CSC 467 2.0 EVOLUTIONARY COMPUTING ASSIGNMENT 2

PRESENTED BY

P.H.S.M.Gunarathna – AS2019360
H.M.T Herath – AS2019369
K.G.S.Manimekala – AS2019445
G.B.C.Prabhashwara – AS2019503
G.M.R.Silva – AS2019542
I.U.Wickramasinghe - AS2019576

2023.08.04

Agenda

1. Problem
2. Algorithm
3. Implementation
4. Representation
5. Results
6. Findings

PROBLEM

Introduction to the problem

- Implement DE(Differential Evaluation) to solve Booth function (The function is defined on 2-dimensional space.

- Obtain the following results,

(a) The best solution in the population for each run and its fitness.

(b) Best fitness values against the iteration for a particular run. (Take a random run you prefer. Include a graph as well)

(c) Best fitness values against the runs.

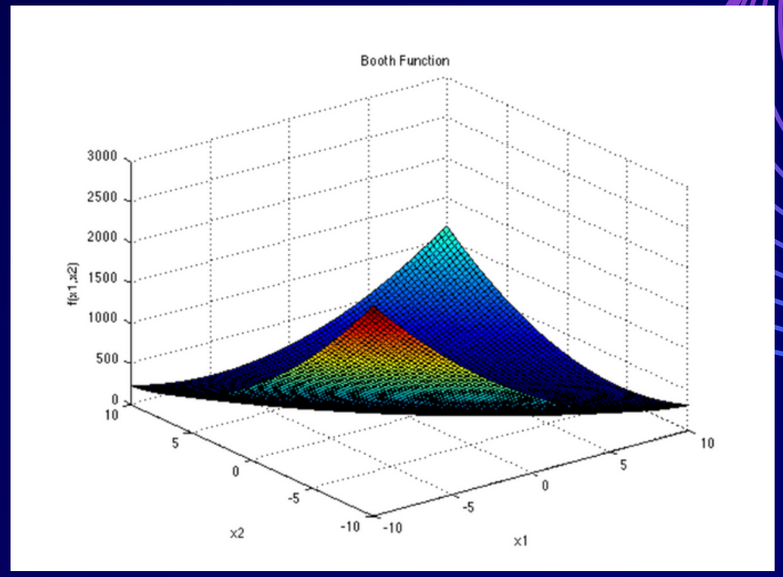
(d) Running time against the runs.

Introduction to the Booth function

- The Booth Function:

  The Booth function is a 2-dimensional ,continous mathematical function.
- It is defined as:

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$



Properties of the Booth Function

2-Dimensional Space:

- The Booth function operates on a 2-dimensional input space with variables x_1 and x_2.

Global Minimum:

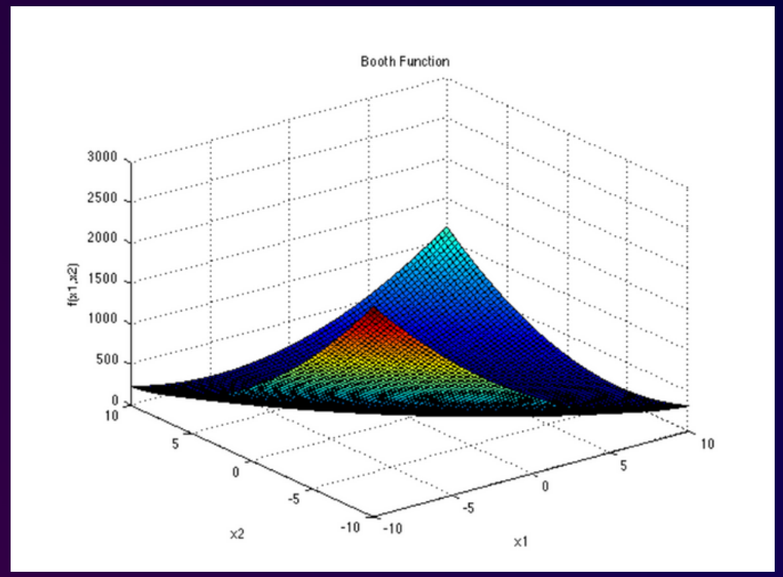
- Global Minimum:
  - The function possesses a single global minimum located at x = (1, 3).
  - The global minimum value is f(x) = 0.

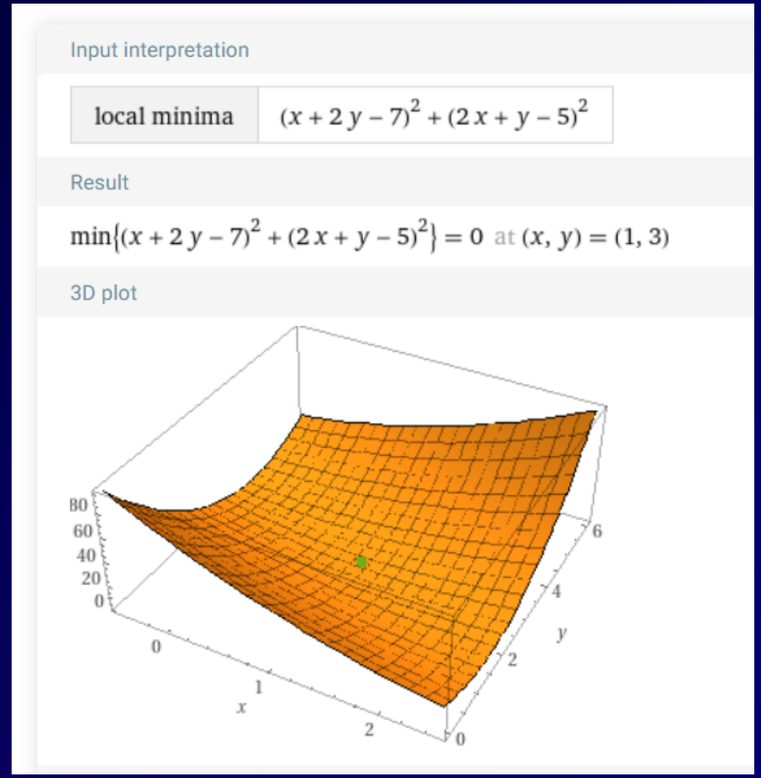
Unbounded:

- The Booth function is unbounded, meaning it continues to infinity as the input values move away from the global minimum point.

Graphical Representation

- Display a 3D plot of the Booth function to visually showcase its landscape
- The function has only one global minimum and no local minimum.





Important facts

Benchmark Function

- The Booth function serves as a standard test case to evaluate the performance of optimization algorithms.
- Algorithms need to efficiently find the global minimum amidst local minima.

