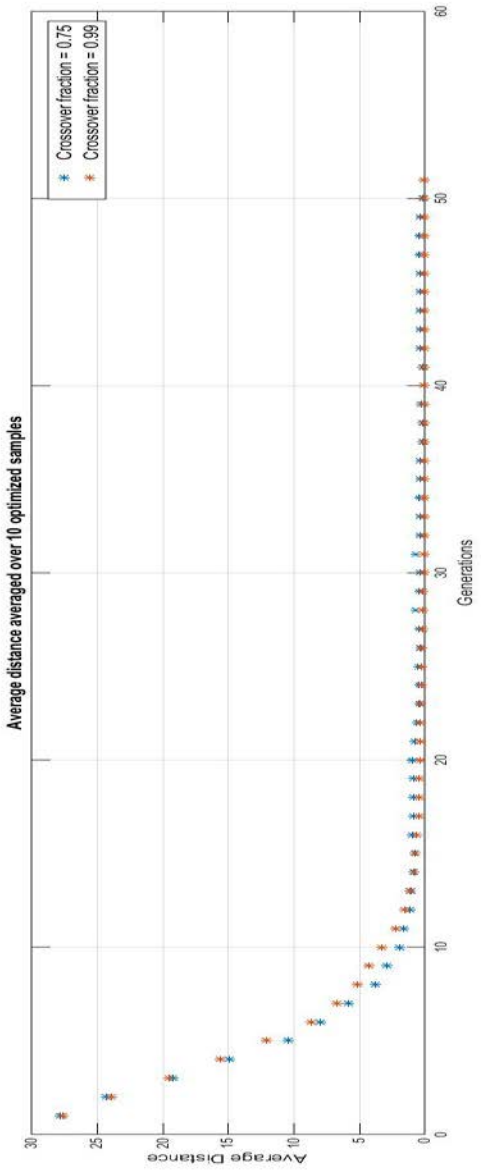


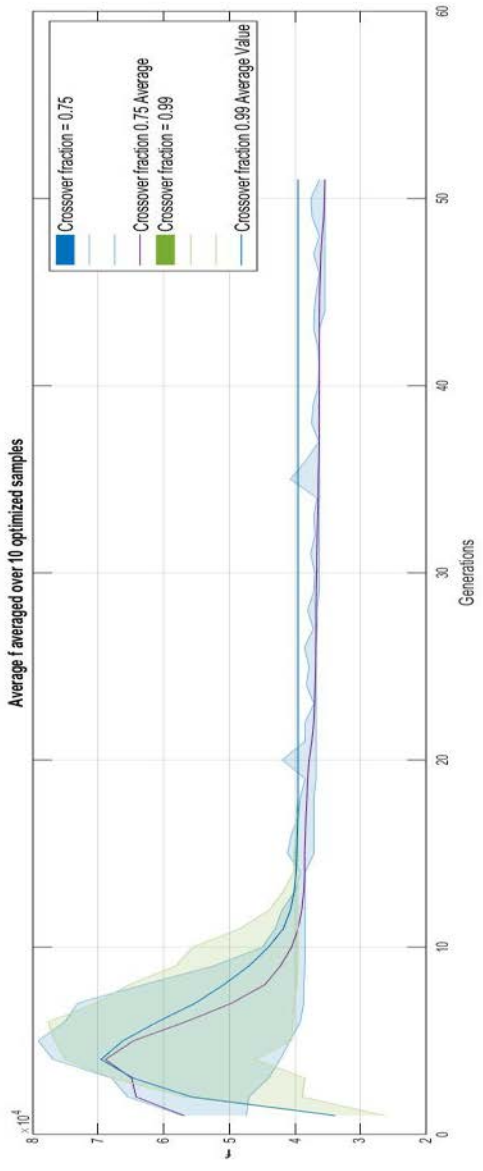


Crossover fraction variation: Figures 2A and 2B here show the required plots for the 10 samples averaged cases of variation of the cross over fraction being 0.4 and 0.99 as compared with an averaged 10 samples averaged plots for the baseline case.

