Artificial Intelligence 791 Assignment 1 Option 2

Is velocity clamping necessary for particle swarm optimization?

Name: S Boshoff
Student Number: 22546510
Email: 22546510@sun.ac.za

***Abstract*—** Particle Swarm Optimization (PSO) is a good method for solving optimization problems in multiple dimensions, however, it can occur that particles diverge and find sub-par solutions due to extreme velocities. A method that can be used to mitigate this problem, is to limit or "clamp" the velocities of these particles with the hopes of obtaining better solutions. The goal of this study is to find whether velocity clamping has a significant effect on the operation of PSOs, and if it should be used when solving optimization problems. This will be achieved by running several simulations under different conditions, comparing the effectiveness of the methods used to solve the optimization problems and determining efficiency.

Keywords—PSO, convergence, clamping, swarm

I. Introduction

Particle Swarm Optimization (PSO) in Computer Science is a computational method that optimizes a problem by iteratively trying to improve a candidate solution regarding a given measure of quality [1]. It involves several homogenous or heterogeneous particles that traverse a search space, that exhibit personal as well as social behaviour. This allows for solving of multi-dimensional problems with search spaces that vary greatly. PSOs are implemented by spreading particles across a search space and allowing the particles to explore and exploit until a solution is found.

The primary method of PSO execution is by repeatedly updating each particle position by some velocity, which is dependent on the inertia of the particle, the distance to its personal best solution, and the distance to the global best solution [2]. For each iteration, particles move around in n-dimensional space, find more optimal solutions, and then converge to the best possible position in the search space. A stochastic component is added by scaling each of the personal and social components by some uniform random number between 0 and 1. This allows the PSO to be non-deterministic and allows for unique behaviour within the system.

When working with PSO operation, a trade-off needs to be made between exploitation and exploration of the search space by the particles. Exploitation is the ability of a particle to concentrate on finding the optimal solution within the space nearby. Exploration is the ability of a particle to move around the search space and find better candidate solutions [3]. Unfortunately, the trade-off between these behaviours does not always result in good solutions. It was discovered that for certain behaviour parameters, the particles in the PSO never converge to a solution, and instead spiral out of control infinitely.

This creates the hypothesis for the experiment, as the goal is to find which of the available methods, work best to mitigate the divergence of particles. The proposed solution, which will be tested against a convergence condition, is velocity clamping. Velocity clamping is a method in which the particles have a maximum velocity that is scaled to the span of the search space, as well as with tightness parameters (K).

In this experiment, a comparison will be drawn between the use of velocity clamping and only using converging parameters. This comparison will be made on the grounds of Diversity, Magnitude, Quality, and Percentage of particles that have left the search space, over time. It will then be determined if velocity clamping is necessary.

II. Background

When working with PSOs, it is common to see divergent behaviour, meaning particles constantly accelerate further away from each other. This leads to underwhelming results and is a significant problem that needs to be mitigated. A relationship between the values of w, c1, and c2 is found and shown to be the cause of this divergence. The works of Cleghorn and Engelbrecht proposed a mathematical condition that needs to be met, in order for the PSO to converge to a point [2].

The convergence condition:

$$c_1 + c_2 \leq \frac{24(1 - w^2)}{7 - 5w} \tag{1}$$

The effects of having w, *c1* and *c2* values that do not satisfy this condition can easily be seen be comparing the following two cases, where w is kept constant, but *c1* and *c2* values are adjusted to no longer satisfy the condition. In

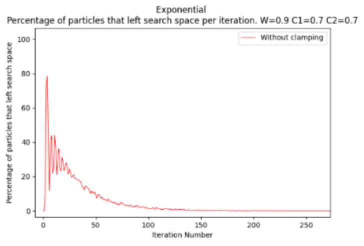

FIGURE 1

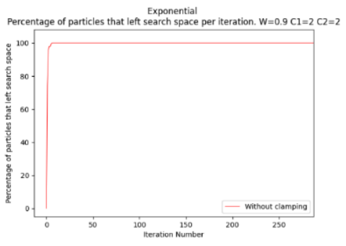

FIGURE 2

Figure 1, some particles leave the search space and then return a few iterations later when there is convergence to the global minimum, however in Figure 2, we see that for unclamped particles, the solution space is left quickly, and particles never return.

The necessity of the convergence condition being met is further shown by the following table, where the prescribed parameters are tested for convergence. If PSA parameters do not meet the convergence condition, particles diverge.

