VERKFRÆÐILEGAR BESTUNARAÐFERÐIR

Day 7 - Group 2

T-423-ENOP

Arnar Gylfi Haraldsson
arnarh23@ru.is

Hafþór Árni Hermannsson
hafthorh20@ru.is

Ragnheiður Gná Gústafsdóttir
ragnheidurg@ru.is

May 10, 2024

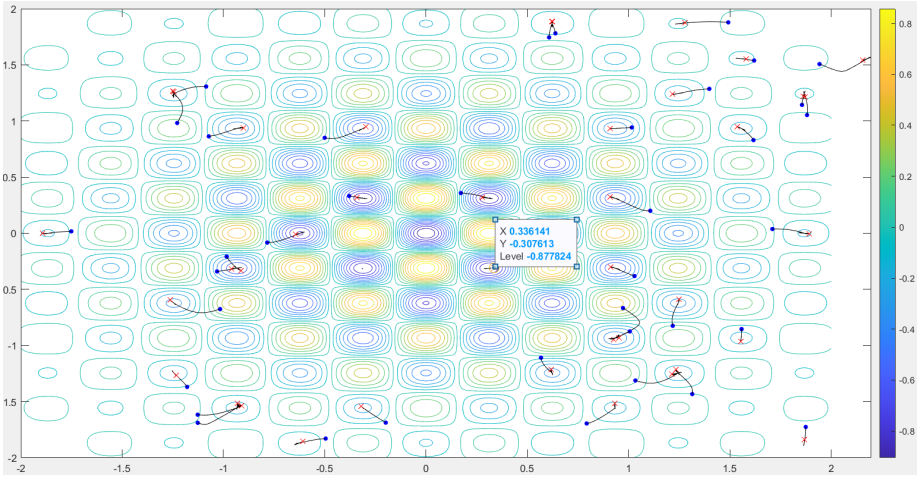
Exercise 1

Description

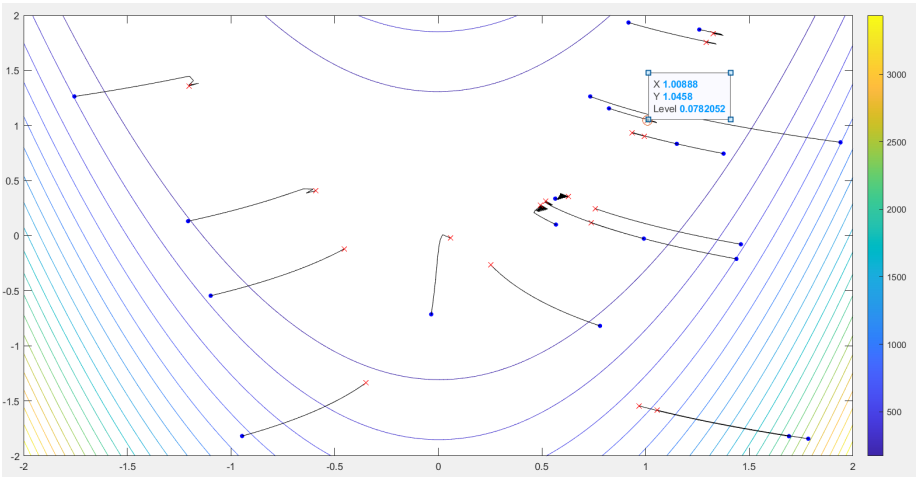
Implement multiple-run gradient-search algorithm using random starting point in each run. The results should be displayed after each run including run number, best objective function value found so far, number of function evaluations.
Test the algorithm on:
(1) Rosenbrock function (n = 2, $-2 \leq x_i \leq 2$),
(2) Function fp (n = 1, 2, $-2 \leq x_i \leq 2$),
(3) Auckley function (n = 1, 2 and 3, $-10 \leq x_i \leq 10$).

Solution

In each iteration we create a random point within a range, and then compute the gradient and take a small step in the opposite direction of the gradient, the step size being inversely proportional to the number of steps taken, so we don't overshoot. We iterate the gradient descent 30 times. We repeat the process numerous times, randomly generate points and locally optimize them, saving the best value. Below is an example output optimizing the p-function with 40 points:



Example output for two dimensional Rosenbrock function:



Running the optimization 30 times on the 3 dimensional Ackley function using the same parameters we get the following averages:

$\overline{(x_1, x_2, x_3, f(x_1, x_2, x_3))}$ = (0.1294, 0.4301, 0.2454, ,4.4585),
the best result:
(0.0076, 0.0200, −0.0040, −1.6596),
and the worst result:
(−1.0285, 4.9608, 0.0004, 7.1812) Standard deviations: 1.8858 2.1224 1.8975 1.8515

Exercise 2

Description

Implement random search algorithm using uniform probability distribution in the search space. The results should be displayed after each iteration: best value of the objective function value found so far and the number of function evaluations. Modify the algorithm to make the search "smarter".

