Asn 3: Particles that Swarm

Ethan Heavey
St. Francis Xavier University
Department of Computer Science
Antigonish, Nova Scotia, Canada
x2018zbf@stfx.ca

***Abstract*—Particle swarm optimization is an optimization algorithm based on real-world particle intelligence. Through velocity clamping and meteorites, the particles will be pushed to their limits in an effort to solve the Ackley function. Further looking into the statistical significance of the length of time an instance of PSO takes to find the optimal solution can provide insight into the results, and even aid in identifying how extinction events and velocity clamping really affects the results of PSO.**

***Index Terms*—Evolutionary Computation; Particle Swam Optimization; Genetic Algorithm; Swarm Intelligence;**

I. Introduction

Particle Swarm Optimization (PSO) is an optimization algorithm based on real-world examples of swarm intelligence. Given a search space, the stochastic, population-based algorithm is based on the simulation of the social behavior of birds within a flock[1], using global bests in conjunction with individual bests top best navigate the search space. An implementation of PSO will be used to solve the Ackley function, shown below.

$$f(x_0 \cdots x_n) = -20 * exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$$

II. The Problem

The Ackley function shown in I was the toy problem this implementation of PSO was tested on. In the initialization phase of the algorithm, the positions of each particle was chosen at random using numpy's *randn* function. The values were generated within in the domain shown below. The minimum value of the Ackley function is also shown.

$$\text{Domain: } -32.768 \leq x_i \leq 32.768, i = 1, \cdots, n$$

$$\text{Minimum at } f(0, \cdots, 0) = 0$$

Characterized by a nearly flat outer region with a large dip in the center, the Ackley function is widely used for testing optimization algorithms[3]. A plot of the function over the domain mentioned is visible in Fig 1.

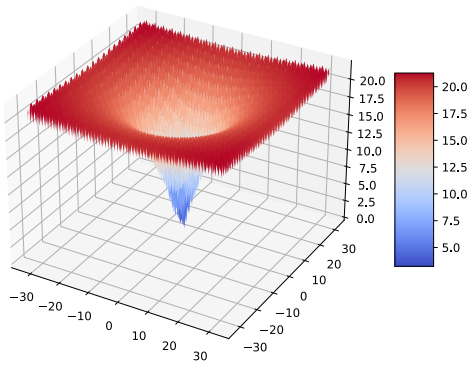

Fig. 1: Ackley function in a 3D space

III. The Algorithm

The first of the trio of letters representing PSO references the particles. Each particle does its own thing but the nature of PSO allows all particles to move *together*. Each particle knows the following attributes:

- $\overrightarrow{x}(t)$ represents the particle's position at time t
- $\overrightarrow{v}(t)$ represents the particle's velocity at time t
- $\overrightarrow{p}_{best}$ represents the particle's best know position
- $\overrightarrow{g}_{best}$ represents the swarm's best known position

The particles are represented using a two-dimensional numpy[2] array, with the rows of the array representing the x and y co-orindates of a given particle, represented by the column number. A second, one-dimensional, numpy array is used to hold the fitness values of each particle, with the column number of the fitness array corresponding to the particle of the same column number. The global best is represented as a par of values, an x and a y value that, when passed through the

Ackley function will yield the value of the global best.

Next up, the Swarm aspect of PSO. This allows the algorithm to utilize the known global best in conjunction with an individual particle's best known position attribute in order to calculate and update the velocity of each particle, and move them towards the optimal solution. One of the most advantageous aspects of PSO is it's ability to avoid local maxima or minima by having the swarm's particles search the entire search space at once. This allows a substantial degree of diversity in the swarm, and allows individual particles to venture off in search of what they believe to be the optimal position.

Last but not least, the Optimization portion. This process revolves around using the current position, velocity, and global best to update the position of all particles. There are three parts to the velocity update function, listed below.