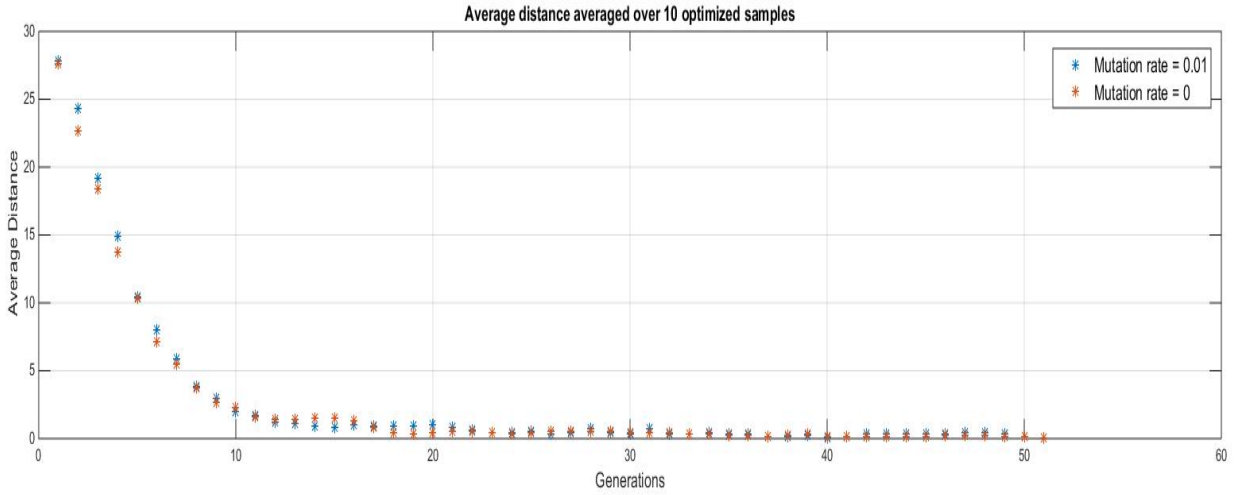

Average distance averaged over 10 optimized samples
Average Distance
Generations
Crossover fraction = 0.75
Crossover fraction = 0.4
0
5
10
15
20
25
30
0
10
20
30
40
50
60

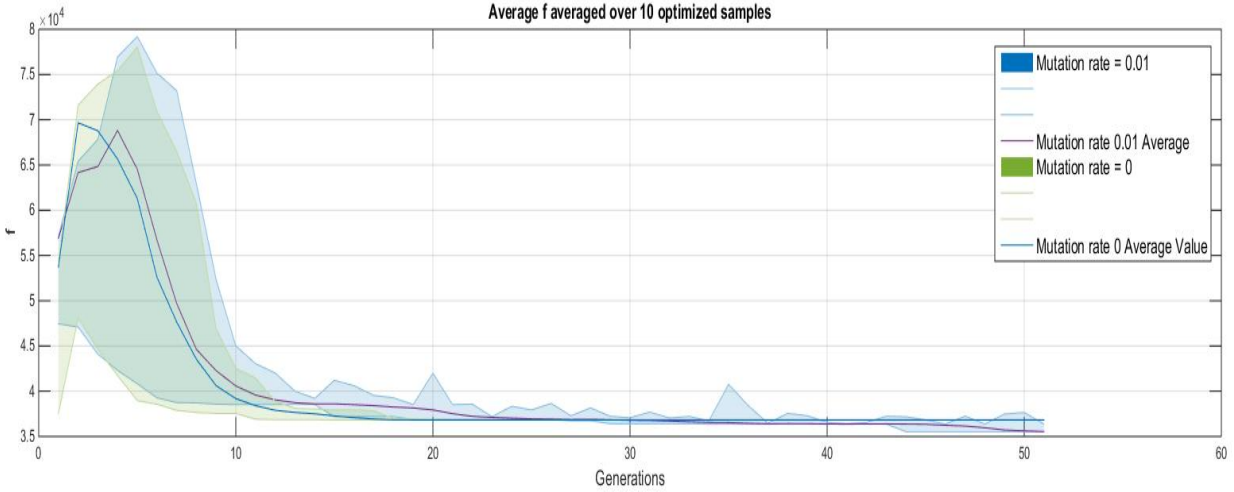

Average f averaged over 10 optimized samples
×10^4
f
Generations
Crossover fraction = 0.75
Crossover fraction 0.75 Average
Crossover fraction = 0.4
Crossover fraction 0.4 Average Value
2
3
4
5
6
7
8
0
10
20
30
40
50
60