Test the algorithm on:

1. Rosenbrock function (n = 2, $-2 \leq x_i \leq 2$),
2. Function fp (n = 1, 2, $-2 \leq x_i \leq 2$),
3. Auckley function (n = 1, 2 and 3, $-10 \leq x_i \leq 10$).

Compare the basic version of the random search with the "smart" one.

Solution

The smartRandomSearch function was created according to the following pseudocode:

x_{best} = generate_random_point();
$f_{best} = f(x_{best})$;
$k = 0$;
while $k < k_{max}$
  x_{new} = generate_random_point();
  $x_{new} = \lambda x_{new} + (1-\lambda)x_{best}$;
  $f_{new} = f(x_{new})$;
  if $f_{new} < f_{best}$
    $x_{best} = x_{new}$;
    $f_{best} = f_{new}$;
  end
  λ = update_lambda(k,k_{max});
  $k = k + 1$;
end

Figure 1: delay.h

where lambda is updated as: $\lambda = 1 - k/kmax$
Plotting the results for all three functions with n = 2 and k = 1000 we get:

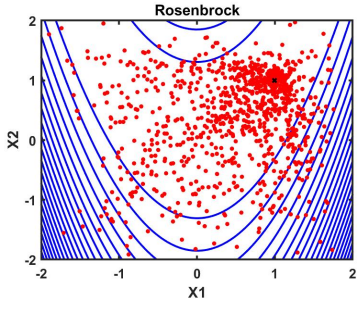


Figure 2: Visualization Rosenbrock

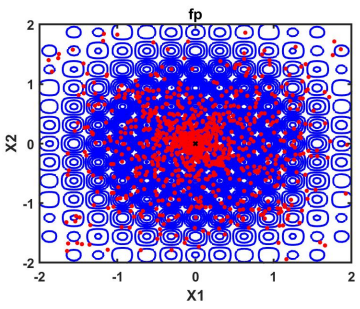


Figure 3: Visualization fp

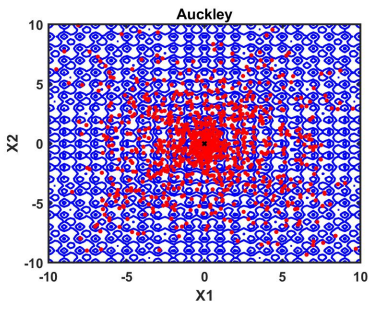


Figure 4: Visualization Auckley

The algorithm was tested on all functions for different n:

Table 1: absolute error values of algorithm solution for 100 runs of algorithm

	n	Mean Error	Std Error	Max Error	Min Error
Rosenbrock	2	0.004153	0.006531	0.035036	0.000005
fp	1	0.000007	0.000009	0.000000	0.000046
	2	0.009951	0.027984	0.000000	0.093954
Auckley	1	0.005108	0.006390	0.000079	0.051091
	2	0.047867	0.039224	0.004258	0.187984
	3	0.203740	0.408685	0.018320	2.165900

Comparing the smartRandomSearch with randomSearch for 1000 runs of each algorithm we get:

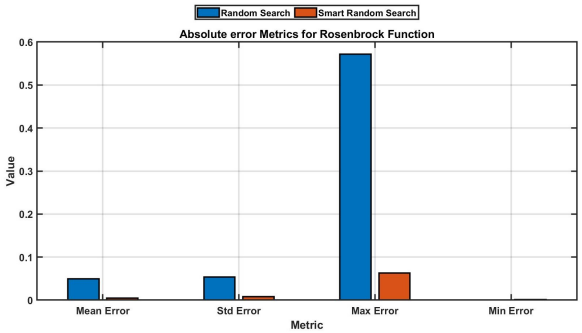


Figure 5: randomSearch VS smartRandomSearch on fr

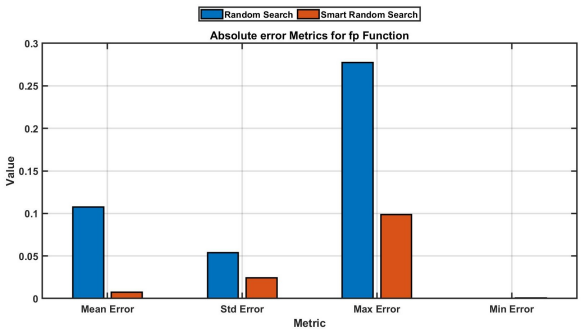


Figure 6: randomSearch VS smartRandomSearch on fp

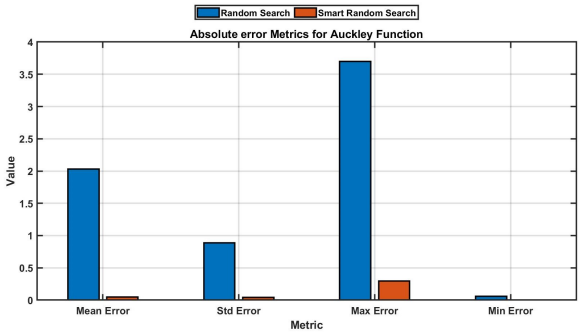


Figure 7: randomSearch VS smartRandomSearch on fa

Exercise 3

Description

Differential evolution (DE) algorithm is one of the most recent population search methods used for minimization of function f of n variables. Outline (0 F 2 and 0 CR 1 are user defined control parameters):

- Initialize N individuals (random selection);
- Until termination repeat:
  - For each individual $x = [x_1, \ldots, x_n]$:
  - Pick three random individuals y_1, y_2, y_3 (all, including x, distinct);
  - Pick a random index $p \in \{1, \ldots, n\}$;
  - Generate new individual $y = [y_1, \ldots, y_n]$ by iterating for $i = 1$ to n:
    - Pick random $r \in (0, 1)$ (uniform distribution);
    - If $i = p$ or $r < CR$, set $y_i = y_{1i} + F(y_{2i} - y_{3i})$, otherwise set $y_i = x_i$;
    - If $f(y) < f(x)$ set $x = y$;

Test the algorithm on:

1. Rosenbrock function ($n = 2$, $-2 \leq x_i \leq 2$),
2. Function fp ($n = 1, 2$, $-2 \leq x_i \leq 2$),
3. Auckley function ($n = 1, 2$ and 3, $-10 \leq x_i \leq 10$).

Use $F = 1$ and $CR = 0.5$ initially, then change F and CR to see how it affects the algorithm performance.

Functionality

We implemented the algorithm according to the instructions and added a convergence stoppage, the fbest and xbest when that convergence was reached or the iterations were done and a collection of that data to be plotted after the algorithm had been run. We also implemented a plotter that shows the process and some statistical analysis.

Results

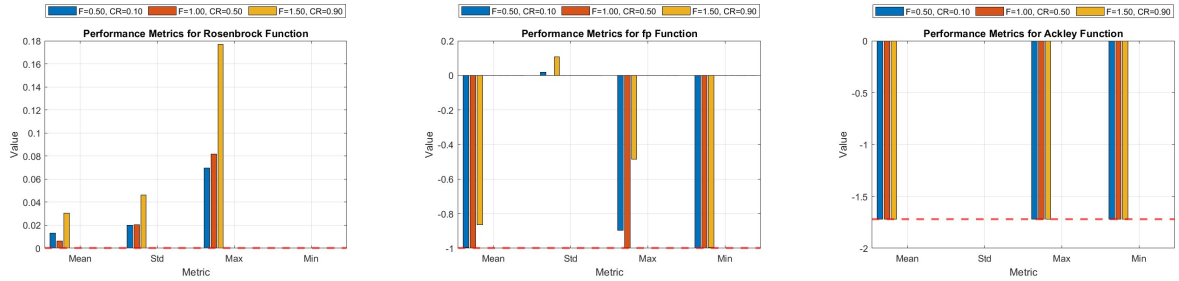
These following tables and figures show our results, the live visualization of the history can be seen if our code is run, specifically the 'Visulization of the history for each function' section at the bottom of the script. These results are as expected, the Rosenbrock function had by far the worst results, as the professor mentioned would happen due to the smoothness of the function.