TABLE 1. CONVERGENCE CONDITION EFFECT

W	C1	C2	Satisfies convergence condition?	Particles diverge?
1	2	2	No	Yes
0.7	1.4	1.4	Yes	No
0.9	2	2	No	Yes
0.9	0.7	0.7	Yes	No

Another approach of eliminating divergence is Velocity Clamping, where particles cannot exceed a predetermined velocity. The following formula is used for velocity clamping:

$$v_{max,i} = k * (x_{max,i} - x_{min,i}), \forall i = 1, \dots, n_x$$
$$with\ k \in \{0.1, 0.3, 0.5\} \quad (2)$$

The proposed solution is that for a certain value of *k*, the particles will be bound inside the feasible search space, and thus forced to converge to a solution. As for the level of clamping, the smaller value k possesses, the tighter the clamping is applied, and the smaller step sizes a particle can make. This approach leads to particles having more exploitative behaviour, at the cost of not exploring the search space adequately.

In order to evaluate the performance of both velocity clamping and the effectiveness of the convergence condition, a simulation is run where PSOs with different characteristics solve functions in multi-dimensional space. The results of these simulations are then used to determine which approach is more optimal.

III. METHODOLOGY

The data obtained in this experiment is calculated with a Python3 script along with libraries Numpy, a powerful library that allows for fast matrix and vector calculations, and MatPlotLib PyPlot, a graphing tool that allows for the informative display of graph plots. In hindsight, a faster language such as Java or C should have rather been used as they are more optimized and can calculate the massive amounts of values at a much quicker rate.

The program is written in such a way that the only variable that needs to be changed between tests is the function in question, as well as the minimum and maximum points in the feasible space. When the code is run, it generates a total of 4 graphs pertaining to the function selected – the graphs describe the mean Diversity (1), Magnitude (2), Quality (3) as well as the Percentage of particles that have left the search space (4), over a set number of iterations. PyPlot allows for the graph scales to be edited before it is stored, which allows the user a great level finetuning with regards to the final result.

An attempt at multithreading was also made, however the results produced were not up to standard. It is therefore recommended that the user run four instances of the program manually, each with a different parameter selection.

A. *Initialization*

Each particle is created as a Python object, as opposed to creating a multi-dimensional array containing the positional data, velocity, and personal best coordinates. This is done for the specific purpose of treating each particle as its own entity, and therefore creating a base understanding of how for instance a bird in a flock would operate. Therefore, the function to update the position of a particle can be located in the particle itself, and all necessary parameters are parsed as to allow the particle to make calculations independently.

Global variables:

1. Set global best position to the 0 vector and global best position to the maximum integer.
2. Initialize various lists that receive test output.

Particle Object:

1. Receives random position in search space when constructed.
2. Evaluates the position and initializes it as personal best position and value.
3. If personal best is better than global best, update global best position and value.
4. Assigns whether velocity is clamped and to what degree.

PSO Simulator:

1. Create an initial random position vector with *random.uniform(MIN, MAX)* in every dimension for every particle.
2. Create a zero vector in the dimension of the search space – used as the initial velocity.
3. Call the particle constructor application and parse the position and velocity vectors, as well as the benchmark function and clamping parameters.
4. Store the particle in the list.

B. *Run-time*

Within the PSO Simulator, a for loop is used to run position updates for every iteration. After each iteration, the state of the system is collected and stored various lists. At the end of the loop, a Dataframe variable is created consisting of all relevant lists and parsed back to the main operating function.

Pseudo code for the position updates are as follows:

1. For every iteration, call the update position function within each particle. A particle then executes the following:
   a. Calculates new velocity based on the PSO formula given.
   b. If the particle makes use of velocity clamping, check if the velocity is over the limit in both positive and negative directions. In the case that the limit is exceeded, the velocity should be clamped respectively.

c. Update the position of itself by adding the new velocity vector to the current position.

d. Evaluate the new position and determine if obtained a greater level of optimality than the current personal and global bests. If it did, update the respective values only if the particle is in the valid search space.

2. After all particles are updated, evaluate all characteristics and store in lists.

After all iterations are completed and the DataFrame variable is returned to the main function and graphs are ready to be made.

C. *Process of generating graphs*

After the PSO simulations have concluded, the program calculates the averages of each of the 20 runs for the various attributes of interest. A plot is then created which displays the data in lines corresponding to unclamped, clamped k = 0.1, clamped k = 0.3, and k = 0.5 PSOs respectively. The graphs are saved automatically but can also be modified.