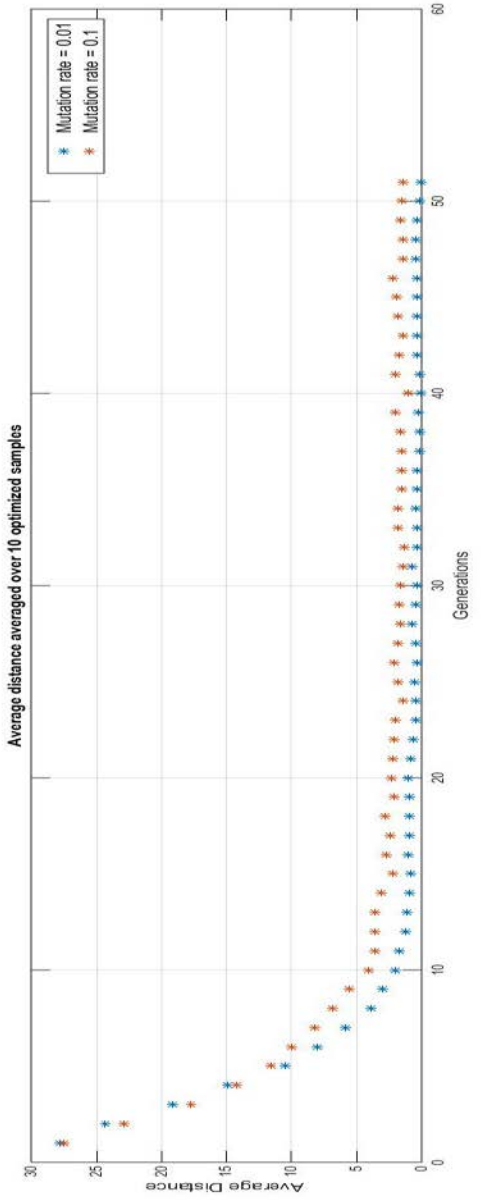


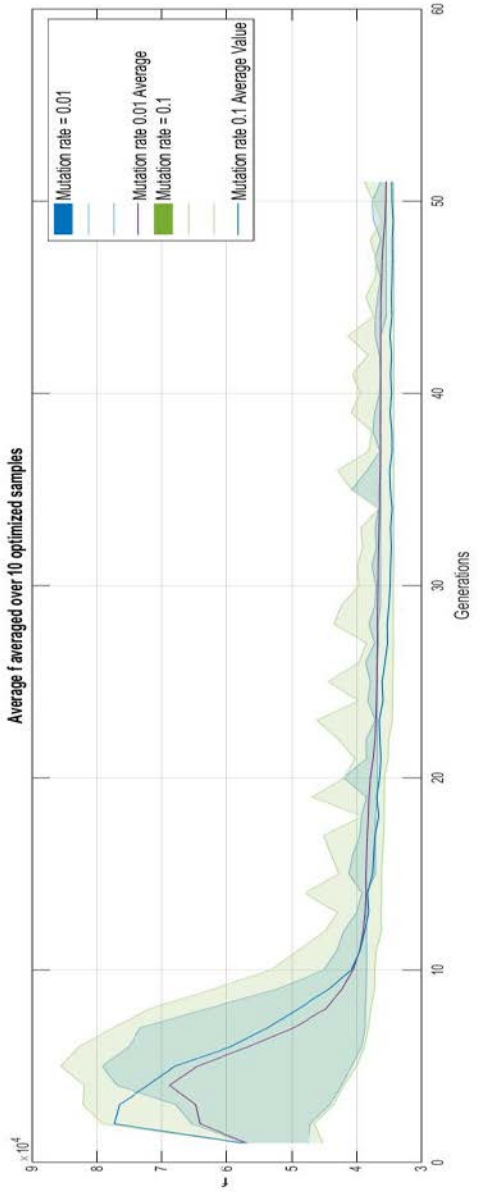


Mutation rate variation: Figures 3A and 3B here show the required plots for the 10 samples averaged cases of variation of the mutation rate being 0 and 0.1 as compared with an averaged 10 samples averaged plots for the baseline case.


Average distance averaged over 10 optimized samples
Average Distance
Generations
Mutation rate = 0.01
Mutation rate = 0

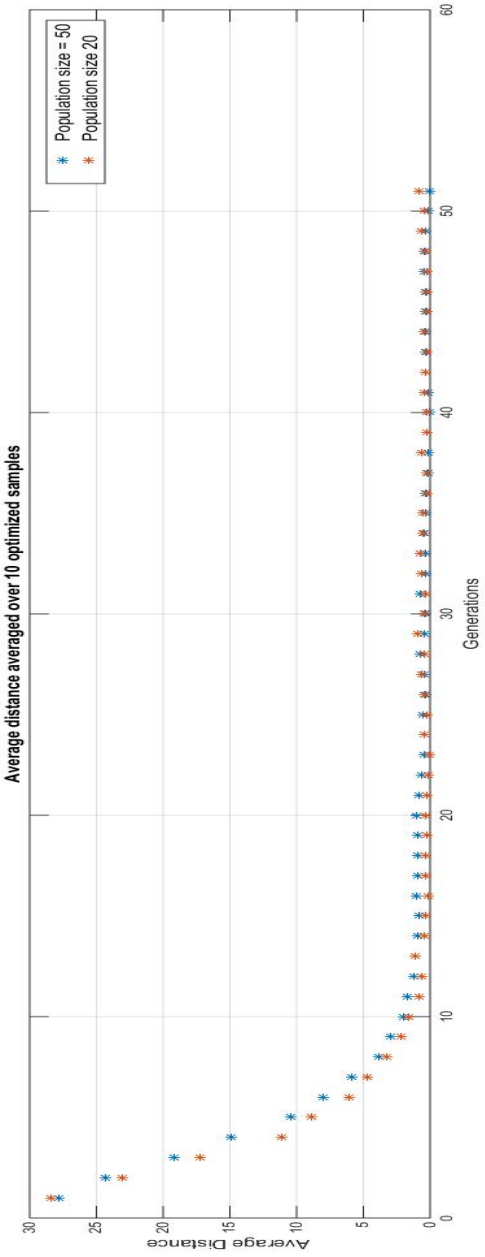

Average f averaged over 10 optimized samples
f
Generations
Mutation rate = 0.01
Mutation rate 0.01 Average
Mutation rate = 0
Mutation rate 0 Average Value