Real-world Applications

- While simple, the Booth function represents challenges found in real-world optimization problems with multiple optima.

Continued...

- The Booth function provides a challenging optimization problem with a single global minimum amidst multiple local minima.
- DE presents a promising approach to efficiently finding the global minimum of the Booth function.
- Successful optimization of the Booth function serves as a stepping stone for applying DE to more complex real-world optimization tasks.

ALGORITHM

What is Differential Evolution (DE)?

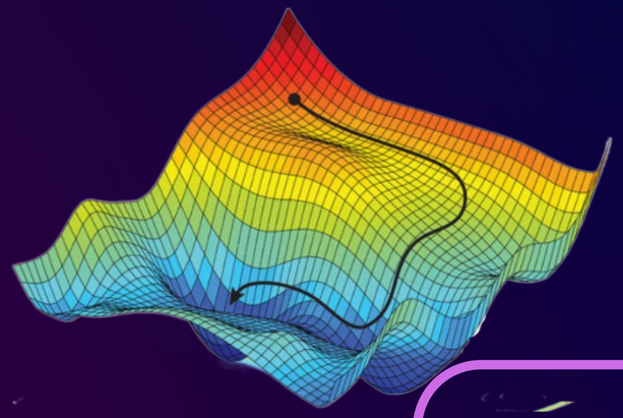
- Introduced by Storn and Price during 1995/1996
- Developed to optimized real parameters, real value functions.
- Initially it was implemented, to cope up with continous domain problems.

Characteristics of Differential Evolution (DE)?

- DE is a population-based evolutionary algorithm:
- Population of candidate solutions evolves over generations.
- Utilizes mutation, crossover, and selection operators to explore and exploit the search space.
- Differential Evolution has been successfully applied to a wide range of optimization tasks

Why use Differential Evolution Algorithm

- **Continuous optimization:** DE is designed for continuous optimization problem
- **Simplicity and ease of implementation:** Relatively easy to understand and implement
- **Fewer function evaluations:** DE typically requires fewer function evaluations compared to traditional optimization methods
- **Versatility:** DE can be adapted to handle various types of optimization problems, including single-objective, multi-objective, constrained, and noisy optimization.



Steps of the DE Algorithm

Target Vector

Mutation

Donor Vector

Crossover & Recombination

Trial Vector

Selection

New Polulation

Initialization

- Defining the Chromosomes and the Population (Set of Solutions)
- The solution which is undergoing evolution is the "**Target Vector**"

Mutation

- Simply means adding the weighted difference vector between two solutions to a third solution.

$$v = a + F(b - c)$$

v -Donor Vector
F - Mutation Factor
a,b,c - Solutions

Crossover

- Combine the mutant vector with the target vector to create a trial vector.
- Increase the Diversity of the Population.

$$u_{j,i} = \begin{cases} v_{j,i} & \text{if} \quad r_{j,i} \leq CR \quad \text{or} \quad j = I_{rand} \\ x_{j,i} & \text{if} \quad r_{j,i} > CR \quad \text{and} \quad j \neq I_{rand} \end{cases}$$

Ui - Trial Vector
Xi - Target Vector
Vi - Donor Vector
ri,j – Random Number
CR – Crossover Probability
I - Random Variable Location

- If the random number is greater than the crossover probability and the jth vector location is not equal to the randomly selected variable location, we choose Target Vector as the Trial Vector. Choose Donor Vector otherwise.

Selection

- The Target Vector is compared with the trial vector and the one with the lowest function value is selected for the new population.

$$x_i = \begin{cases} u_i & if \quad f(u_i) \leq f(x_i) \\ x_i & Otherwise \end{cases}$$

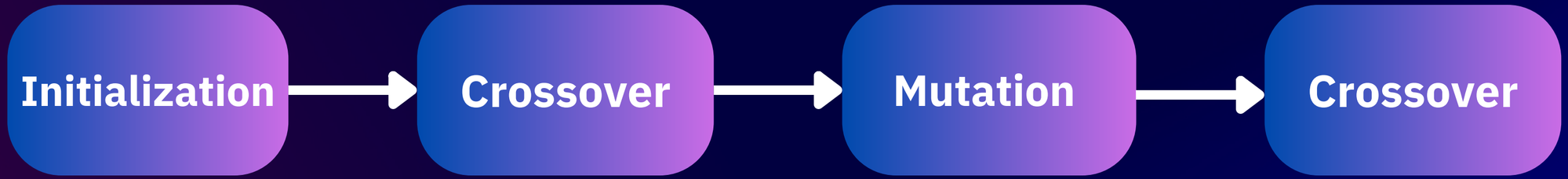
- Eg:- Evaluate the fitness of the trial vector using the Booth function, compare it with the target vector and keep the fitter one.

Termination

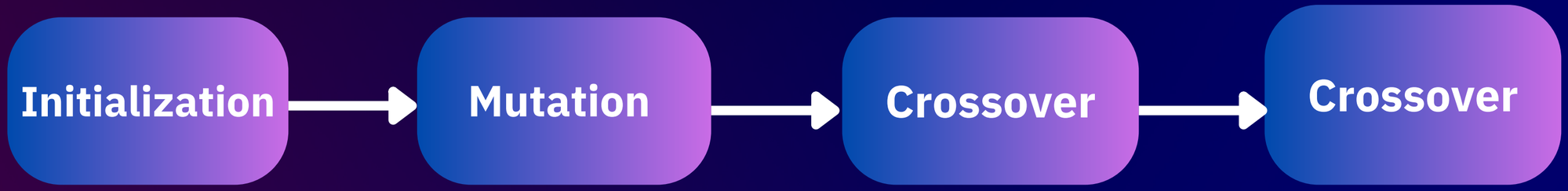
- Stop the algorithm when the desired fitness level is obtained.

Difference Between GA and DEA

Genetic Algorithm



Differential Evolution Algorithm



IMPLEMENTATION

General Implementation of the DE

1: Begin;
2: Generate random population of n individuals
3: Define the objective function $f(X)$
4: Determine the fitness I_i of i^{th} individual X_i via $f(X_i)$

5: **while** End condition **do**
6: **for** Each individual i **do**
7: Choose 3 numbers a, b and c, that is $1 \leq a, b, c \leq n$ and $a \neq b \neq c \neq i$ [**Mutation**]
8: Compute the agent's potentially new position according to `equation 2` and `equation 3`. [**Crossover & Selection**]
9: **end for**
10: **end while**
11: Pick the agent from the population that has the highest fitness or lowest cost and return it as the best found candidate solution.
12: End

REPRESENTATION

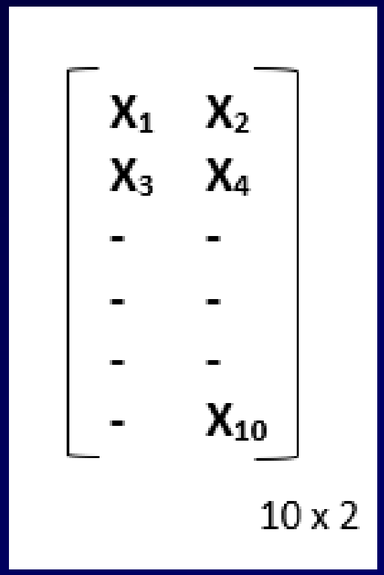
Booth Function Representation

- The Booth Function, f(x) = (x1 + 2*x2 - 7)^2 + (2*x1 + x2 - 5)^2 is defined on a 2-dimentional space and the function has a global minimum.
- The Solution of the global minimum comes in this format - (x1, x2)
- As such, we employed a real-value representation to generate the solutions.

