AE4878 - Mission Geometry and Orbit Design
Part 5 - Genetic Algorithm

Ali Nawaz
3ME, LR, Delft University of Technology.

September 2, 2018

1 Introduction

The goal of this assignment is to develop and analyse the performance of a simple binary based "Genetic Algorithm" to obtain the global optima of a Himmelblau function. The Himmelblau function under consideration represented with the aid of Equation 1. Analysing the equation it can be said that the function has a global optima, bearing a function value of $f(x_1, x_2) = 0$. To verify this, plots of the Himmelblau function are illustrated with the aid of Figure1-4. Figures 1 and 2 illustrate the 2 dimensional view of the Himmelblau function, in the function-x_1 plane and function-x_2 plane. While Figure 3 and 4 represent different 3 dimensional views of the Himmelblau function.

$$f(x_1, x_2) = (x_1^2 + x_2 - 13)^2 + (x_1 + x_2^2 - 8)^2 \tag{1}$$

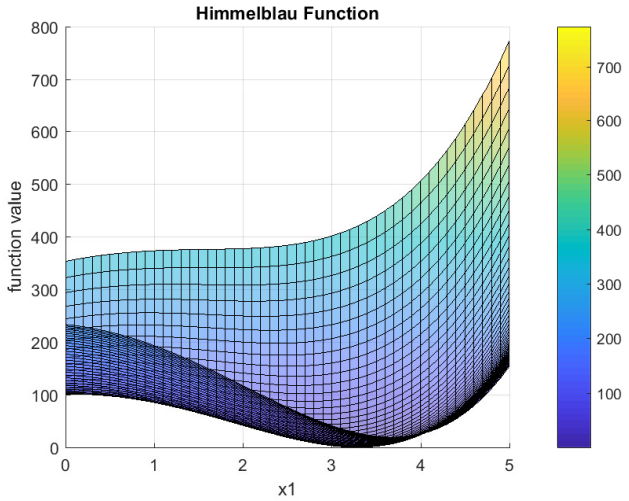

Figure 1: Function value and x_1 axis, 2 dimensional view of Himmelblau function.

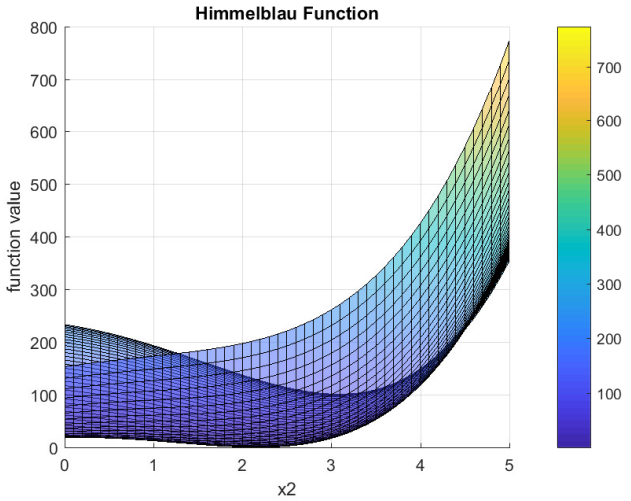

Figure 2: Function value and x_2 axis, 2 dimensional view of the Himmelblau function.

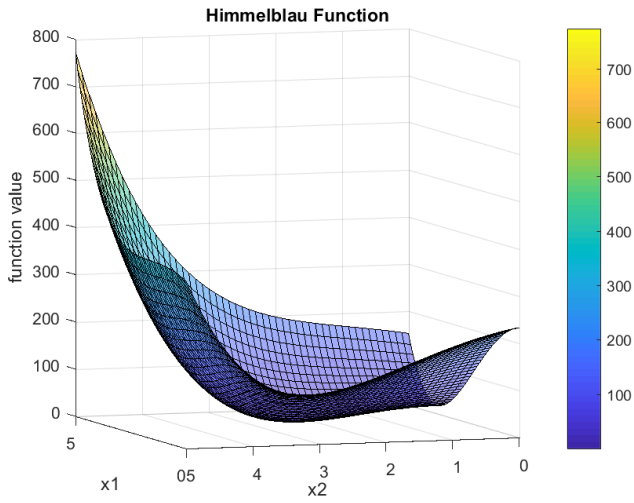

Figure 3: First orientation of 3 dimensional view of Himmelblau function.