- Inertia Term: $w\overrightarrow{v_i}(t)$
  - $w \in [0, 1]$
  - The inertia of an object allows the previous time-step's velocity to affect the new velocity
- Cognitive Term: $c_1\overrightarrow{r_1}(\overrightarrow{p_{best}} - \overrightarrow{x_i}(t))$
  - $\overrightarrow{c_1} \in [0, 2]$
  - $\overrightarrow{r_1} \in [0, 1]$, adds stochasticity
  - Signifies the influence of the personal best known position on velocity
  - A high c_1 value results in introverted particles, where the particles will exploit local parts of the search space.
- Social Term: $c_2\overrightarrow{r_2}(\overrightarrow{g_{best}} - \overrightarrow{x_i}(t))$
  - $\overrightarrow{c_2} \in [0, 2]$
  - $\overrightarrow{r_2} \in [0, 1]$, adds stochasticity
  - Signifies the influence of the global best known position on velocity
  - A high c_2 value results in extroverted particles, where the particles will exploit the area nearest the global best within the search space.
- $\overrightarrow{r_1}$and $\overrightarrow{r_2}$ values assist in preventing premature convergence on top of adding stochasticity.

All these terms come together in the velocity calculation. In the following equation, i represents some particle, with $t + 1$ indicating the next time-step of the algorithm.

$$\overrightarrow{v_i}(t+1) = w\overrightarrow{v_i}(t) + c_1\overrightarrow{r_1}(\overrightarrow{p_{best}} - \overrightarrow{x_i}(t)) + c_2\overrightarrow{r_2}(\overrightarrow{g_{best}} - \overrightarrow{x_i}(t))$$

A. Parameters

In this PSO implementation, there are a number of parameters that can be tuned. First and foremost, the number of particles ($NUM\_PARTICLES$) in the swarm. The higher the density of the particles, the quicker the algorithm tends to find the global optimum. Other PSO specific parameters include the values of c_1, c_2, and w. In section III-C, the purpose of the $EXTINCTION\_CHANCE$, $EXTINCTION\_PERCENTAGE$, and $VELOCITY\_MAX$ will be elaborated upon.

B. Structure

This implementation of PSO has been structured as follows:

- Randomly initialize particles and calculate their velocities
- While stop condition is not met or max iterations has not been reached:
  - Update particle position
  - Update individual $\overrightarrow{p_{best}}$ attribute for each particle
  - Update particle velocities
  - Update g_{best}
  - Check if extinction is to occur
    - Perform extinction
- return number of iterations and the $\overrightarrow{g_{best}}$ value

The choice to encode particles as elements in a numpy array was made to allow efficient calculations during the update process, without any bookkeeping overhead. The update to particle velocity was performed following the equation outlined in section III.

C. Modifications

So, meteorites. Or whatever kills particles these days (radiation)? An extinction event has a chance (*EXTINCTION_CHANCE*, *EC*) to eliminate a predetermined percentage of the population (*EXTINCTION_PERCENTAGE*, *EP*), which causes the "extinct" particles to randomly acquire a new position within the search space. The fitness values of these particles are updated accordingly.

A second modification, velocity clamping, was also added. The *VELOCITY_MAX* parameter controls the affect of this modification on the algorithm. Velocity clamping is used to control the maximum value of each individual particles velocity, which affects the overall progression of the algorithm, and causes it to take its time getting to the optimal solution.

Table I shows eight different rows consisting of eight different tests. All tests were conducted with $NUM\_PARTICLES$ set to a static value of 30, with each individual test running 50 times. The mean column represents the average number of iterations PSO ran for until reaching the optimal solution. The cap on the number of iterations per run of PSO was 10000. As is clearly visible in the table, and to common sense, a high change of an extinction event occurring would delay the algorithm in finding the optimal solution. An interesting note regarding the extinction events is that, even when given a mere 10% chance to occur, with 90% of the particles having their positions randomized, PSO takes the maximum number of iterations to conclude, without finding the optimal solution. Credit where credit is due however, the results of the runs that went to the maximum number of iterations were relatively close to the global minima of $(0, 0)$.

IV. Analysis Methodology

In the analysis of PSO's performance on the Ackley function, plots were generated to visualize the movement of particles through the search space. The effect of parameter changes can be compared through these visualizations. Another comparison that can be made is observing how close the algorithm gets to the "true" minimum of $(0, 0)$. In comparing the proposed best to actual best, it can been observed how efficient and accurate the algorithm can be. A final comparison can be comparing the effect of one (or both) modifications on the base PSO algorithm.

V. Results

For context, Fig 2 is the search space that PSO will be operating over. The domain pictured is $-25 \leq x \leq 25$. The minima of $(0, 0)$ can be identified by the white "X".