Since the Initial population size is 10, initial Chromosomes are a matrix of 10 by 2 as below,

STEP 1

Defining the Population and the Population Size.



STEP 2

Initial Population (xi) → **Mutation** → **Donor Matrix (vi)**

$$\begin{bmatrix} X_1 & X_2 \\ X_3 & X_4 \\ - & - \\ - & - \\ - & - \\ - & X_{10} \end{bmatrix}$$

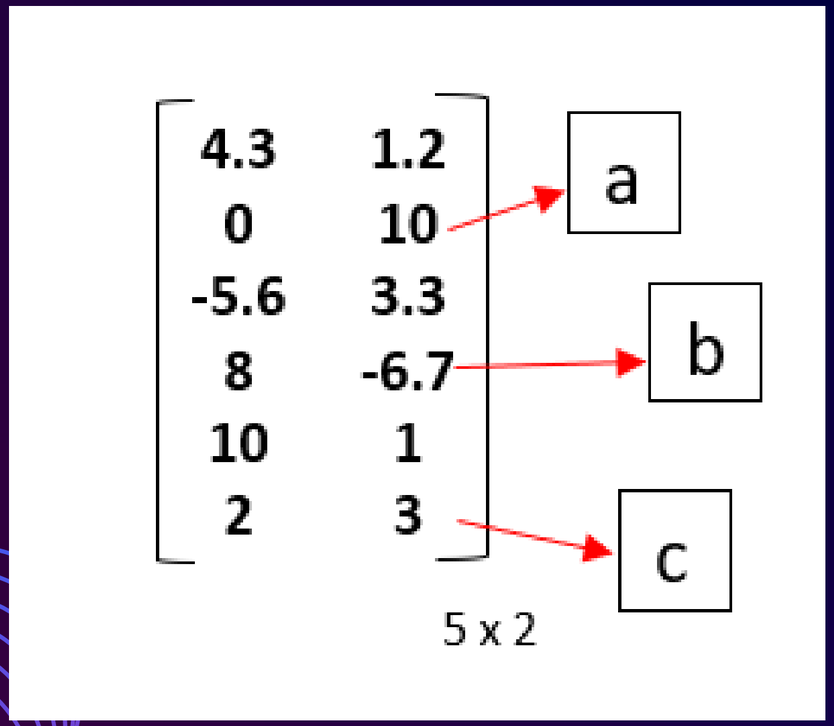
10 x 2

$$\begin{bmatrix} V_1 & V_2 \\ V_3 & V_4 \\ - & - \\ & \\ - & - \\ - & V_{10} \end{bmatrix}$$

10 x 2

Let N=5 F = 0.5

$$v = a + F(b - c)$$



Initial Population (xi)

For i=1: population size (N):

(v1,v2) = (o, 10) + 0.5*((8, -6.7) - (2, 3)

(v1,v2) = (3, 5.15)

- a, b, c vectors are chosen randomly from initial population.(randomly choose 3 indexes from initial population which is not current index.
- We do this step N times to generate N x 2 donor matrix.
- In this case N = 5.

STEP 3

Donor Matrix (vi) + Target Matrix (xi) → **Crossover** → **Trial Matrix (ui)**

$$\begin{bmatrix} V_1 & V_2 \\ V_3 & V_4 \\ - & - \\ & \\ - & - \\ - & V_{10} \end{bmatrix}$$

10 x 2

$$\begin{bmatrix} X_1 & X_2 \\ X_3 & X_4 \\ - & - \\ - & - \\ - & - \\ - & X_{10} \end{bmatrix}$$

10 x 2

$$\begin{bmatrix} U_1 & U_2 \\ U_3 & U_4 \\ - & - \\ & \\ - & - \\ - & U_{10} \end{bmatrix}$$

10 x 2

$$u_{j,i} = \begin{cases} v_{j,i} & if \quad r_{j,i} \leq CR \quad or \quad j = I_{rand} \\ x_{j,i} & if \quad r_{j,i} > CR \quad and \quad j \neq I_{rand} \end{cases}$$

1

2

1

2

3	5.15
-2.9	7.1
2.3	-5
8	-6.7
0.15	1
2	8.91

5 x 2

Donor (vi)

4.3	1.2
0	10
-5.6	3.3
8	-6.7
10	1
2	3

5 x 2

Initial (xi)

U_1	U_2
U_3	U_4
-	-
-	-
-	U_{10}

10 x 2

Trial (ui)

STEP 4

Trial Matrix (ui) + Target Matrix (xi) → **Selection** → **New Population**

$$\begin{bmatrix} U_1 & U_2 \\ U_3 & U_4 \\ - & - \\ & \\ - & - \\ - & U_{10} \end{bmatrix}$$

10 x 2

$$\begin{bmatrix} X_1 & X_2 \\ X_3 & X_4 \\ - & - \\ - & - \\ - & - \\ - & X_{10} \end{bmatrix}$$

10 x 2

$$\begin{bmatrix} S_1 & S_2 \\ S_3 & S_4 \\ - & - \\ & \\ - & - \\ - & S_{10} \end{bmatrix}$$

10 x 2

$$x_i = \begin{cases} u_i & if \quad f(u_i) \le f(x_i) \\ x_i & Otherwise \end{cases}$$

1

2

f(Xi)

$$\begin{bmatrix} X_1 & X_2 \\ X_3 & X_4 \\ - & - \\ - & - \\ - & - \\ - & X_{10} \end{bmatrix}$$

10 x 2

if 2

f(Ui)

$$\begin{bmatrix} U_1 & U_2 \\ U_3 & U_4 \\ - & - \\ - & - \\ - & - \\ - & U_{10} \end{bmatrix}$$

10 x 2

if 1

$$\begin{bmatrix} S_1 & S_2 \\ S_3 & S_4 \\ - & - \\ - & - \\ - & - \\ - & S_{10} \end{bmatrix}$$