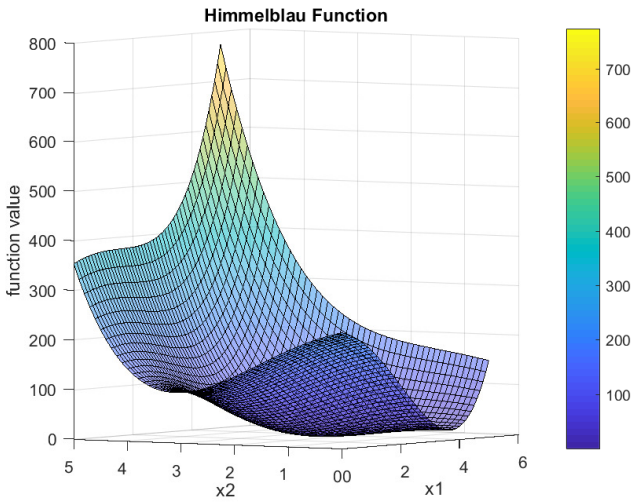

Figure 4: Second orientation of 3 dimensional view of Himmelblau function.

Why Genetic Algorithm? In the previous assignment two different techniques were analysed, 1. grid search (which approaches a problem in a uniform order of parameter increment) 2. Monte Carlo (which approaches a problem in a stochastic order of parameter increment). In an abstract way, Genetic algorithm provides a combination of these utilities i.e. putting order in chaos. Genetic algorithm is a meta-heuristic or combinatorial optimisation process, which can obtain sub-optimal results by searching over a large feasible solution space. Often times the search region is narrowed down by grid-search/Monte Carlo and after convergence of Genetic Algorithm a gradient descent method is used to fine tune the results. However, the objective of this assignment is to focus entirely on Genetic Algorithm.

2 Genetic Algorithm algorithm generation

Figure 5 outlines the flow of genetic algorithm used to find the global optimum of Himmelblau function. This will be used as a reference to guide through the intricate details of the algorithm.

