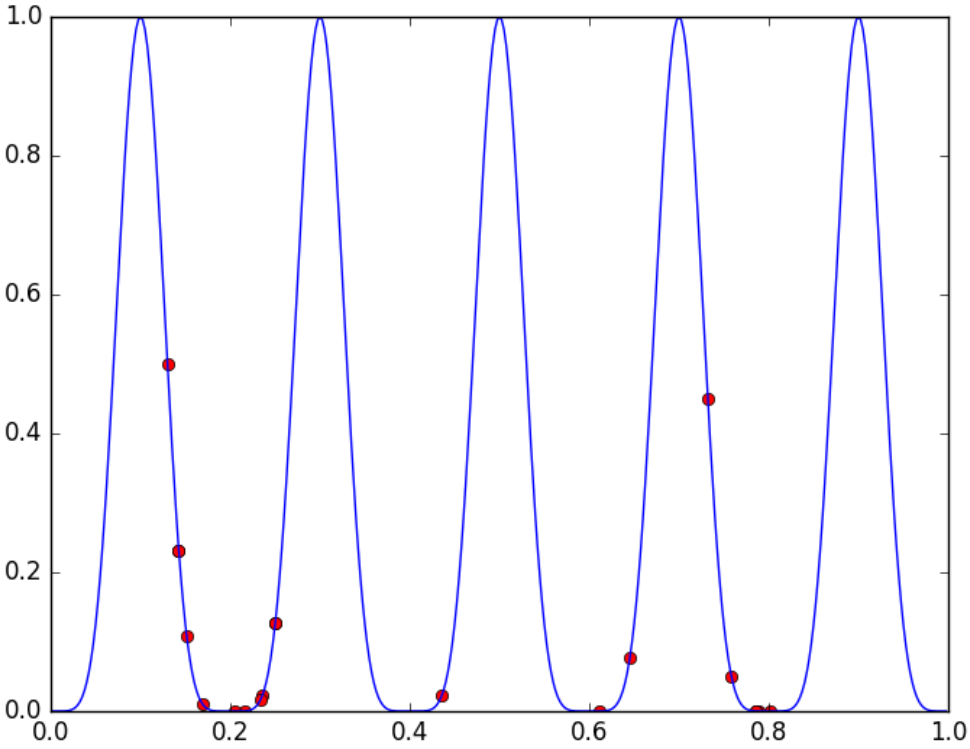
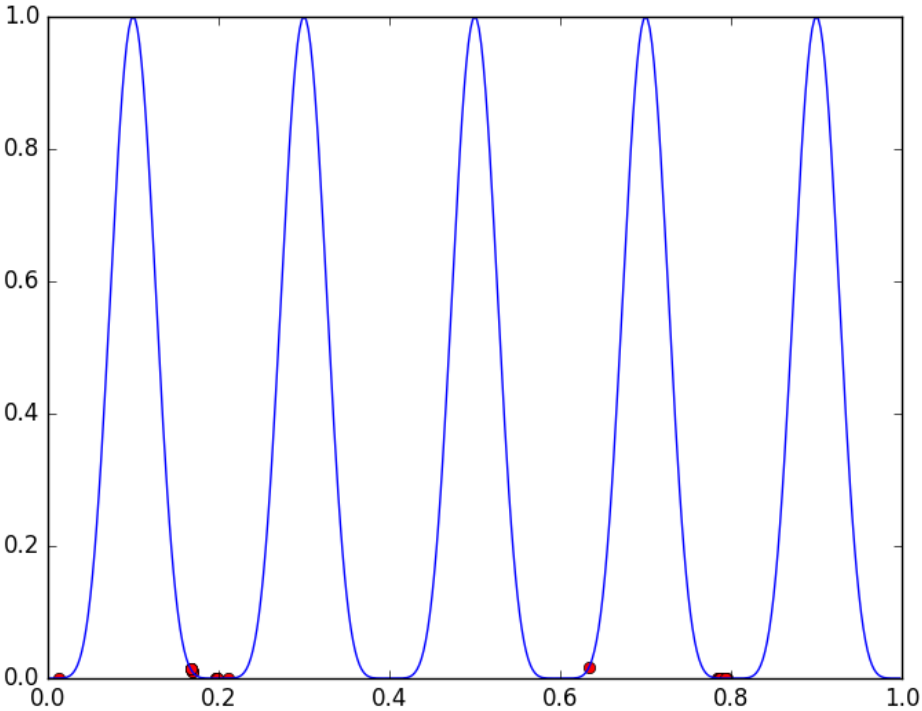
Crowding Algorithm with for function y = sin ^ 6 (5 pi x)