10 x 2

New Population

Code Implementation

1.Variable, Constant Declaration

```
function [] = main_func()
 clc;
 clear;

 runs_count = 1; % no.of runs
 iterations = 300; % no.of iterations
 n = 10; % population size
 f = 0.5; % mutation factor
 cr = 0.1; % recombination probability
 run_details = zeros(runs_count,3);

   % Loop for running the optimization algorithm multiple times
   for run = 1:runs_count
     best_val = zeros(iterations,3);
```

2 . For loop for running the algorithm for each iteration

```
% Start measuring the overall algorithm's running time
overall_tic = tic;

chromosomes = init_population(n);

% Loop for running the DE algorithm for each iteration
for itr = 1:iterations
  donor = mutate(chromosomes,f);
  trial = crossover(chromosomes,donor,cr);
  [chromosomes, mse_val(itr)] = select(chromosomes, trial);
  [best_val(itr,1),best_val(itr,2), best_val(itr,3)] = evaluate(chromosomes);
end

% Stop the overall timer and calculate the elapsed time
overall_elapsed_time = toc(overall_tic);
```

3. Creating Initial Population

```
% Create initial population
function [chromosomes] = init_population(n)
  % Iterates over the population size n
  for i=1:n
    % Generate random variables x1 & x2 of the ith chromosome
    % Generate random variables x1 & x2 of the ith chromosome
    lower_bound = -10;
    upper_bound = 10;
    chromosomes(i,:) = lower_bound + (upper_bound - lower_bound) * rand(1, 2);
  end
end
```

4. Performing Mutation Operation

```
% Perform mutation operation in the DE algorithm
function [donor] = mutate(chromosomes, f)
  % Initialize donor vector with the same size as the initial population matrix
  donor = zeros(n, 2);

  for i = 1:n
    % Loop until we find distinct random values a, b, and c, and they are different from i
    a = randi([1, n], 1, 1);
    b = randi([1, n], 1, 1);
    c = randi([1, n], 1, 1);
    while (a == b || b == c || a == c || a == i || b == i || c == i)
      a = randi([1, n], 1, 1);
      b = randi([1, n], 1, 1);
      c = randi([1, n], 1, 1);
    end

    % Mutation step
    donor(i, :) = chromosomes(a, :) + f * (chromosomes(b, :) - chromosomes(c, :));
  end
end
```

$1 \leq a, b, c \leq n$

$a \neq b \neq c \neq i$

4. Performing Mutation Operation

```
% Perform mutation operation in the DE algorithm
function [donor] = mutate(chromosomes, f)
  % Initialize donor vector with the same size as the initial population matrix
  donor = zeros(n, 2);

  for i = 1:n
    % Loop until we find distinct random values a, b, and c, and they are different from i
    a = randi([1, n], 1, 1);
    b = randi([1, n], 1, 1);
    c = randi([1, n], 1, 1);
    while (a == b || b == c || a == c || a == i || b == i || c == i)
      a = randi([1, n], 1, 1);
      b = randi([1, n], 1, 1);
      c = randi([1, n], 1, 1);
    end

    % Mutation step
    donor(i, :) = chromosomes(a, :) + f * (chromosomes(b, :) - chromosomes(c, :));
  end
end
```

$$v = a + F(b - c)$$

5. Performing Crossover Operation

```
% perform the crossover operation
function [trial] = crossover(target, donor, cr)
  trial = zeros(n, 2);
  % Generating crossover points using binomial crossover
  for i=1:n
    Irand = randi([1, 2], 1, 1);
    for j=1:2
      val = randn;
      if(val>cr && j~=Irand)
      trial(i,j) = target(i,j);
      else
      trial(i,j) = donor(i,j);
      end
    end
  end
end
```

Initial Vector
Donor Vector
Crossover probability

```
% perform the crossover operation
function [trial] = crossover(target, donor, cr)
  trial = zeros(n, 2);
  % Generating crossover points using binomial crossover
  for i=1:n
    Irand = randi([1, 2], 1, 1);
    for j=1:2
      val = randn;
      if(val>cr && j~=Irand)
      trial(i,j) = target(i,j);
      else
      trial(i,j) = donor(i,j);
      end
    end
  end
end
```

```
% perform the crossover operation
function [trial] = crossover(target, donor, cr)
  trial = zeros(n, 2);
  % Generating crossover points using binomial crossover
  for i=1:n
    Irand = randi([1, 2], 1, 1);
    for j=1:2
      val = randn;
      if(val>cr && j~=Irand)
      trial(i,j) = target(i,j);
      else
      trial(i,j) = donor(i,j);
      end
    end
  end
end
```

```
% perform the crossover operation
function [trial] = crossover(target, donor, cr)
  trial = zeros(n, 2);
  % Generating crossover points using binomial crossover
  for i=1:n
    Irand = randi([1, 2], 1, 1);
    for j=1:2
      val = randn;
      if(val>cr && j~=Irand)
      trial(i,j) = target(i,j);
      else
      trial(i,j) = donor(i,j);
      end
    end
  end
end
```

$$u_{j,i} = \begin{cases} v_{j,i} & if \quad r_{j,i} \leq CR \quad or \quad j = I_{rand} \\ x_{j,i} & if \quad r_{j,i} > CR \quad and \quad j \neq I_{rand} \end{cases}$$

6. Selection of New Population

```
% Selection of new population
function [new_chromosomes, mse] = select(target, trial)
  new_chromosomes = zeros(n, 2);
  for i=1:n
    target_fitness = (target(i,1) + 2*target(i,2) - 7)^2+(2*target(i,1) +
    target(i,2) - 5)^2;
    trial_fitness = (trial(i,1) + 2*trial(i,2) - 7)^2+(2*trial(i,1) +
    trial(i,2) - 5)^2;
    % Since minimization problem variables with minimum function value is selected
    if(target_fitness < trial_fitness )
    new_chromosomes(i,1)=target(i,1);
    new_chromosomes(i,2)=target(i,2);
    else
    new_chromosomes(i,1)=trial(i,1);
    new_chromosomes(i,2)=trial(i,2);
    end
  end
end
```