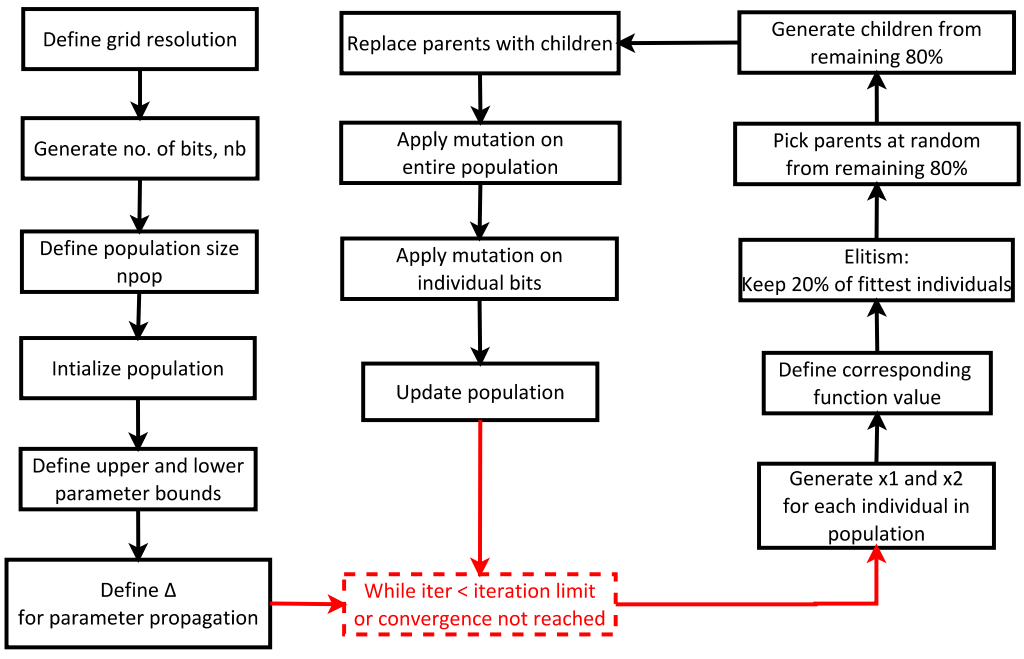

Figure 5: Flowchart outlining the flow of genetic algorithm.

First choice is to select the bit representation depending on the grid points. This is obtained with the aid of Equation 2.[1][Slide 96/120]. UB and LB defines the upper and lower bounds on the parameter value.

$$\Delta = \frac{\text{UB} - \text{LB}}{2^{\text{nb}} - 1}$$
The number of bits(nb) can be obtained as follows: (2)
$$\text{nb} = \frac{\ln(\ 1 + \text{grid points})}{\ln(2)}$$

The number of bits for an individual in the population is used to estimate the population size, with the aid of Equation 3 [1][86/120].

$$\text{npop} = \text{Number of parameters} \times 2^{\text{nb\_avg}} \tag{3}$$

Matlab's built in random integar generator is used to generate population of size npop, which each individual bearing $2 \times$ npop randomly generated bits (0 or 1). Once the population is initialised, Δ parameter is estimated to facilitate parameter value over the constrained work-space. This is outlined with the aid of Equation 4[1][Slides 95,96/120]. Number of bits determines the resolution of search space.

$$x_1 = LB_1 + \Delta_1 \sum_{i=1}^{nb} 2^{nb-i}$$
$$x_2 = LB_2 + \Delta_2 \sum_{i=nb+1}^{2nb} 2^{2nb-i} \tag{4}$$
Where, Δ is given by:
$$\Delta = \frac{\text{UB} - \text{LB}}{2^{\text{nb}} - 1}$$

Recommended convergence criteria in the assignment required a convergence at 0.1% or realistic maximum number of generations achieved. The results seemed to converge within a few iteration steps, and for the given mutation rate even after mutation took place no effect in the optimum result was observed for increasing iterations. Thus the generation size (or iteration limit) of 50 was set. Every generation parameters x_1 and x_2 are estimated with the aid of Equation 4 for every individual in the population. The corresponding function value of stored with the aid of Equation 1. Once the function values and their corresponding values of x_1 and x_2 are stored, 20% of individuals achieving the fittest results are stored for the next generation. The remaining 80% of individuals are picked at random in pairs of two. Two individuals(parents) undergo bit crossover in the last nb bits to form 2 new children. The children replace the parents. For elitism different ratios of population preservation were tried out. Elitism helps preserve the fittest solution. The choice of the percentage of population is strongly coupled with the population size, number of generations and the mutation rates. If the other parameters are fixed and elitism percentage is increased, the optimum value reaches closer to the solution than achieved in the first iteration. Beyond an optimum percentage of elitism, the performance starts to deteriorate since very little flexibility is allowed to program to update it's population. 100% elitism means simple grid search where the grid points are randomly generated and only change when a mutation takes place.

After elitism and children generation, the population of children and elites undergo mutation. Every individual bit and all individuals undergo mutation with 0.1% chance(i.e. 1 in 1000). Mutation results in complementary bits. This helps in avoiding local optima over increasing generations. However the given mutation percentage and with the choice of generation, no significant effect was observed after population/bits encountering mutation.

3 Result Analysis

This section first outlines the results obtained for different grid points. And later focuses on comparing the results and discussing what can be improved. The results obtained for three separate runs for variable grid points are presented with the aid of Table 1.