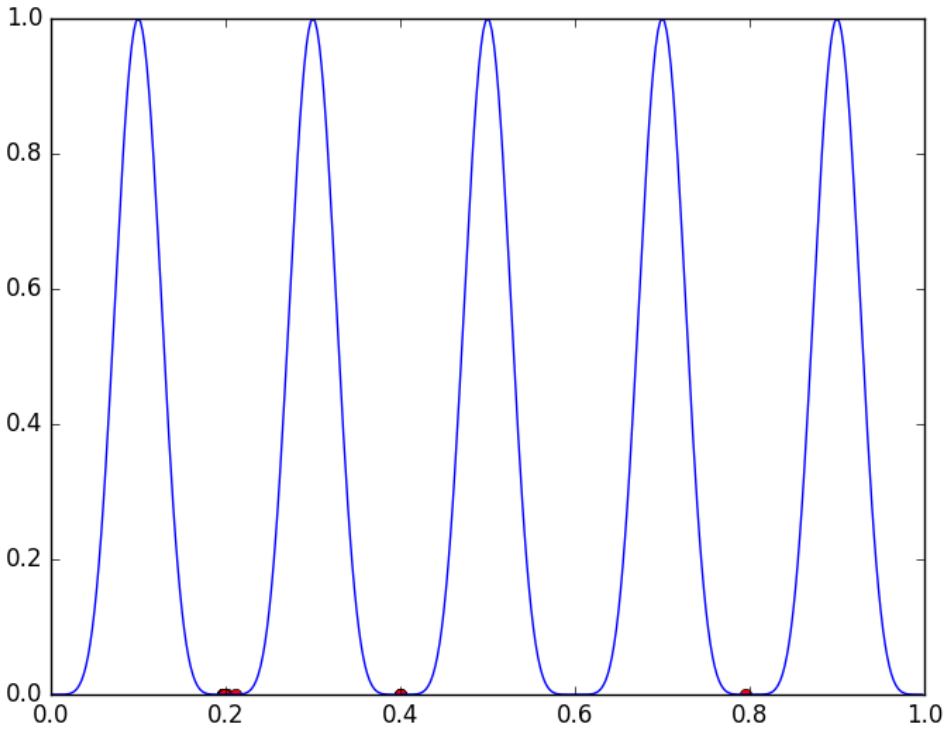
First random generation:



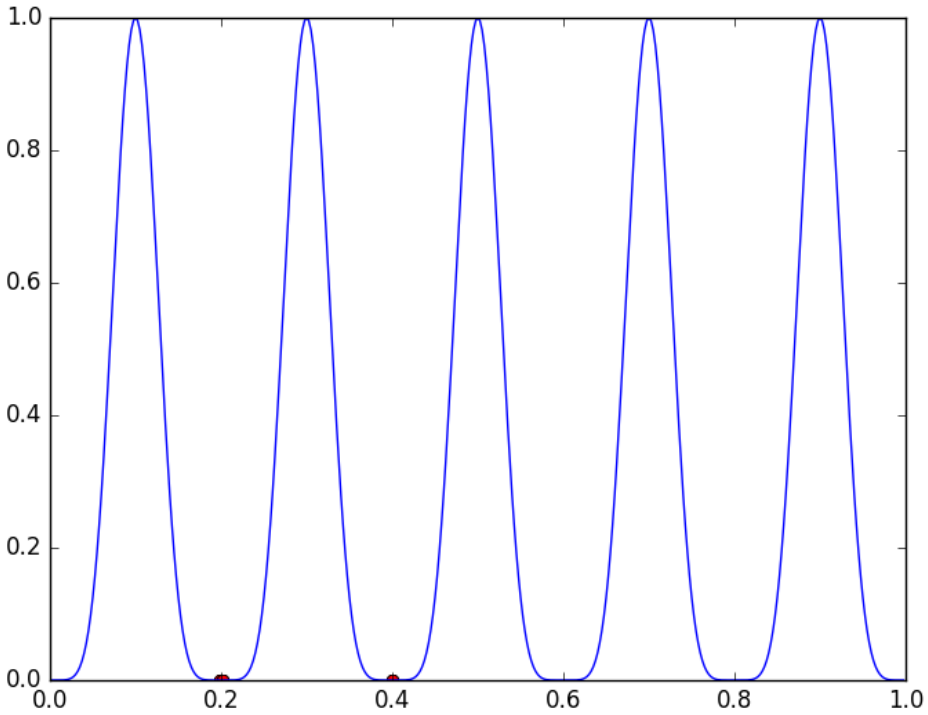
5th iteration:



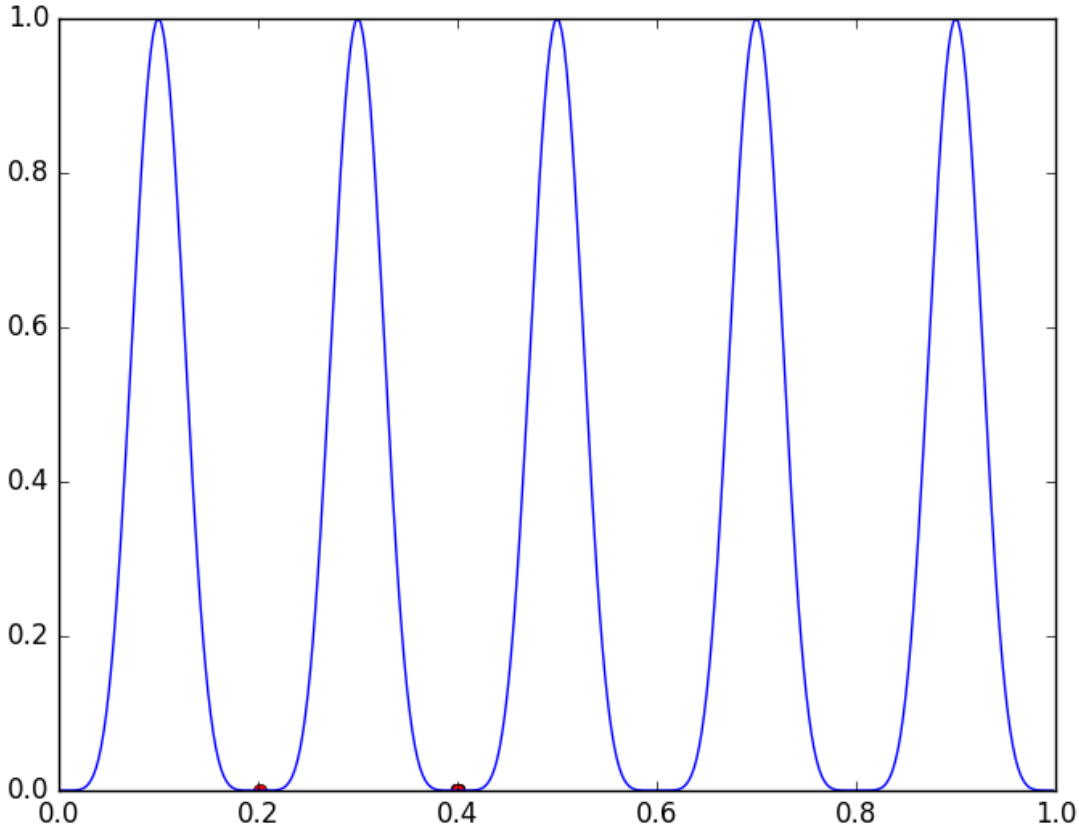
10th iteration



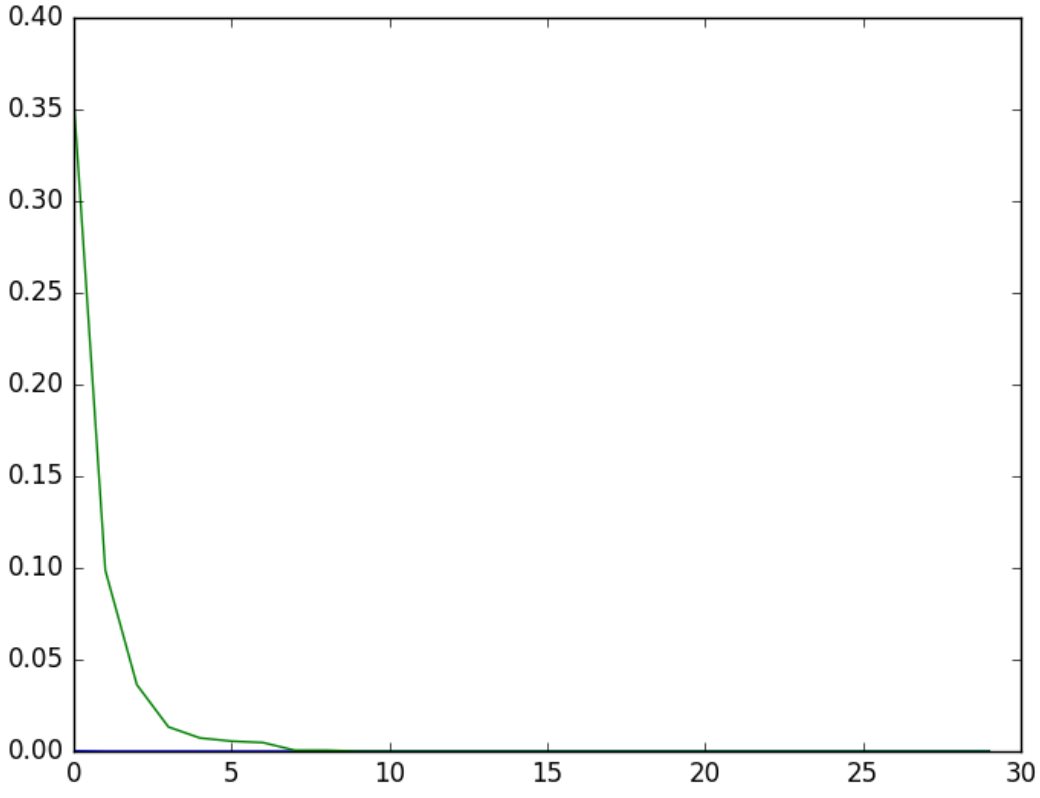
15th iteration



Final 20th iteration:



Graph of average and minimum values VS iteration



Note: Minimum values is very small, that because at the graph its a little bit difficult to recognize it.