6. Selection of New Population

```
% Selection of new population
function [new_chromosomes, mse] = select(target, trial)
  new_chromosomes = zeros(n, 2);
  for i=1:n
    target_fitness = (target(i,1) + 2*target(i,2) - 7)^2+(2*target(i,1) +
    target(i,2) - 5)^2;
    trial_fitness = (trial(i,1) + 2*trial(i,2) - 7)^2+(2*trial(i,1) +
    trial(i,2) - 5)^2;
    % Since minimization problem variables with minimum function value is selected
    if(target_fitness < trial_fitness )
    new_chromosomes(i,1)=target(i,1);
    new_chromosomes(i,2)=target(i,2);
    else
    new_chromosomes(i,1)=trial(i,1);
    new_chromosomes(i,2)=trial(i,2);
    end
  end
end
```

calculating target and trial fitnesses for each population

7 .Finding the Best Fit for each Iteration

```
% Find the best fit out of all chromosomes
function [op_val, x1, x2] = evaluate(ch)
  % Assuming first chromosome as the best fit
  op_val = (ch(1,1) + 2*ch(1,2) - 7)^2+(2*ch(1,1) + ch(1,2) - 5)^2;
  x1 = ch(1,1);
  x2 = ch(1,2);
  for i=2:n
    temp = (ch(i,1) + 2*ch(i,2) - 7)^2+(2*ch(i,1) + ch(i,2) - 5)^2;
    if( temp<=op_val )
      op_val = temp;
      x1 = ch(i,1);
      x2 = ch(i,2);
    end
  end
end
```

Takes a population matrix as an input

```
% Find the best fit out of all chromosomes
function [op_val, x1, x2] = evaluate(ch)
  % Assuming first chromosome as the best fit
  op_val = (ch(1,1) + 2*ch(1,2) - 7)^2+(2*ch(1,1) + ch(1,2) - 5)^2;
  x1 = ch(1,1);
  x2 = ch(1,2);
  for i=2:n
    temp = (ch(i,1) + 2*ch(i,2) - 7)^2+(2*ch(i,1) + ch(i,2) - 5)^2;
    if( temp<=op_val )
      op_val = temp;
      x1 = ch(i,1);
      x2 = ch(i,2);
    end
  end
end
```

8.Finding Best Fit for the particular Run

```
function [op_val, x1, x2] = best_at_run(ch)
  op_val = ch(1,1);
  x1 = ch(1,2);
  x2 = ch(1,3);
  for it=2:iterations
    if(ch(it,1)<=op_val)
      op_val = ch(it,1);
      x1 = ch(it,2);
      x2 = ch(it,3);
    end
  end
end
```

8.Finding Best Fit for the particular Run

takes the details of each iteration

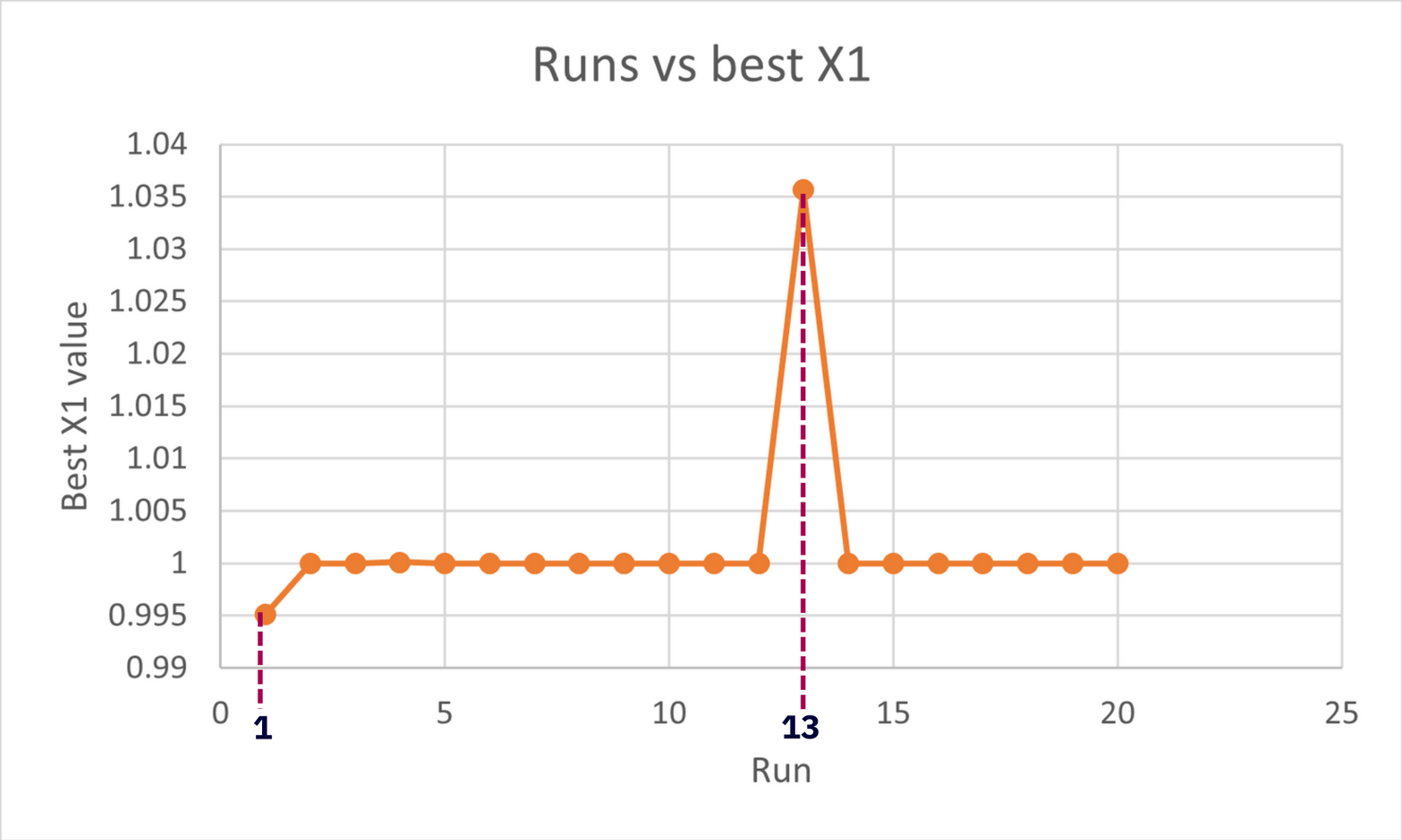
```
function [op_val, x1, x2] = best_at_run(ch)
  op_val = ch(1,1);
  x1 = ch(1,2);
  x2 = ch(1,3);
  for it=2:iterations
    if(ch(it,1)<=op_val)
      op_val = ch(it,1);
      x1 = ch(it,2);
      x2 = ch(it,3);
    end
  end
end
```