Table 1: Results obtained for different grid point resolution.

	Run 1					Run 2					Run 3				
Grid Points	**Min. function value**	x_1	x_2	**Run time [s]**	**Iterations**	**Min. function value**	x_1	x_2	**Run time [s]**	**Iterations**	**Min. function value**	x_1	x_2	**Run time [s]**	**Iterations**
32	1.8510	3.0645	2.2581	0.667	1	2.2570	3.0645	2.4194	0.660	3	0.7698	3.3871	2.2581	0.671	1
128	0.2337	3.2677	2.2835	3.360	9	0.1770	3.2283	2.1654	3.340	22	0.1712	3.3465	2.0866	3.309	25
512	0.0069	3.2779	2.1820	46.653	18	0.05547	3.3268	2.1331	45.945	4	0.006917	3.2779	2.1820	46.337	1

Since the population generation is random, sometimes the value obtained for the first generation is the optimum and the iteration converges on first step. However for others the process converges over increasing iterations. Figures 6, 7 and 8 outline the convergence behaviour for longest runs for each grid point value.

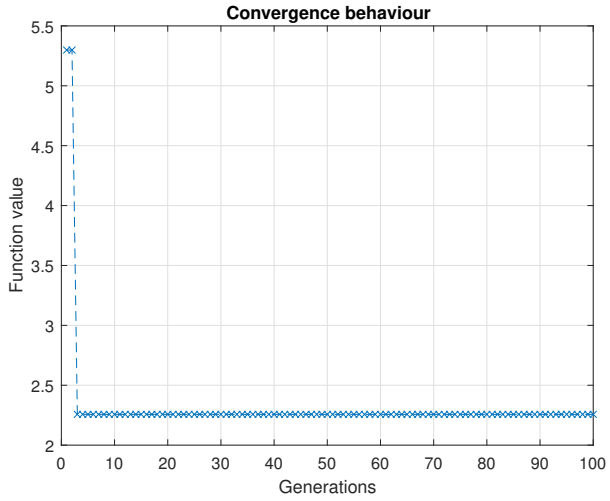

Figure 6: Convergence behaviour for 32 grid points on run 2.

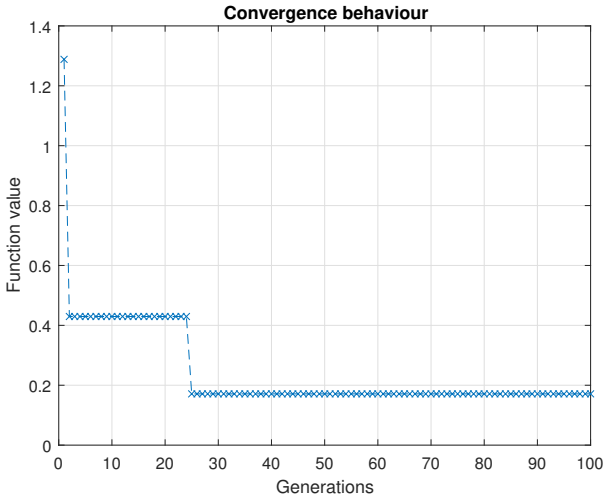

Figure 7: Convergence behaviour for 128 grid points on run 3.

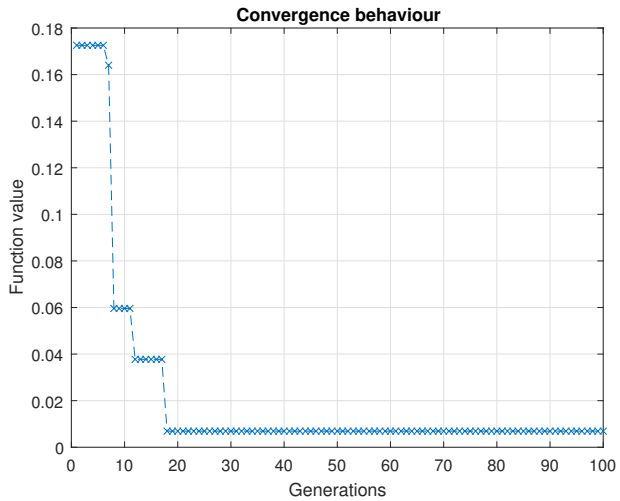

Figure 8: Convergence behaviour for 512 grid points on run 1.