Table of replacement for 0, 5, 10, 15 and 20th iterations

Parent 1	Parent 2	Child 1	Child 2	Result 1	Result 2
Iteration: 0					
0.301075	0.639296	0.435973	0.645161	0.645161	0.435973
0.784946	0.829912	0.610948	0.788856	0.788856	0.610948
0.247312	0.169110	0.216031	0.169110	0.169110	0.216031
0.905181	0.130010	0.888563	0.151515	0.151515	0.130010
0.140762	0.142717	0.140762	0.205279	0.205279	0.142717
0.250244	0.784946	0.292278	0.265885	0.250244	0.784946
0.247312	0.301075	0.732160	0.801564	0.801564	0.732160
0.140762	0.784946	0.157380	0.758553	0.758553	0.784946
0.142717	0.250244	0.127077	0.098729	0.142717	0.250244
0.487781	0.493646	0.233627	0.235582	0.235582	0.233627
Iteration: 5					
0.198436	0.786901	0.523949	0.773216	0.198436	0.786901
0.167155	0.012708	0.231672	0.042033	0.167155	0.012708
0.211144	0.633431	0.246334	0.633431	0.211144	0.633431
0.167155	0.783969	0.419355	0.877810	0.167155	0.783969
0.197458	0.169110	0.043988	0.197458	0.197458	0.197458
0.198436	0.169110	0.231672	0.133920	0.198436	0.169110
0.169110	0.784946	0.284457	0.153470	0.169110	0.784946
0.786901	0.783969	0.795699	0.847507	0.786901	0.795699
0.167155	0.638319	0.638319	0.167155	0.167155	0.167155
0.786901	0.167155	0.792766	0.161290	0.786901	0.792766
Iteration: 10					
0.197458	0.198436	0.202346	0.478983	0.197458	0.198436
0.198436	0.196481	0.198436	0.196481	0.198436	0.198436
0.197458	0.197458	0.197458	0.197458	0.197458	0.197458
0.794721	0.400782	0.262952	0.400782	0.400782	0.400782
0.198436	0.198436	0.197458	0.448680	0.198436	0.198436
0.794721	0.197458	0.697947	0.043988	0.794721	0.197458
0.197458	0.198436	0.201369	0.198436	0.198436	0.201369
0.198436	0.198436	0.135875	0.448680	0.198436	0.198436
0.198436	0.794721	0.544477	0.198436	0.198436	0.198436
0.211144	0.400782	0.150538	0.461388	0.211144	0.400782
Iteration: 15					
0.198436	0.400782	0.150538	0.448680	0.198436	0.400782
0.198436	0.201369	0.701857	0.198436	0.198436	0.201369
0.198436	0.400782	0.150538	0.452590	0.198436	0.400782
0.400782	0.201369	0.439883	0.201369	0.400782	0.201369
0.400782	0.400782	0.150538	0.400782	0.400782	0.400782
0.198436	0.201369	0.201369	0.714565	0.198436	0.201369
0.201369	0.400782	0.201369	0.392962	0.201369	0.400782
0.201369	0.201369	0.076246	0.138807	0.201369	0.201369
0.400782	0.400782	0.401760	0.400782	0.400782	0.400782
0.201369	0.201369	0.467253	0.201369	0.201369	0.201369
Iteration: 20					

0.400782 0.400782 0.403715 0.901271 0.400782 0.400782
0.400782 0.400782 0.404692 0.463343 0.400782 0.400782
0.201369 0.400782 0.701857 0.279570 0.201369 0.400782
0.400782 0.400782 0.150538 0.400782 0.400782 0.400782
0.400782 0.400782 0.150538 0.385142 0.400782 0.400782
0.400782 0.400782 0.400782 0.400782 0.400782 0.400782
0.399804 0.400782 0.398827 0.397849 0.399804 0.400782
0.400782 0.400782 0.154448 0.385142 0.400782 0.400782
0.201369 0.400782 0.701857 0.400782 0.400782 0.400782
0.400782 0.400782 0.025415 0.405670 0.400782 0.400782

Source code:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import random
import math
import matplotlib.pyplot as plt
import numpy as np
def create_population(gens, chromos):
    population = []
    for chrom in xrange(chromos):
        population.append([])
        for gen in xrange(gens):
            population[chrom].append(random.randint(0, 1))
    return population
def calc_y(x):
    return math.sin(5 * 3.1415 * x) ** 6
def calc_fitness(arr):
    x = convert_from_bin_to_dec(arr)
    x /= 1023.
    y = calc_y(x)
    return y
def find_best_chromos(population):
    list = []
    for chrom in population:
        list.append((calc_fitness(chrom), chrom))
    bubble_sort(list)
    r = []
    for item in list[len(population) / 2:]:
        r.append(item[1])
    return r
def create_childs(pop):
    size = len(pop)
    par1, par2 = random.randint(0, size - 1), random.randint(0, size - 1)
    cut_point = random.randint(0, len(pop[0]))
    parent1 = pop[par1]
    parent2 = pop[par2]
    child1 = parent1[:cut_point] + parent2[cut_point:]
    child2 = parent2[:cut_point] + parent1[cut_point:]
    return child1, child2, parent1, parent2
def bubble_sort(A):
    for i in range(len(A)):
        for k in range(len(A) - 1, i, -1):
            if A[k][0] > A[k - 1][0]:
                swap(A, k, k - 1)
def swap(A, x, y):
```