Table 2: Statistical Measures for Rosenbrock, f_p and Ackley Functions

Function	n	Mean	Std Dev	Max	Min
Rosenbrock Function	1	0.00000	0.00000	0.00000	0.00000
Function f_p	1	-1.00000	0.00000	-1.00000	-1.00000
	2	-0.99982	0.00081	-0.99561	-1.00000
Ackley Function	1	-1.71828	0.00000	-1.71828	-1.71828
	2	-1.71828	0.00000	-1.71828	-1.71828
	3	-1.71828	0.00000	-1.71828	-1.71828

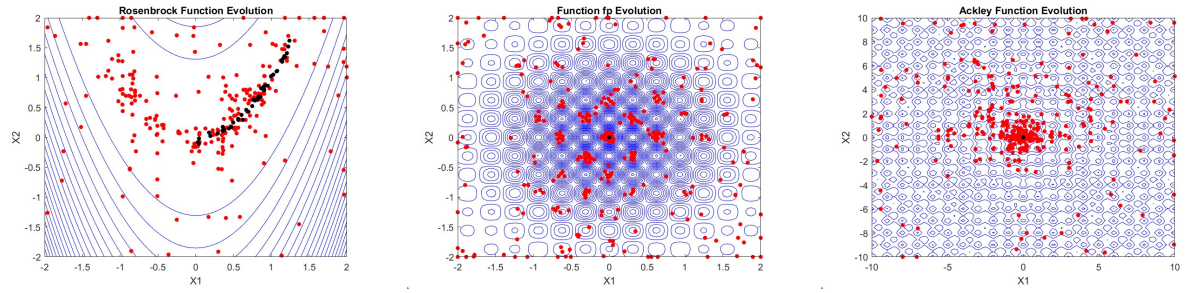
Table 3: Statistical Results for Differential Evolution Algorithm,

Function	F	CR	Mean	Std	Max	Min
Rosenbrock true minima = 0	0.50	0.10	0.01041	0.01633	0.08068	0.00001
	1.00	0.50	0.00872	0.02856	0.14419	0.00000
	1.50	0.90	0.02017	0.03363	0.11183	0.00000
f_p true minima = -1	0.50	0.10	-0.99998	0.00007	-0.99963	-1.00000
	1.00	0.50	-0.99994	0.00032	-0.99827	-1.00000
	1.50	0.90	-0.87087	0.07663	-0.68260	-0.98596
Ackley true minima = -1.7183	0.50	0.10	-1.71828	0.00000	-1.71828	-1.71828
	1.00	0.50	-1.71828	0.00000	-1.71828	-1.71828
	1.50	0.90	-1.71616	0.01159	-1.65478	-1.71828



(a) RosenBrock function (b) f_p function (c) Ackley function

Figure 8: Statistic on different values of F and CR with n=2 for all three test functions



(a) F = 0.5, CR = 0.1 and n = 2 for the RosenBrock function (b) F = 0.5, CR = 0.1 and n = 2 for the f_p function (c) F = 0.5, CR = 0.1 and n = 2 for the Ackley function

Figure 9: Visualization of the history of the differential evolution algorithm with red being previous populations and black being the current one.

Exercise 4

Description

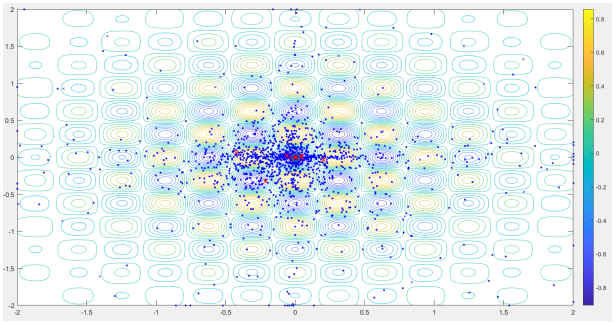
Implement the PSO algorithm and test its performance on the following benchmark functions:

1. Rosenbrock function ($n = 2$, $-2 \leq x_i \leq 2$),
2. Function f_p ($n = 1,2$, $-2 \leq x_i \leq 2$),
3. Ackley function ($n = 1,2$, and 3, $-10 \leq x_i \leq 10$).