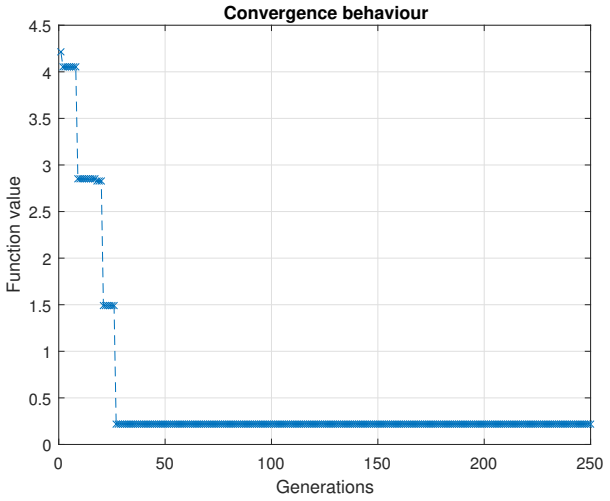

Figure 9: Convergence behaviour for 32 grid points for updated population, elitism and mutation rate.

A common observation is that increasing grid points, and hence number of bits, increases the resolution and hence the function converges closer and closer to the analytical minimum. It is important to note that the analytical minimum is 0, this can be seen from Equation 1 which is a sum of two squares. Another observation is that the parameter values for minimum function value are similar to the nearest integer value for all grid points under consideration. Depending on the resolution the algorithm is capable of converging closer to the analytical solution. If computation time is the priority then 32 grid points return the results in shortest amount of time. If minimum function value has more priority than computation time then 512 grid points perform the best. Looking behind the algorithm, for 512 grid points the population size is greater as per definition. Furthermore, higher population size indicates the possibility of higher total mutations, which helps avoid convergence to a lo-

cal minima. The algorithm can be improved by increasing the population size, decreasing the elitism percentage and increasing the mutation rate. This is illustrated for the case of 32 grid points with the aid of Figure 9. The population size is increased to twice the previous value, elitism is decreased to 10%(lowering this value means more random children generation) and the mutation rate is set to 10%. All of these choices result in more random random point generation, which helps avoid the local minima (if any) or converge prematurely.

4 Matlab Scipt

Following script is used for Genetic Algorithm.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% AE4878 Mission Geometry and Orbit Design %%%%%%%%%%%
%%%%% Assignment 5 - Week 3.6 - OPTIMIZATION - Version 3%%
%%%%% Author Info: Ali Nawaz; Student ID - 4276477 %%%%%%%
%%%%% Assigned tasks : OPTIM_8, GENETIC ALGORITHM %%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Genetic algorithm
% Plotting the Himmelblau function
[x1,x2] = meshgrid(0:0.1:5,0:0.1:5); % generating mesh
f = ( (x1.*x1 + x2) - 13).^(2) + ( (x1 + x2.*x2) -8).^(2); % Himmelblau function

figure(1)
surf(x1, x2, f,'FaceAlpha',0.5);
title('Himmelblau Function');
xlabel('x1')
ylabel('x2')
zlabel('function value')
colorbar

% Defining number of bits for each parameter
% grid_points = 32;
% grid_points = 128;
grid_points = 512;

% Number of bits for given grid points
nb = round(log(1 + grid_points)/log(2)); % round off bits to nearest integar

nparam = 2; % no. of parameters
npop = nparam*2^nb; % population size

% Randomly generated bits
bit1 = randi([0 1],npop,nb); % Generating bit sequence for parameter x1
bit2 = randi([0 1],npop,nb); % Generating bit sequence for parameter x2
bit = [bit1,bit2];