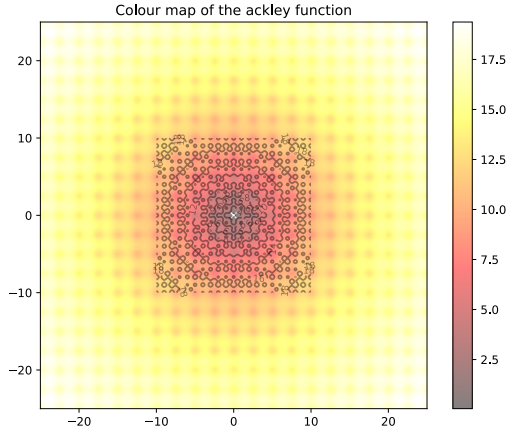

Fig. 2: Top-Down 2D view of the Ackley function

A. Search Space

1) Low Extinction Chance, Low Extinction Percentage, No Velocity Clamping: Figs 5 to 7 depict the search space with a low chance of extinction, extinction percentage, and no velocity clamping. As can be seen in Fig 5, the particles are randomly assigned an x and y value. By iteration 50, shown in Fig 6, most particles have been able to use $\overrightarrow{g_{best}}$ to converge on the global minimum. The final iteration of PSO under the aforementioned parameters yields the colour map of the Ackley function in Fig 7, with almost all particles in the center of the plot.

2) Low Extinction Chance, Low Extinction Percentage, Velocity Clamping: In another instance of PSO, Figs 8 to 10 show how velocity clamping affects the overall progression of the algorithm. Comparing the velocity arrows in Fig 7 to Fig 10, the Fig 7 arrows are clearly much larger than the ones in Fig 10. An interesting note is that the PSO instance with velocity clamping activated concluded it's execution 5 iterations before the PSO instance with no clamping. This could be due to the fact that the clamping provides a stable and steady increase to the velocity, ensuring each particle moves in the general direction of the of $\overrightarrow{g_{best}}$, but does not overshoot it.

3) High Extinction Chance, High Extinction Percentage, No Velocity Clamping: In Fig 11, it can be seen how, with no restriction on the velocity value, the particles want to go everywhere. An effect of extinction can be seen in comparing Fig 12 and Fig 13, where the position of the particles are completely different and have actually moved *backwards* from the velocities depicted in Fig 12.

4) High Extinction Chance, High Extinction Percentage, Velocity Clamping: Comparing Figs 14 to 16 to Figs 11 to 13 it can be seen that the velocity of the particles is significantly reduced, due to clamping, and no velocity is overshooting the global minima.

Overall, velocity clamping has a significant impact on the algorithm, with the extinction event significantly impeding the progress of the algorithm$\cdots$ if it occurs. A higher extinction event chance causes the space to get drastically altered, drastically affecting the particles and their values.

B. Stats and Box Plots

Given the stochastic nature of the algorithm, it is important to perform a statistical analysis on the results. After 50 executions of the algorithm, the results were compiled into box plots in order to isolate and identify any outliers. The results being analyzed are the number of iterations the algorithm has taken to complete execution. In Fig 3, it can be seen that there are some outlying instances of PSO that took a bit longer to run than other instances, a mild plot in comparison to Fig 4. As can be seen, instance "1" of PSO, when executed with velocity clamping turned on and set to a value of 0.25, had 4% of executions utilize the maximum number of iterations before completing, without even getting to the center of the function.

Conducting a t-test on the data used to generate the aforementioned box plots, the results were not out of the ordinary. As could be predicted, the results from comparing instances "1" and "2" from Fig 3 could easily be from the same distribution, resulting in an extremely low p-value. A more interesting result stems from the t-test conducted on Fig 4. With the *significant* outlying values instance "1" contains, the p-value skyrockets to the value seen in table **??**. The outlying values indicate that instances "1" and "2" of PSO plotted in Fig 4 are not part of the same distribution. As previously mentioned, these results are comprised of 50 values, ensuring statistical significance.

VI. Conclusions and Future Work

PSO is an incredible algorithm, and the fact that it's based on real-world swarm intelligence lends to it's awe factor. This implementation was successfully able to solve Ackley's function, while overcoming the extinction events and movement restrictions imposed by a higher-power.