Perform experiments to investigate how the performance of the algorithm depends on its control parameters (e.g., swarm size N, inertia weight χ, cognitive parameter c_1, social parameter c_2, etc.).

solution

We initially set $c_1 = 0.205$, $c_2 = 4.1 - c_1$, $\chi = 0.7298$ and we use N = 20. The number of function evaluations is always $N(n+1)$ For 100 iterations this yields the following results:
Rosenbrock n = 2:
`xbest:  [0.9998 0.9998], fbest:  0.000000`
f_p n = 1
`xbest:  1.2698e-08, fbest:  -1.000000`
f_p n = 2
`xbest:  [0.3110 0.3110], fbest:  -0.906904`
Ackley function, n = 1
`xbest:  -9.9405e-08, fbest:  -1.718281`
Ackley function, n = 2
`xbest:  [5.3254e-06 -2.6105e-06], fbest:  -1.718265`
Ackley function, n = 3
`xbest:  [2.1249e-05 -1.1868e-05 -0.000108], fbest:  -1.718025`
We also plot up the optimization process when n = 2, the blue dots are previous swarms and the red is the current swarm iteration.
For example the Rosenbrock function:



Next we experiment with changing the chi values. We change it each iteration so that $\chi = \chi_{max} \frac{\chi_{max} - \chi_{min} \cdot i}{i_{max}}$ Where $\chi_{max} = 1$, $\chi_{min} = 0.4$, i is the current iteration count and i_{max} is the total number of iterations to be performed. N, c_1 and c_2 remain unchanged. We then count the iterations and function evaluations needed to reach within 10^{-10} of the true value. For the Auckley function in 3 dimensions we needed on average 250 iterations and 5020 function evaluations in the case where $\chi = 0.7298$. In the case where χ changes iteratively, the number of iterations and function values needed is marginally higher, around 265 iterations.

Comparing all methods

For fun we compared the methods from exercises 1-4 for n = 2, k = 1000, and running each algorithm 20 times:

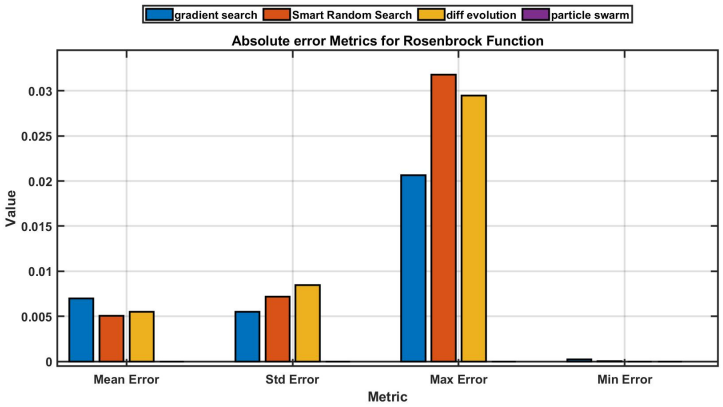


Figure 10: comparison of all algorithms on fr

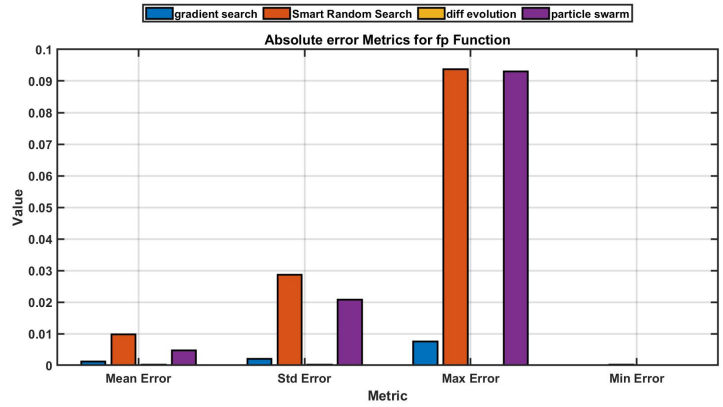


Figure 11: comparison of all algorithms on fp

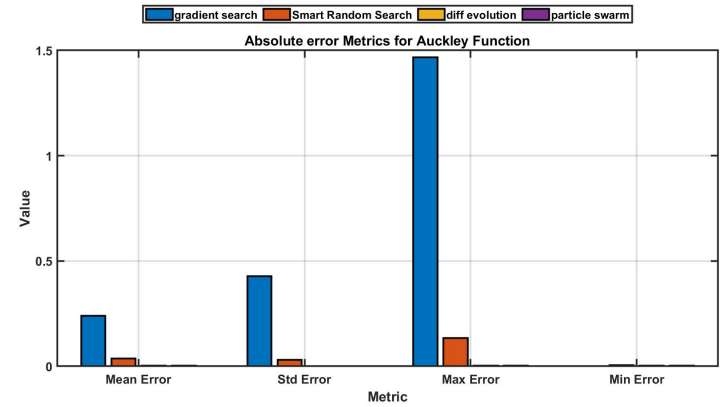


Figure 12: comparison of all algorithms on fa