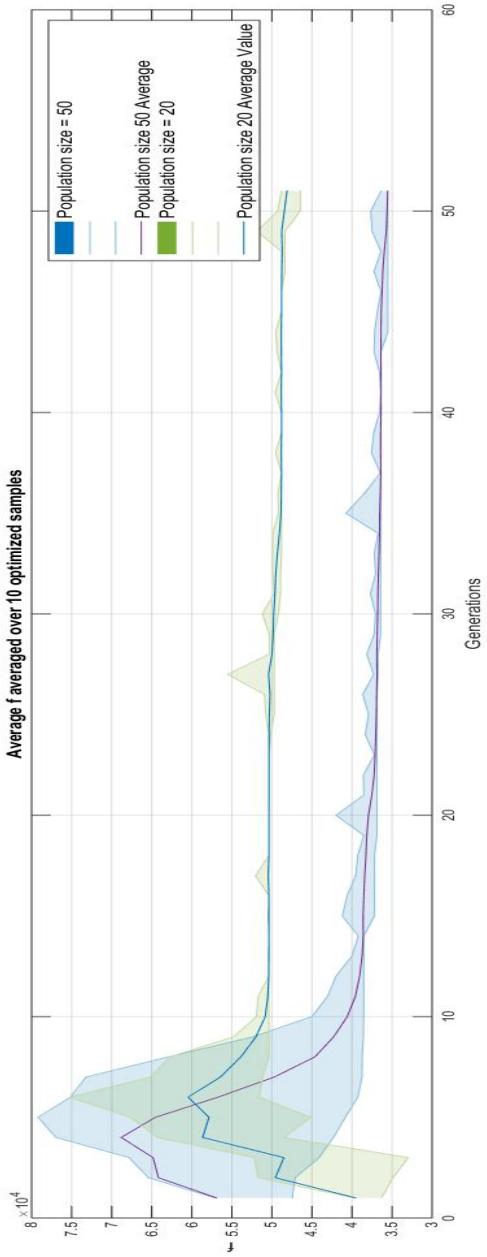


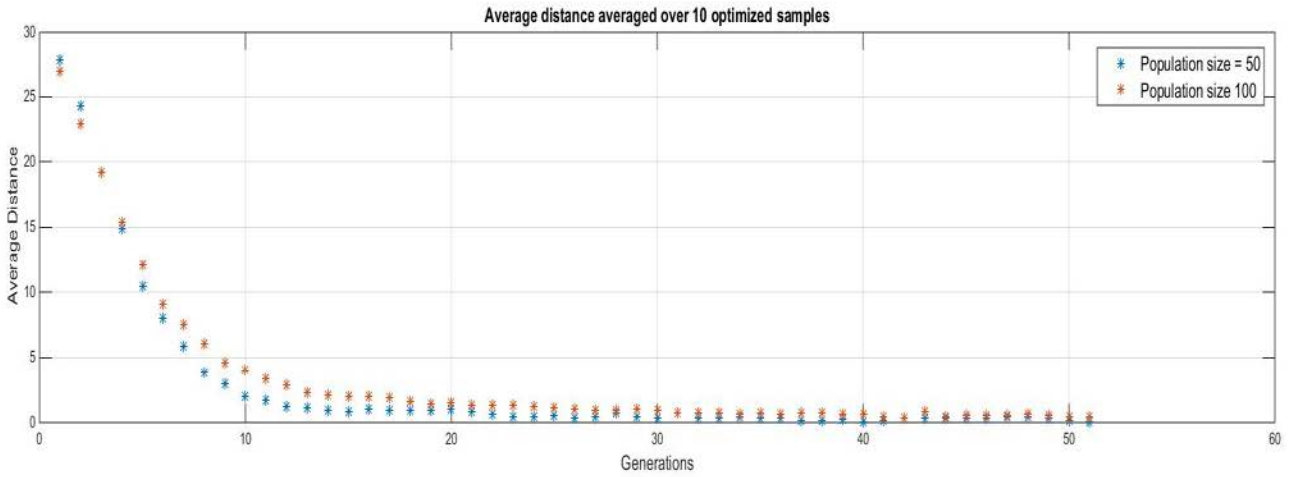


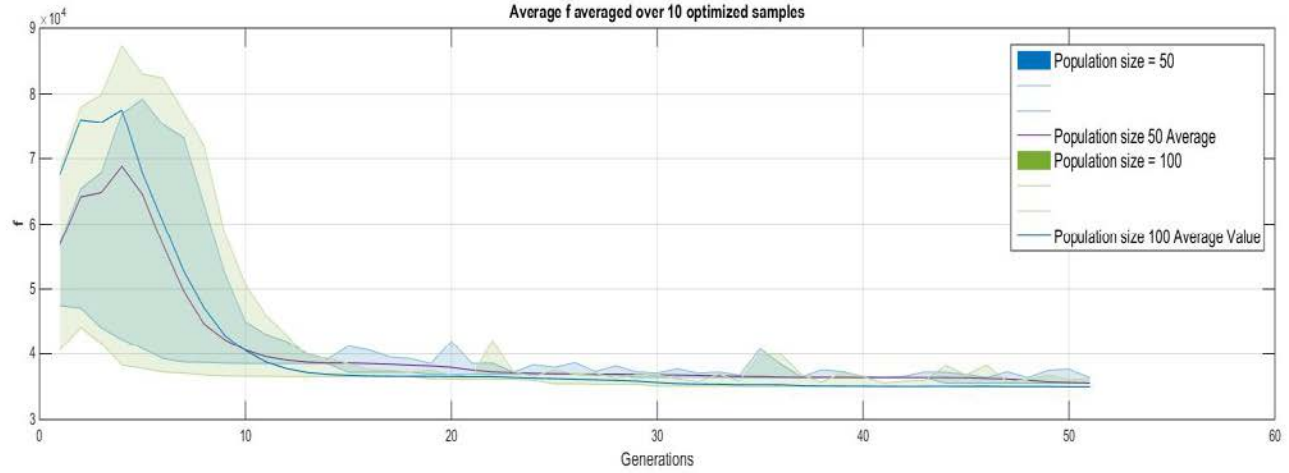
Population size variation: Figures 4A and 4B here show the required plots for the 10 samples averaged cases of variation of the population size being 20 and 100 as compared with an averaged 10 samples

averaged plots for the baseline case.