TABLE I: This table depicts a multitude of tests performed testing the modifications to PSO

EC	EP	c_1	c_2	w	$VELOCITY\_MAX$	Mean Iterations	Mean Best
0.9	0.9	0.1	0.1	0.8	0.0	10000.0	0.005941
0.9	0.1	0.1	0.1	0.8	0.0	157.267	0.0
0.1	0.9	0.1	0.1	0.8	0.0	10000.0	0.000085
0.1	0.1	0.1	0.1	0.8	0.0	273.067	0.0
0.9	0.9	0.1	0.1	0.8	0.25	10000.0	0.010868
0.9	0.1	0.1	0.1	0.8	0.25	811.467	0.003560
0.1	0.9	0.1	0.1	0.8	0.25	10000.0	0.000101
0.1	0.1	0.1	0.1	0.8	0.25	270.0	0.0

TABLE II: This table shows the parameter values for Figs 5 - 10

$EXTINCTION\_CHANCE$	$EXTINCTION\_PERCENTAGE$	Clamp value where applicable
0.1	0.1	0.25

TABLE III: This table shows the parameter values for Figs 11 - 16

$EXTINCTION\_CHANCE$	$EXTINCTION\_PERCENTAGE$	Clamp value where applicable
0.9	0.9	0.25

Revisiting this problem in the future, further clamping methods could be tested, along with neighbourhoods or islands implemented as a way to further diversify the particles.

REFERENCES

[1] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[2] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[3] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 101:48, 2005.