RESULTS

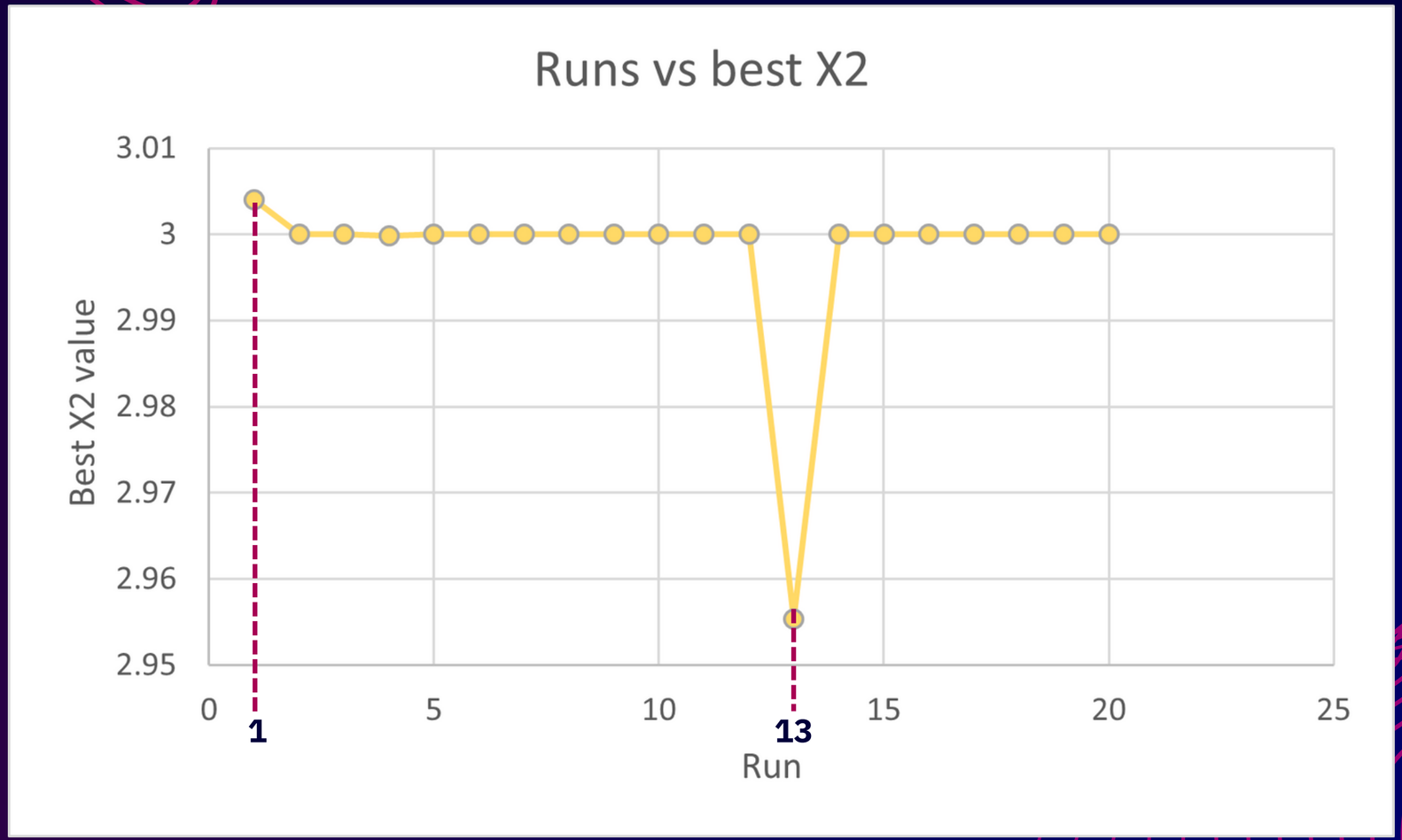
		Best_solution		
Run_no	Fitness_of_best_value	X1	X2	runtime
1	4.39E-05	0.99506	3.004	2.4423
2	8.08E-16	1	3	2.6489
3	1.48E-13	1	3	2.5567
4	4.16E-08	1.0001	2.9998	2.589
5	0	1	3	2.4955
6	0	1	3	2.4912
7	0	1	3	2.6376
8	0	1	3	2.6409
9	0	1	3	2.5062
10	0	1	3	2.4748
11	0	1	3	2.5173
12	0	1	3	2.5602
13	0.003581	1.0357	2.9554	2.5191
14	0	1	3	2.456
15	0	1	3	2.5302
16	0	1	3	2.5338
17	0	1	3	3.005
18	0	1	3	2.7429
19	0	1	3	2.9309
20	0	1	3	3.1321

The Best solution in the population, its fitness and running time for each run

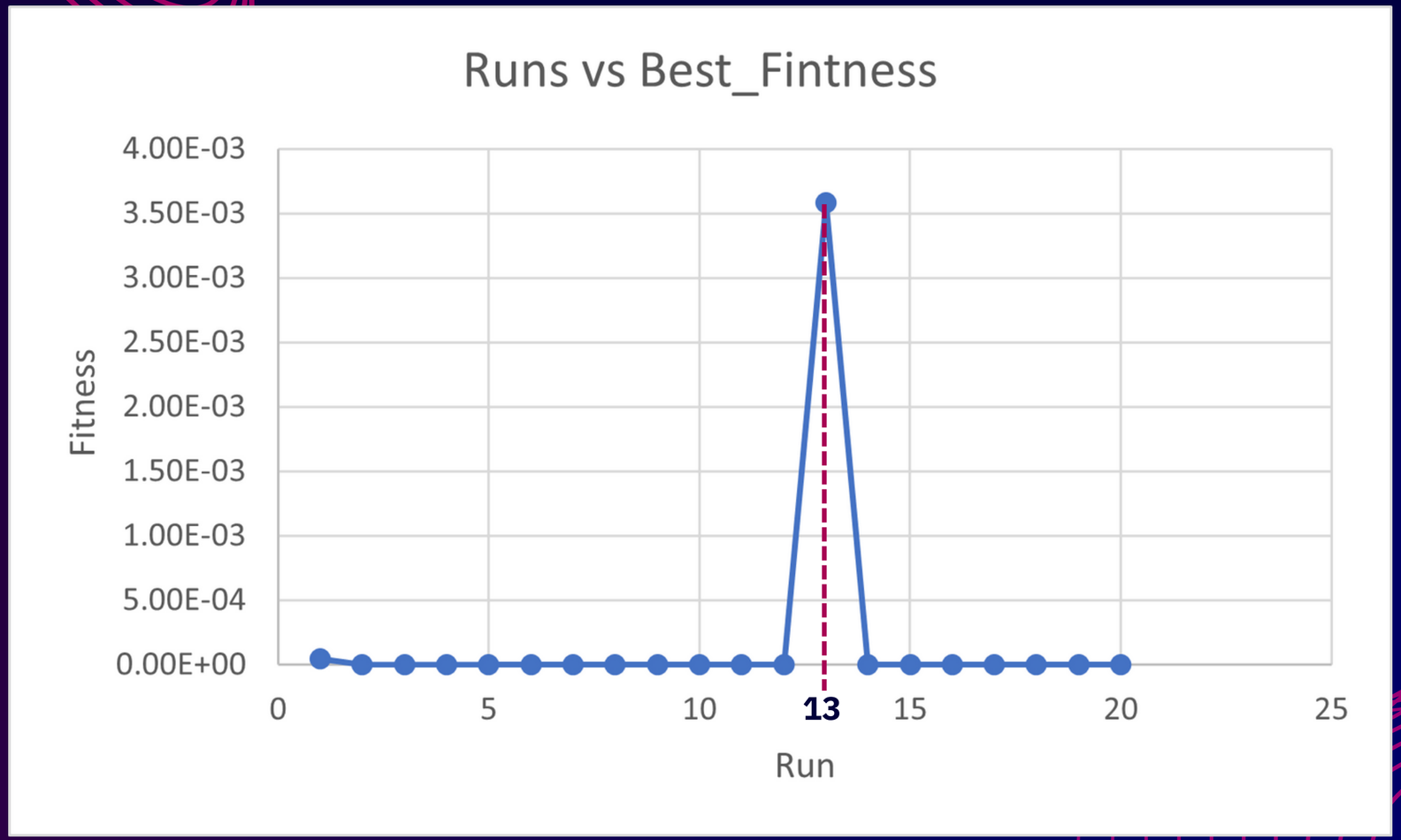
X1 values of the Best solution against the runs