% Upper and lower bounds for parameter 1 and 2
lb1 = 0;
ub1 = 5;

lb2 = 0;
ub2 = 5;

% Delta parameter for step generation
Δ1 = (ub1-lb1)/( 2^(nb) - 1);
Δ2 = (ub2-lb2)/( 2^(nb) - 1);

% x1 = [];
% x2 = [];
x = []; % parameters x1, x2
func =[]; % function value
resf = []; % store best function value for all iterations
resx1 = []; % corresponding value of x1
resx2= []; % corresponding value of x2
run = 1;

limit = 100; % Max number of generations
iter = 0; % intialize iteration
while iter <  limit;
iter = iter + 1 ;
    % Scanning through each individual in the population
    for p = 1:length(bit(:,1))
```

```
        sum1 = [];
        sum2 = [];
        for b=1:length(bit(p,:))
            if b ≤ length(bit(p,:))/2
                sum1 = [sum1, bit(p,b)*2^(nb-b)];
            else
                sum2 = [sum2, bit(p,b)*2^(2*nb-b)];
            end
        end
        x1_buf = lb1 + Δ1*sum(sum1); % generating x1 parameter
        x2_buf = lb2 + Δ2*sum(sum2); % generating x2 parameter
        f_buf = ( (x1_buf.*x1_buf + x2_buf) - 13).^(2) + ( (x1_buf + x2_buf.*x2_buf) ...
            -8).^(2); % Himmelblau function value
        func = [func;f_buf]; % store the function value
        x = [x; [x1_buf,x2_buf]]; % store corresponding paramter value
    end
    % best results
    resf(iter, run) = [unique(min(func))]; % best function value
    opti_x = x(find( func == min(func),1),:);
    resx1(iter, run) = opti_x(1); % corresponding x1 value
    resx2(iter, run) = opti_x(2); % corresponding x2 value

    % Elitism, keeping 20% of best population
    f_sort = sort(func); % sort in terms of min to max function value
    f_keep = f_sort(1:round(npop*0.2)); % keep lowest 20%

    index_elite = []; % index elite population
    for j = 1:length(f_keep)
        index_elite = [index_elite, [find(func == f_keep(j))]'];
    end
    index_elite = sort(unique(index_elite)); % index elite population
%         pop_elite = bit(index_elite,:); % bit representation of elite population

    index_buf  = 1:npop; % index
    index_nelite = setdiff(index_buf, index_elite); % index non-elite population

    % Generation selection among non elite population
    % Replacing parent by children
    for g = 1:2:length(index_nelite)
        if g ≠ length(index_nelite)
            bit_extract1 = bit(index_nelite(g),nb+1:end);
            bit_extract2 = bit(index_nelite(g+1),nb+1:end);
            % bit crossover for children generation
            bit(index_nelite(g),nb+1:end) = bit_extract2;
            bit(index_nelite(g+1),nb+1:end) = bit_extract1;
        else
            break
        end
    end

    % Mutation, looping over all parents and all bits

    for r = 1:length(bit(:,1))
        % Mutation of population
        mr = randi([0 1000],1); % 0.1 percent = 1 in 1000
        if mr == 1
            for c = 1:length(bit(r,:))
                % Mutation of bits
                mc = randi([0 1000],1); % 0.1 percent = 1 in 1000
                if mc == 1
                    bit(r,c) = ¬bit(r,c);
                    disp('Mutation encountered');
                else
                    continue
                end
            end
        else
            continue
        end
    end
end
% Plot the convergence behaviour
figure(2)
plot(1:length(resf),resf,'--x');
title('Convergence behaviour');
xlabel('Generations');
```

```
137 ylabel('Function value');
138 grid on;
```

References

[1] R. Noomen. *AE4878 Mission Geometry and Orbit Design, Optimisation v4-16*. TU Delft, 2017.