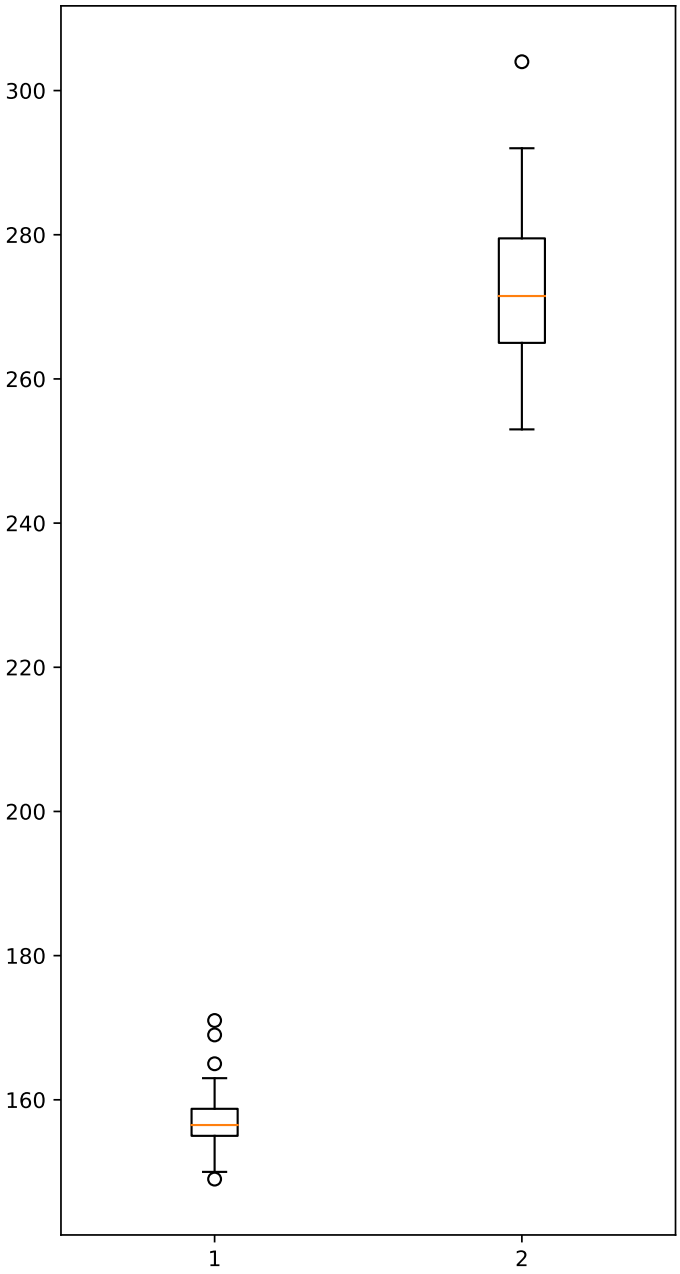

Fig. 3: Results with no velocity clamping

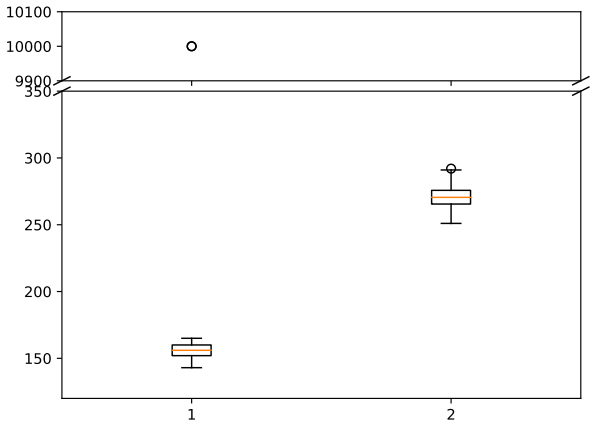

Fig. 4: Results with velocity clamping

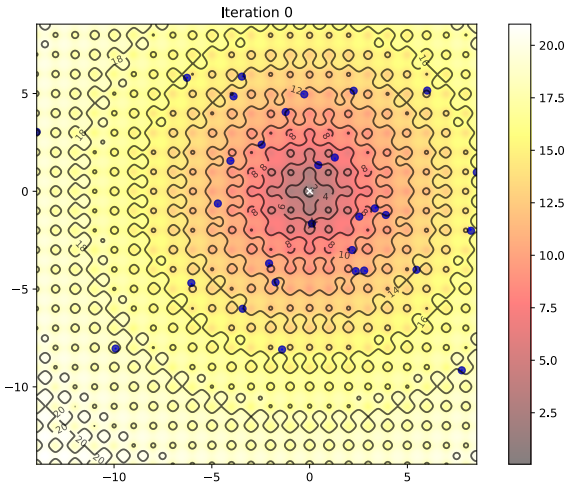

Fig. 5: Iteration 0, Low Extinction, No clamp

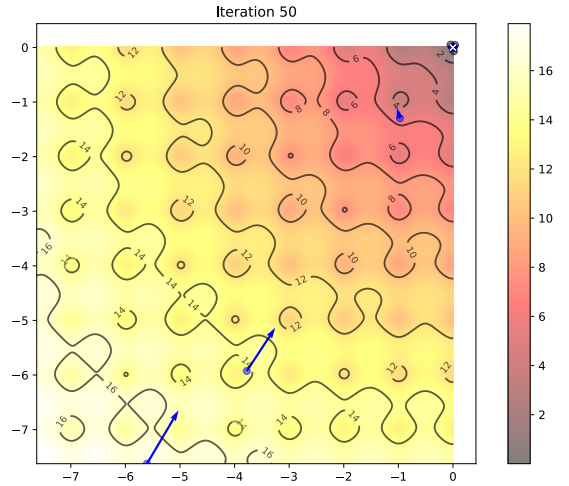

Fig. 6: Iteration 50, Low Extinction, No clamp

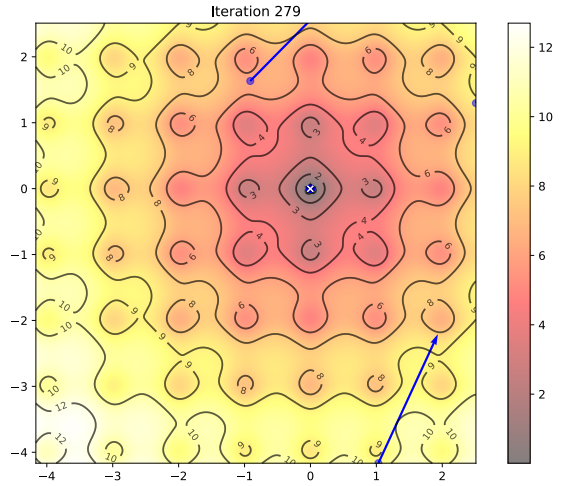

Fig. 7: Final Iteration, Low Extinction, No clamp