IV. EMPIRCAL PROCEDURE

To evaluate the optimality of a PSO, an experiment needs to be constructed to create the particles and manage their movements throughout the search space. To create a comparison between clamped and non-clamped velocity PSOs, a simulation of particles over ***N*** iterations is used. The particles are tasked with finding the global minimum of different functions that each span ***D*** dimensions.

For this experiment, the following values are used:

- ***N*** = 5000
- ***D*** = 30
- ***Number of Particles*** = 30
- ***Number of simulated runs*** = 20

TABLE 2. FUNCTIONS CHOSEN FOR THE EVALUATION OF PSO BEHAVIOUR.

Function Name	**Function Equation**	**Function Domain**	**Function Minimum**
Exponential	$f(x) = -\exp\sum_{i=1}^{D}(x_i)^2$	$-1 \le x_i \le 1$ for all $i = 1,\cdots,d$	$x* = f(0,\cdots,0)$ $f(x*) = -1$
Schwefel	$f(x) = 418.9829d - \sum_{i=1}^{d} x_i sin\left(\sqrt{\lvert x_i \rvert}\right)$	$-500 \le x_i \le 500$ for all $i = 1,\cdots,d$	$x^* = f(420.9687,\ldots,420.9687)$ $f(x^*) = 0$
Qing	$f(x) = \sum_{i=1}^{D}(x_i^2 - i)^2$	$-500 \le x_i \le 500$ for all $i = 1,\cdots,d$	$x^* = f(\pm\sqrt{i})$ $f(x^*) = 0$
Rosenbrock	$f(x) = \sum_{i=1}^{D-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$-30 \le x_i \le 30$ for all $i = 1,\cdots,d$	$x* = f(1,\cdots,1)$ $f(x^*) = 0$
Brown	$f(x) = \sum_{i=1}^{n-1}(x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)}$	$-1 \le x_i \le 4$ for all $i = 1,\cdots,d$	$x* = f(0,\cdots,0)$ $f(x*) = 0$

[5]

A. *Particle Velocity Update*

When this function is called for a specific particle, it calculates the velocity for the next iteration with the following formula:

$$v_{ij}(t+1) = w \cdot v_{ij}(t) + c1 \cdot r_{1j}(t) \cdot \left(y_{ij}(t) - x_{ij}(t)\right) + c2 \cdot r_{2j}(t) \cdot \left(\hat{y}_j(t) - x_{ij}(t)\right) \tag{3}$$

B. *Variable definition*

- $\boldsymbol{v_{ij}}$: The velocity of the i^{th} particle, in the j^{th} dimension at iteration *t*.
- ***w***: The inertia component multiplier.
- ***c1***: The cognitive component multiplier.
- ***c2***: The social component multiplier.
- $\boldsymbol{x_{ij}}$: The current position of the i^{th} article in the $\boldsymbol{j^{th}}$ dimension at iteration *t*.
- $\boldsymbol{y_{ij}}$: The personal best position of the i^{th} particle in the j^{th} dimension at iteration *t*.
- $\boldsymbol{\hat{y}_j}$: The global best position in the j^{th} dimension at iteration *t*.
- $\boldsymbol{r_{1j}}$ & $\boldsymbol{r_{2j}}$: A uniform random value between 0 and 1 in the j^{th} dimension at iteration *t*.

The position of the particle is then updated by the following addition:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{4}$$

This adds the velocity to the particle position in every dimension, to move the particle to the next position.

C. *Mean Particle Diversity*

This function has the purpose of returning a scalar value that indicates the mean distance of all particles to the particle center of mass. The formula used for this:

$$Mean\ swarm\ diversity\ = \frac{1}{N}\sum_{i=1}^{N} \parallel \bar{x}_i - \frac{\sum_{j=1}^{N}\bar{x}_j}{N} \parallel \tag{5}$$

The Numpy function linalg.norm performs this calculation.

D. *Mean Particle Magnitude*

The mean particle magnitude is found by calculating the sum of the lengths of all velocity vectors across particles. The formula is as follows:

$$Mean\ particle\ magnitude = \frac{1}{N}\sum_{i\,=\,0}^{N} \lVert\vec{v}_i\rVert \tag{6}$$

The Numpy function linalg.norm performs this calculation.