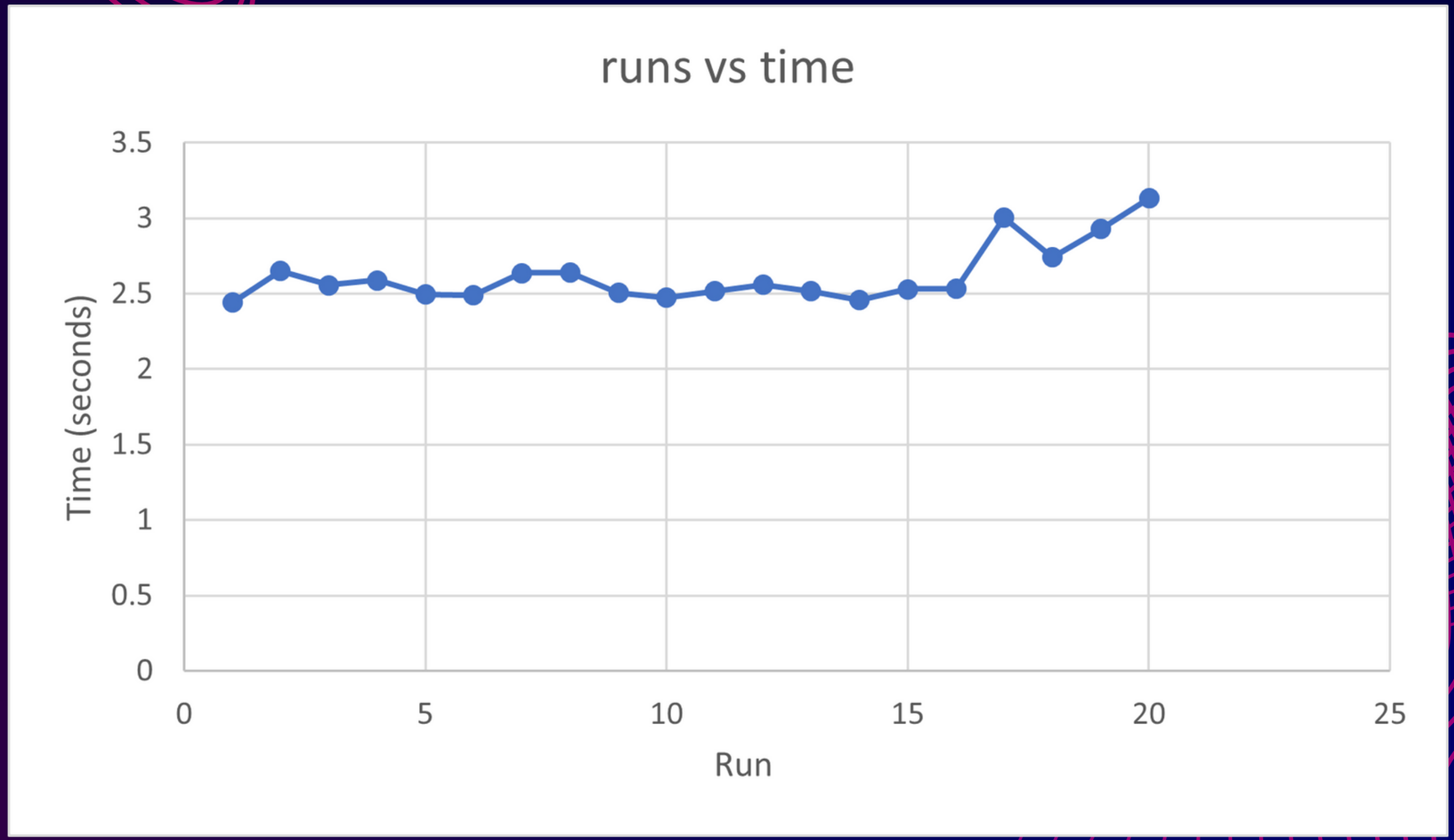
X2 values of the Best solution against the runs