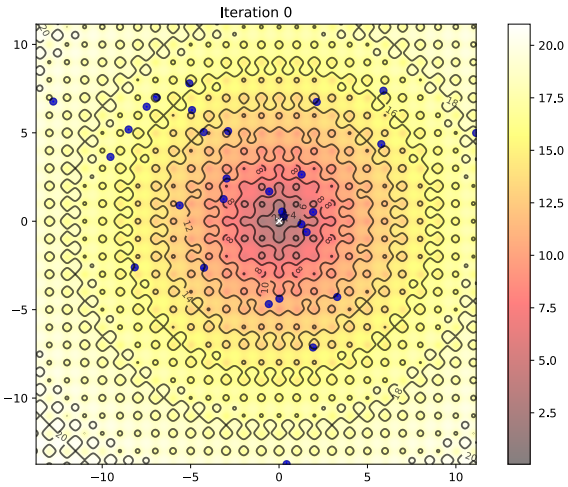

Fig. 8: Iteration 0, Low Extinction, Clamp = 0.25

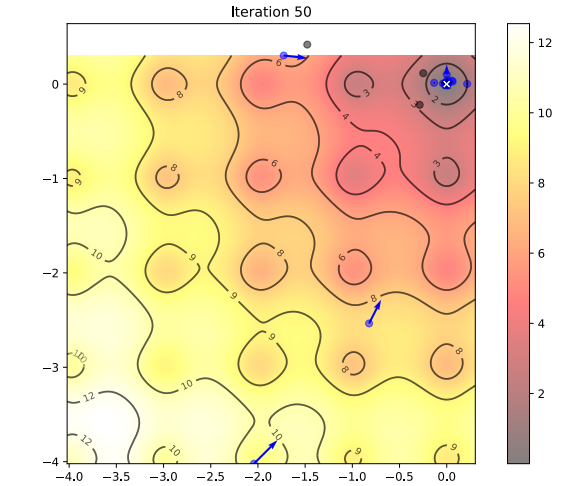

Fig. 9: Iteration 50, Low Extinction, Clamp = 0.25

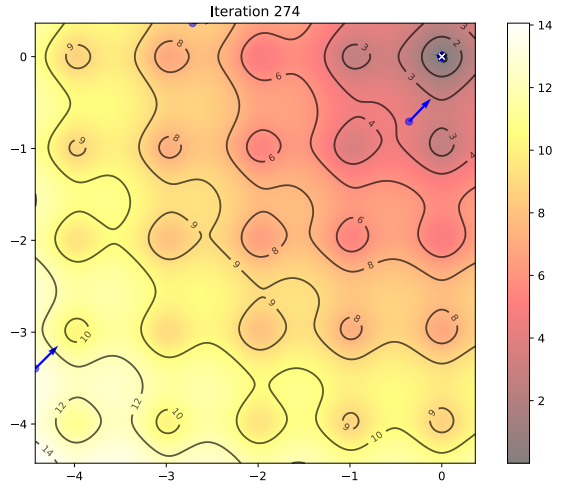

Fig. 10: Final Iteration, Low Extinction, Clamp = 0.25

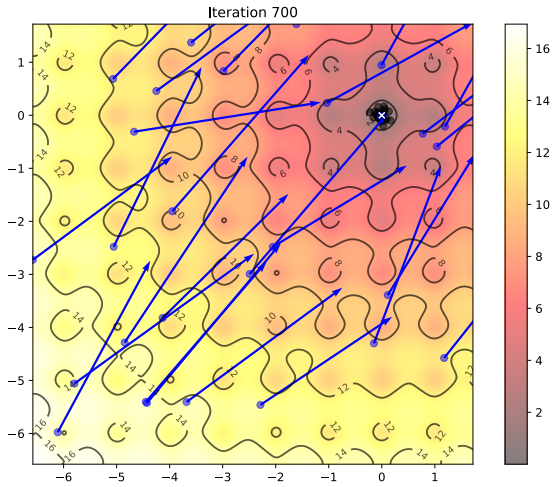

Fig. 11: Iteration 700, High Extinction, No clamp

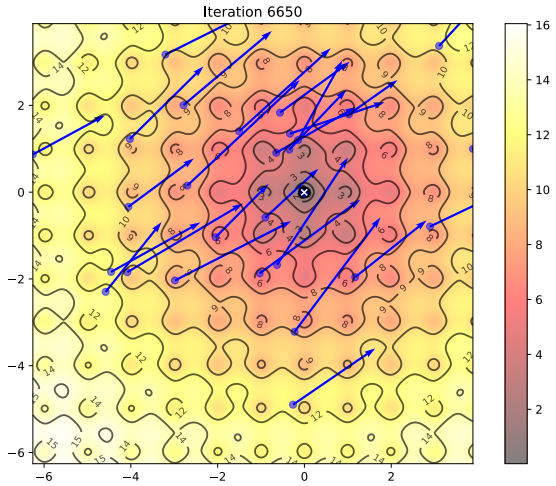