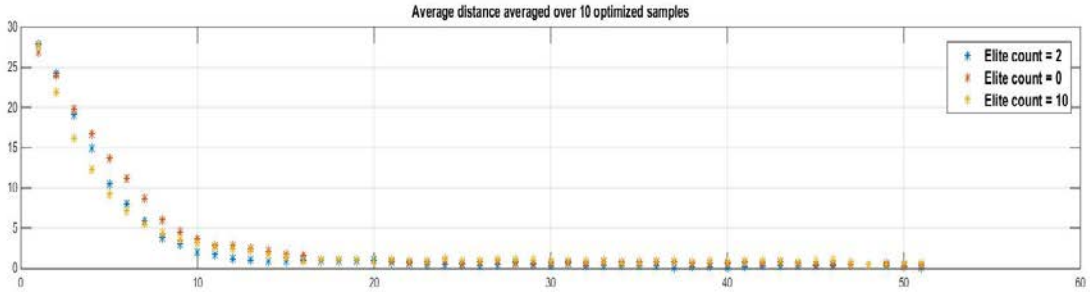


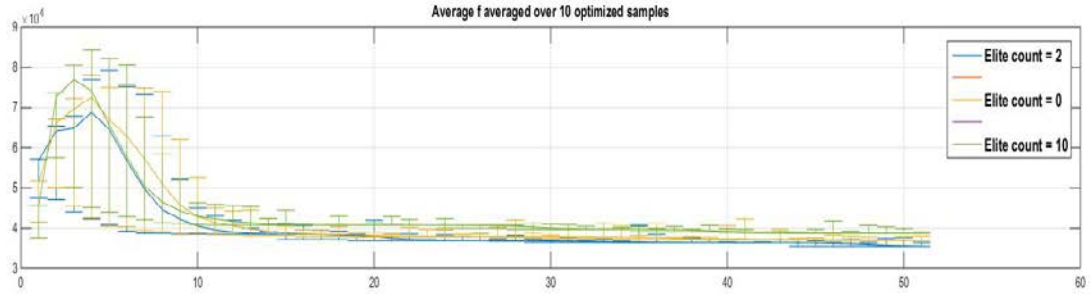


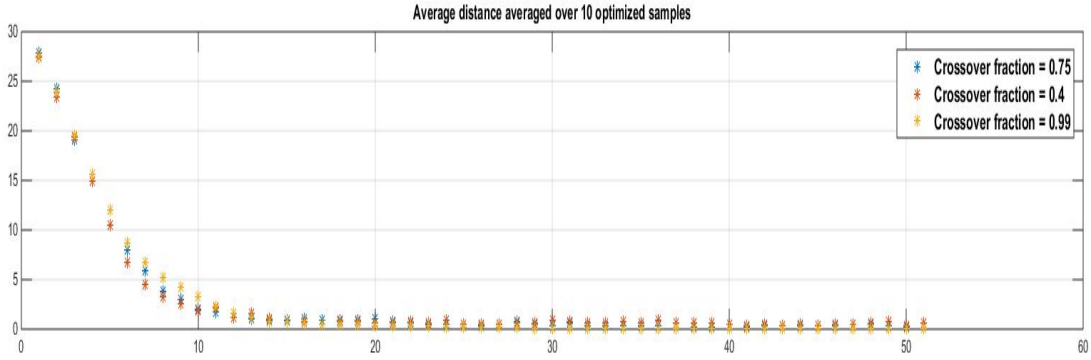

Average distance averaged over 10 optimized samples
Average Distance
Generations
Population size = 50
Population size 100

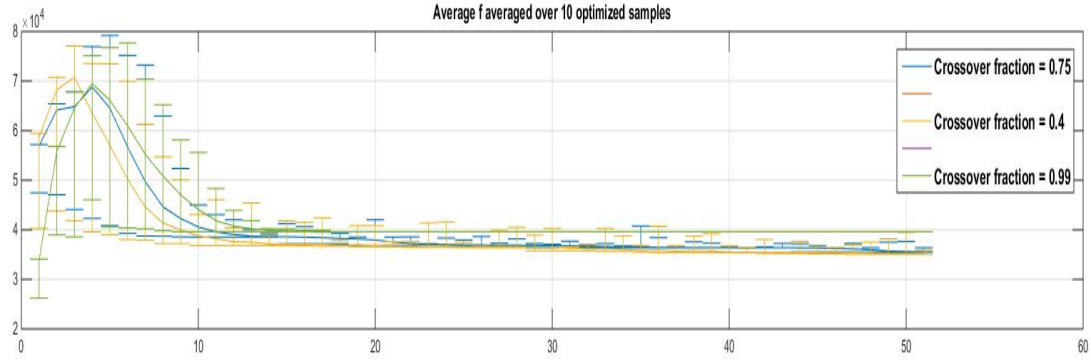

Average f averaged over 10 optimized samples
f
Generations
Population size = 50
Population size 50 Average
Population size = 100
Population size 100 Average Value