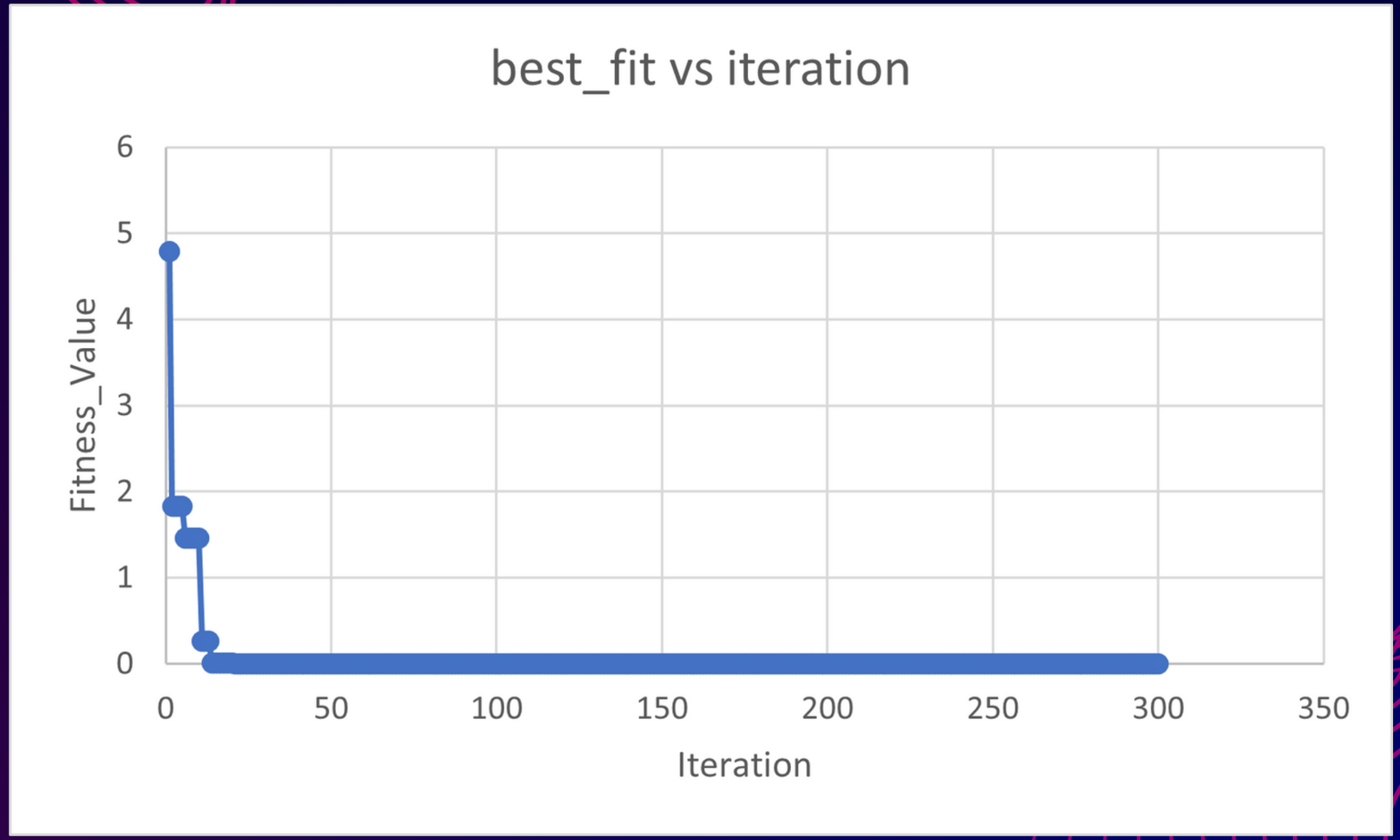
Best fitness values against the runs



Running time against the runs

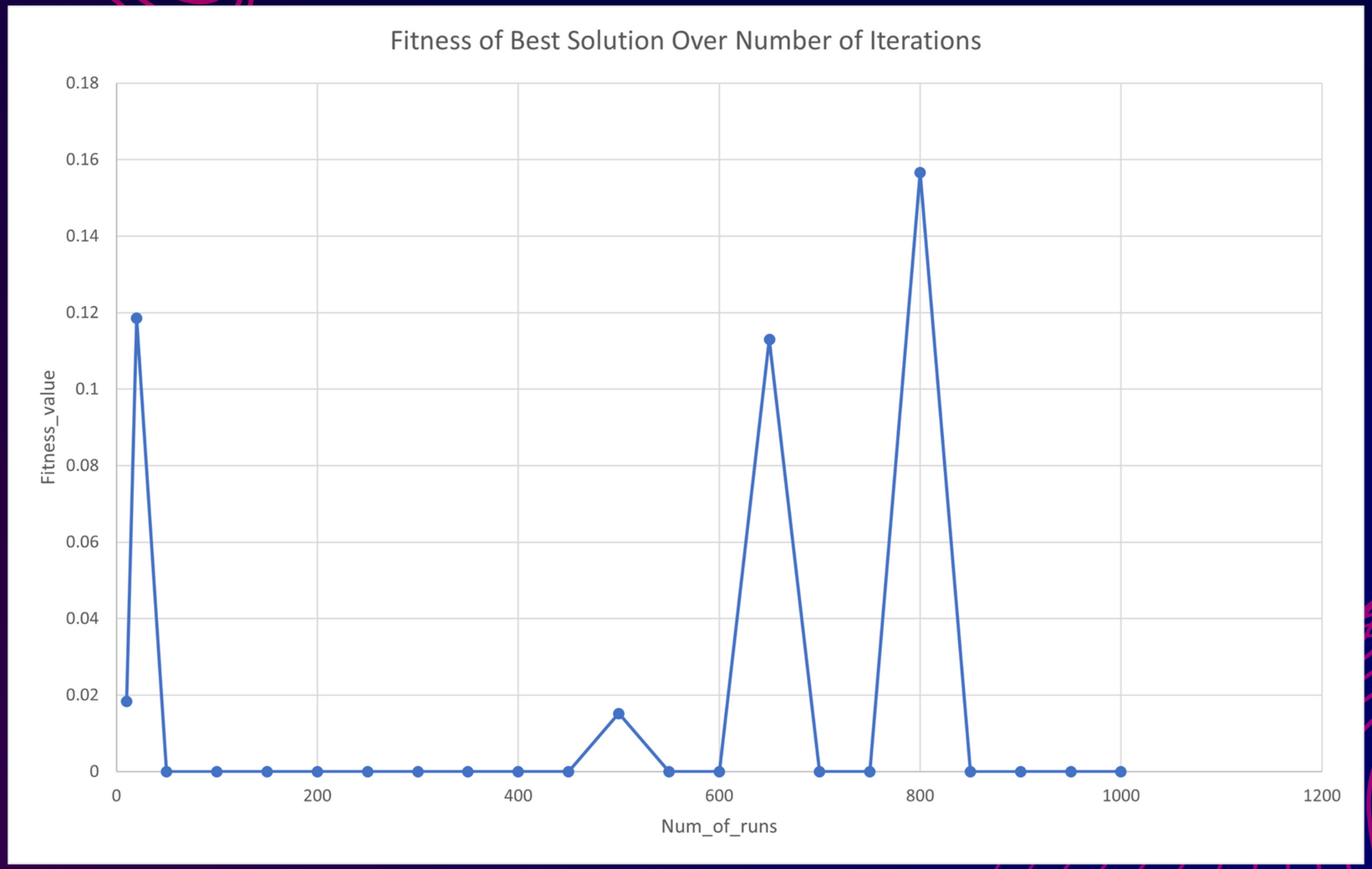


Best fitness values against the iterations considering a particular run



FINDINGS

Why have we selected 300 as the number of iterations for each run?



THANK YOU!