```
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp
def swap(s1, s2):
    tmp = s1
    s1 = s2
    s2 = tmp
    return s1, s2
def get_avg_count(population):
    chromos = len(population)
    average_list = []
    for chrom in xrange(chromos):
        cur_counter = calc_fitness(population[chrom])
        average_list.append(cur_counter)
    return sum(average_list) / len(average_list)
def minimum(population):
    min = 9999999999.
    for chrom in population:
        fintess = calc_fitness(chrom)
        if min > fintess:
            min = fintess
    return min
def convert_from_bin_to_dec(arr):
    num_str = ''
    for a in arr:
        num_str += str(a)
    return int(num_str, 2)
def is_mutant(prob, count_all):
    val = random.randrange(0, count_all)
    if val <= prob:
        return True
    return False
def inverse_value(val):
    if val is 0:
        return 1
    return 0
def crossover_mutate(child):
    # print("mutation of child")
    # print(child)
    len_all = len(child)
    prob = 1. / len_all
    mutated_child = []
    for bit in child:
        if is_mutant(prob, len_all):
            mutated_child.append(inverse_value(bit))
        else:
            mutated_child.append(bit)
    # print(mutated_child)
    return mutated_child
def distance(point1, point2):
    p1 = convert_from_bin_to_dec(point1) / 1023.
    p2 = convert_from_bin_to_dec(point2) / 1023.
    return math.fabs(p1 - p2)
def replace_parents_and_childs(p1, p2, c1, c2):
    if distance(p1, c1) + distance(p2, c2) > distance(p1, c2) + distance(p2,
c1):
        if calc_fitness(c1) < calc_fitness(p1):
            ret1 = c1
        else:
```

```
            ret1 = p1
        if calc_fitness(c2) < calc_fitness(p2):
            ret2 = c2
        else:
            ret2 = p2
    else:
        if calc_fitness(c2) < calc_fitness(p1):
            ret1 = c2
        else:
            ret1 = p1
        if calc_fitness(c1) < calc_fitness(p2):
            ret2 = c1
        else:
            ret2 = p2
    return ret1, ret2
def create_new_generation(old_gen):
    new_gen = []
    for iter in xrange(len(old_gen) / 2):
        ch1, ch2, pr1, pr2 = create_childs(old_gen)
        ch1 = crossover_mutate(ch1)
        ch2 = crossover_mutate(ch2)
        rep1, rep2 = replace_parents_and_childs(pr1, pr2, ch1, ch2)
        new_gen.append(rep1)
        new_gen.append(rep2)
    return new_gen
def get_avg_count(population):
    chromos = len(population)
    average_list = []
    for chrom in xrange(chromos):
        cur_counter = calc_fitness(population[chrom])
        average_list.append(cur_counter)
    return sum(average_list) / len(average_list)
def find_best_chromosome(min, all_chromos):
    for chr in all_chromos:
        if min == calc_fitness(chr):
            return chr
def main():
    population = create_population(10, 20)
    mins = []
    avgs = []
    count_iter = 30
    for i in xrange(count_iter):
        min = minimum(population)
        mins.append(min)
        avg = get_avg_count(population)
        avgs.append(avg)
        # print(str(i) + ";min;" + str(min))
        # print(str(i) + ";" +
str(convert_from_bin_to_dec(find_best_chromosome(min, population))/1023.))
        population = create_new_generation(population)
        if i in [0, 5, 10, 15, 25]:
            print("Iteration: " + str(i))
            xs, ys = [], []
            for chr in population:
                x = convert_from_bin_to_dec(chr) / 1023.
                y = calc_fitness(chr)
                print(str(x) + ";" + str(y))
                xs.append(x)
                ys.append(y)
```

```
            b_x = np.arange(0, 1, 0.0025)
            b_y = []
            for _x in b_x:
                b_y.append(calc_y(_x))
            plt.plot(xs, ys, 'ro')
            plt.plot(b_x, b_y,'b')
            plt.show()
    print(mins)
    plt.plot(range(count_iter), mins, 'b')
    plt.plot(range(count_iter), avgs, 'g')
    plt.show()
if __name__ == "__main__":
    main()
```