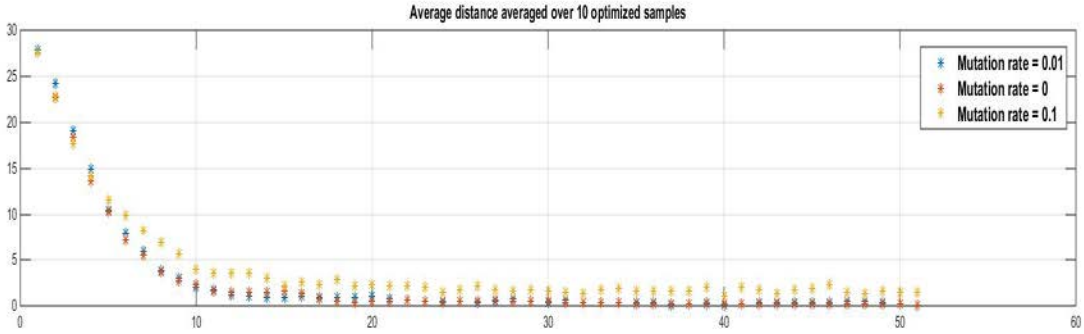
A comparison plot of the 10 samples averaged plots for variation of the elite count, cross over fraction, mutation rate and the population size with the 10 samples averaged baseline case are given below.

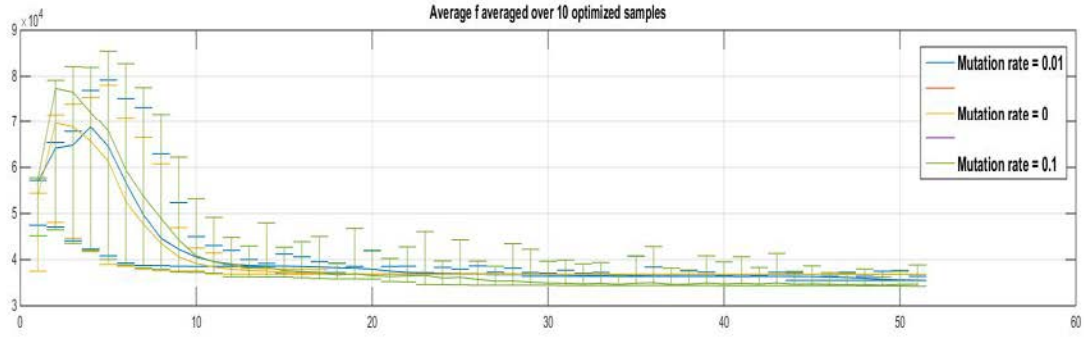


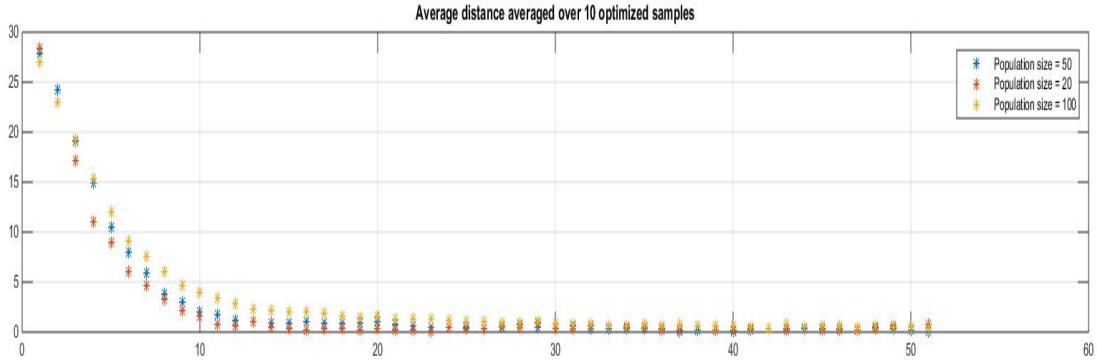


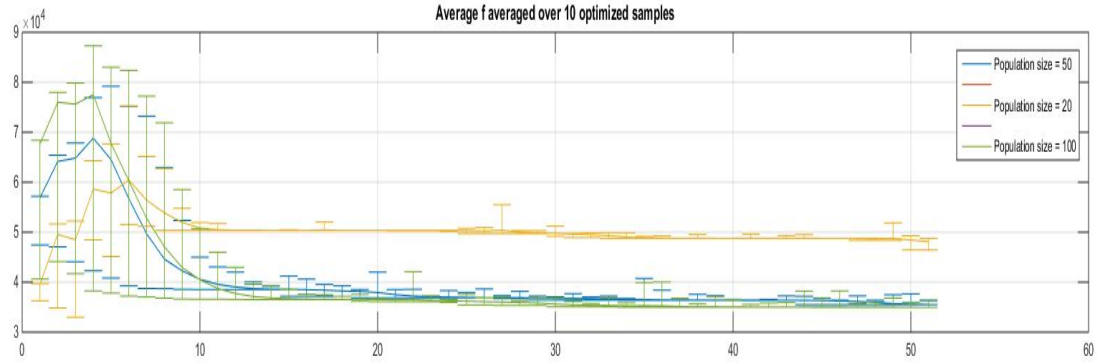





Average distance averaged over 10 optimized samples
Mutation rate = 0.01
Mutation rate = 0
Mutation rate = 0.1


Average f averaged over 10 optimized samples
Mutation rate = 0.01
Mutation rate = 0
Mutation rate = 0.1


Average distance averaged over 10 optimized samples
Population size = 50
Population size = 20
Population size = 100


Average f averaged over 10 optimized samples
Population size = 50
Population size = 20
Population size = 100