Fig. 12: Iteration 6650, High Extinction, No clamp

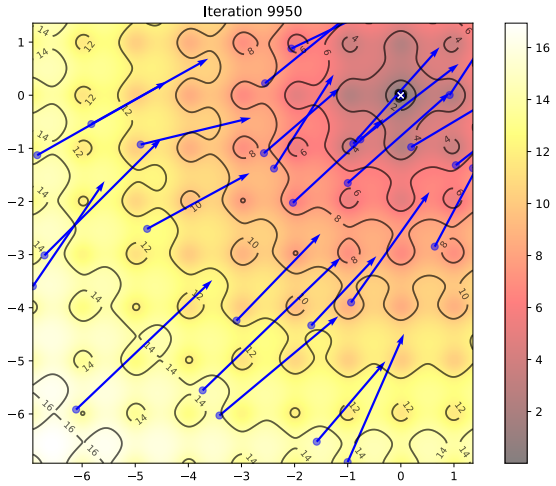

Fig. 13: Final Iteration, High Extinction, No clamp

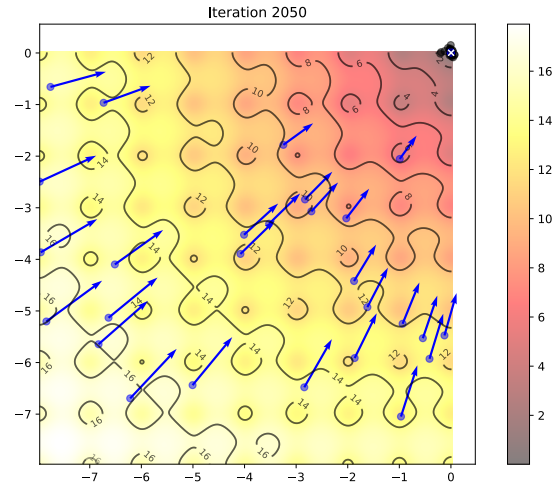

Fig. 14: Iteration 2050, High Extinction, Clamp = 0.25

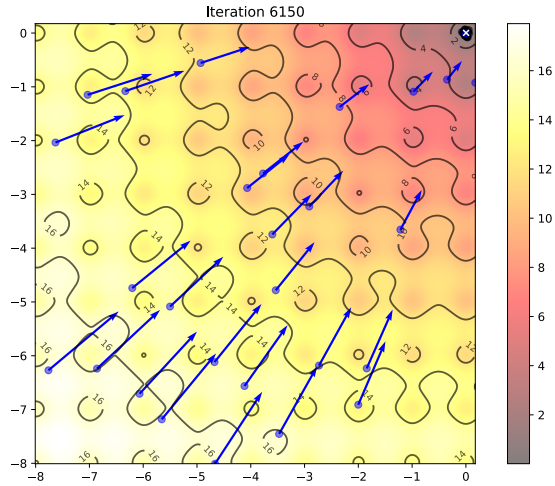

Fig. 15: Iteration 6150, High Extinction, Clamp = 0.25

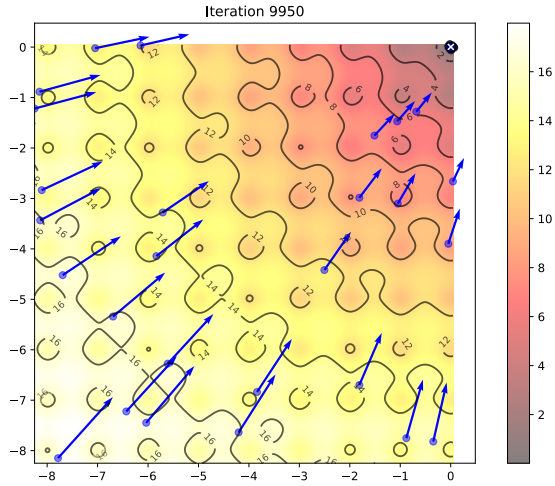

Fig. 16: Final Iteration, High Extinction, Clamp = 0.25