Observations on the changed in optimizer behaviour for each paramter:

Elite count variation: Elite count is the positive integer in the MATLAB GA function which specifies the number of individuals in each generation which survive to the next generation. Elitism generally involves simply copying certain number of the best individuals in each generation to the next generation. In the present problem we change the elite count from the baseilne 2 to 0 and then to 10. From the comparitive graphs which are 10 samples averaged graphs we see that the performance is best for the elite count 2 while it decreases for elite count 0 and is lowest for elite count 10. Further convergence of the algorithm in terms of the average distance between individuals is the best when the elite count is 2. This means that copying the best individuals from one generation to the next does not always benefit or harm the optimization algorithm. This is due to the fact that while copying individuals across generations leads to a set back in terms of exploration although it might help us find better the optimal individual. We therefore have to rely on experience to tune this factor in this GA function.

Crossover fraction variation: Cross over fraction is the fraction of the population at the next generation not including the elite children created by the cross over function. This means the number of children at each generation deducted from the population at that generation. From the comparitive graphs which are 10 samples averaged graphs we see that the performance is best for the fration 0.4 while it decreases for fraction 0.75 and is lowest for fraction 0.99. Further convergence of the algorithm in terms of the average distance between individuals is the best when the fraction is 0.99 and is harder as we decrease the fraction. This means that a lower cross over fraction turns out to be better for the performance of this optimizer for this particular design problem. This is probably due to the fact that a lower cross over means there are more children than parents pushed into each next generation. This leads to a better generation of individuals since it is more diversified. Naturally this also means that having a lower fraction proves to be harder to converge to a solution. This also explains the lower error window at the converged solution for a higher cross over fraction optimizer.

Mutation rate variation: Mutation rate is the probability of the event of mutation of the selected entries of an individual. This means that a higher rate will enable higher exploration of the design space but also change the current generation of individuals across each generation. From the comparitive graphs which are 10 samples averaged graphs we see that the performance is best for the rate 0.1 while it decreases for rate 0.01 and is lowest for rate 0. Further convergence of the algorithm in terms of the average distance between individuals is the best when the rate is 0 and is harder as we increase the rate. This means that a higher mutation rate is better for the performance of this optimizer for the given range of rates for this particular design problem. This is probably due to the fact that higher mutation rate means that that each generation has a higher degree of mutated individuals. This leads to a better generation of individuals since it is more diversified. Naturally this also means that having a higher rate proves to be harder to converge to a solution. This also explains the lower error window at the converged solution for a lower mutation rate fraction optimizer.

Population size variation: Population size is the number of individuals specified to exist in each generation. From the comparitive graphs which are 10 samples averaged graphs we see that the performance is best for the size 100 while it decreases for size 50 and is lowest for size 20. Further convergence of the algorithm in terms of the average distance between individuals is the best when the size is 20 and is harder as we increase the size. This means that a larger size is better for the performance of the optimizer. This is true due to the fact that a higher population size certainly implies testing a higher number of individuals w.r.t. lower sizes. This leads to a better exploration of the design space assuming all other parameters remain the same. Naturally this also means that having a higher population makes it harder to converge to a solution. This also explains the lower error window at the converged solution for a lower population size optimizer. From the fact that performance varies significantly when changing the population size we conclude that the optimizer is very sensitive to the choice of the population size.

Task 4

For one run of the KaivalyaBakshi_GA function for this problem we obtain the results:

x_opt = [6.8821 2.1176 40.635 2 21.9784]' inches.

The value of the objective function at x_opt is f(x_opt) = 35918 in^3.

Genetic algorithm scheme:

Selection	Roulette
Elitism	None
Crossover	Single point crossover
Mutation	Uniform mutation

Parameters used in the single run of KaivalyaBakshi_GA:

```
crossoverfraction = 0.75;
mutationrate = 0.01;
populationsize = 50;
generations = 50;
```

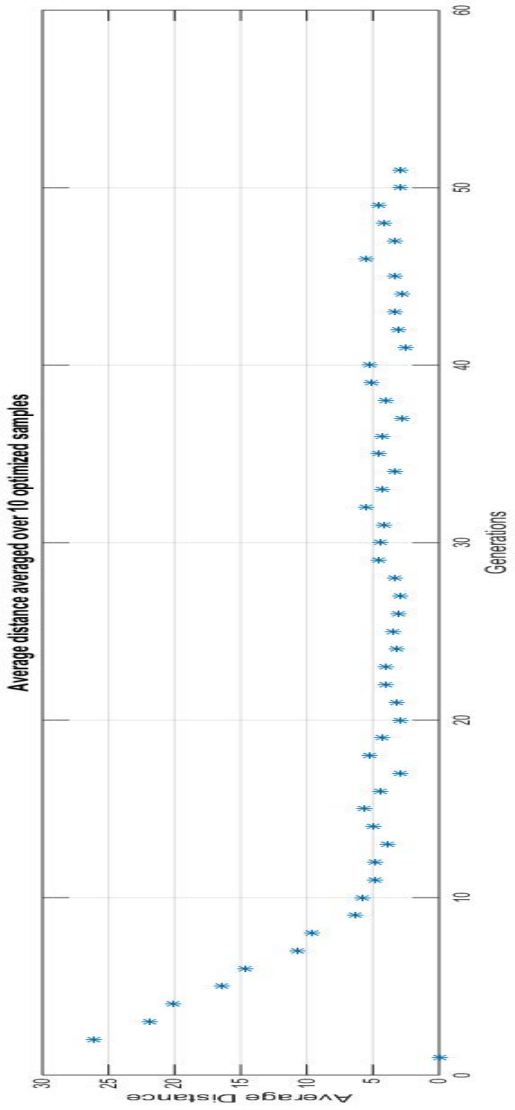
Desciption of constraints handles in the GA constructed:

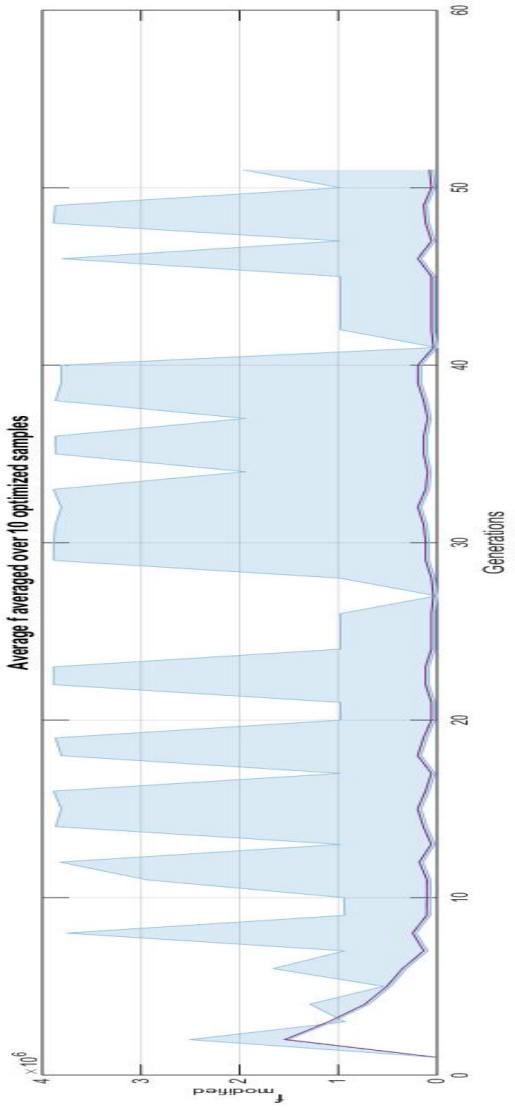
In the task 4 for this project a general genetic algorithm was created for the minimization of an objective function using the scheme detailed above. Therefore the constraints were applied to the minimization problem by penalizing the objective function whenever the constraints were violated. The side constraints as well as all the inequality constraints were applied to the problem in this manner. There are a total of 14 inequality constraints if we combine all the constraints for this problem as can be seen from the problem formulation in task 1.

Let us represent all the constraints including the side and inequality constraints for this problem by the equations $g_k(\vec{x}) \leq 0 \; \forall \; 1 \leq k \leq 14$. The constraints handling strategy used here then implies that we apply the devised genetic algorithm via MATLAB code to a minimize a modified objective function which can be computed as follows:

$$f_{modified}(\vec{x}) = f(\vec{x}) + I_k \times A_m$$

where $I_k = 1$ if $g_k(\vec{x}) > 0$ and $I_k = 0$ otherwise and A_m is the amplifying factor which is taken to be 10^6 for this problem.





Comparison of constructed genetic algorithm with MATLAB GA:

We have plots above of the performance and convergence of the genetic algorithm developed here averaged over 10 samples. We can see from a comparison of the plots given above with the MATLAB GA plots which are plots for the same genetic algorithm parameters with the exception of the lack of elitism in the developed genetic algorithm, that the quality of the MATLAB GA function is significantly better than the one developed here viz. KaivalyaBakshi_GA in terms of convergence.

We see that convergence in terms of the average distance between individuals is much better in the case of MATLAB GA since we obtain almost 0 distance at the end of 50 generations. At the same time we obtain very small upper and lower bound brackets on the objective function values for all the individuals for MATLAB GA as compared with KaivalyaBakshi_GA. The main case of the effects seen here is the way the constraints are handled in KaivalyaBakshi_GA which is a brute force approach hampering the ability of the genetic algorithm to differentiate between individuals. The KaivalyaBakshi_GA uses a factor of 10^6 for penalizing the side constaints which is very high compared to the actual value of the objective function for all the individuals considered. However at present this is necessary since we do not have any other means to account for the constraint equations. This is the reason why the maximal value bounds on the graph for the objective function are very high while the minimal value bounds are very high.

Another factor which affects performance is the lack of an elitism employing scheme in the presently developed version of this genetic algorithm. However this can be remedied given time and very little effort.

A final comment is that KaivalyaBakshi_GA actually gives better performance in terms of the optimal design and minimizing of the objective function. This can be seen by directly comparing the values of these performance criterion noted at the beginning of tasks 1 and 4 respectively. The main issue with KaivalyaBakshi_GA as compared with MATLAB GA is convergence in which the latter fares better due to the reasons noted in this section.

References

[1] AE 8803 GER Optimization for the Design of Engineered Systems Lecture Notes, Brian German, Spring 2015

[2] Numerical Optimization, Nocedal and Wright, Springer 2010

[3] Practical Genetic Algorithms, Haupt, Wiley, 2004