E. *Particle Quality*

The quality of a solution is quite trivial to calculate. Formula:

$$Q = \hat{y} - min(function) \tag{7}$$

F. Percentage of particles that have left the search space boundaries

For this, every particle is evaluated individually, and if its position in a dimension is outside of the bounds, we mark that particle as departed and then count all the departed particles at the end. Formula:

$$\%\ of\ particles\ that\ have\ left\ the\ search\ space = \frac{1}{N}\sum_{i=1}^{N} u_i$$

$$u_i = \begin{cases} 0\ if\ particle\ is\ in\ search\ space \\ 1\ if\ particle\ left\ search\ space \end{cases} \tag{8}$$

G. Control Parameters

TABLE 3. PSO VELOCITY PARAMATERS

W	C1	C2
1	2	2
0.7	1.4	1.4
0.9	2	2
0.9	0.7	0.7

V. RESEARCH RESULTS

TEST RESULTS FOR NON-CONVERGING PARAMETERS

The following results are obtained by testing the PSOs with parameters that do not satisfy the convergence conditions. Figures are selected based on how accurately and informatively they reflect the state of the PSO.

A. Particle Diversity

Under the conditions that the PSO parameters do not satisfy the convergence condition, the general form of the graphs can be observed. The unclamped PSO usually gains enormous momentum in the first few iterations, and particles spread out over the infeasible search space almost infinitely and never stop increasing.

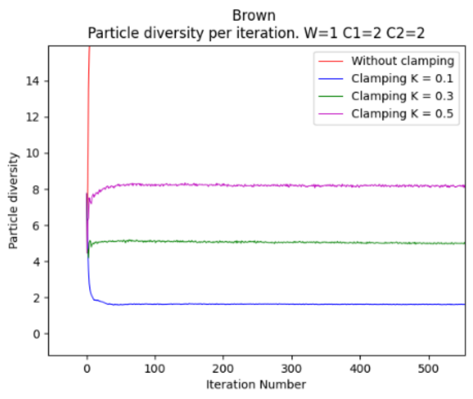

FIGURE 3

Within these graphs, compared to the unclamped PSO, the clamped PSO diversity becomes stable at certain points. Even though this stability is not achieved at 0 diversity, which would imply that all particles converged, it still shows how Velocity Clamping can aid in limiting the vast explosion of particles thus allowing the PSO to perhaps find a better solution.

B. Particle Magnitude

Extreme particle magnitude is the main cause of the divergent behavior seen in the PSO. If the velocity variable in a particle becomes too large, the step sizes of the particle

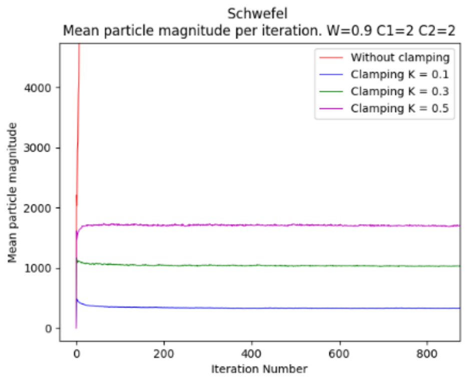

FIGURE 4

become uncontrollable, and the particle experiences rapid acceleration almost immediately, as can be seen in figure 5. If velocity clamping is applied, particles are not allowed to reach a velocity greater than defined bound (2) and stay within reasonable parameters. As can be seen in the figures, the tightest clamping (K = 0.1) has the lowest mean magnitude, and the loosest clamping (K = 0.5) has the highest. Important to note is that none of the particle magnitudes ever reach 0, which implies that swarm will always be making significant movements within.

C. Particle Quality

Particle quality for suboptimal parameters exhibited similar results for most of the testing, with the tightest clamping usually finding the best solution of all approaches. This could be because the particles explore more of the feasible search space, and then forced to find better

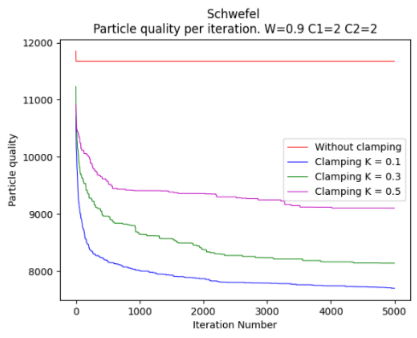

FIGURE 5

solutions. The results overall indicate clear examples of utility that velocity clamping provides, allowing PSOs to reach the optimal solution in certain cases. When a particle leaves the feasible space with divergent parameters, the likelihood of returning and finding a better solution is extremely small.

TABLE 4. RANKING OF QUALITY PER CLAMPING METHOD.

(LOWER IS BETTER)

Function	Unclamped	Clamped K = 0.1	Clamped K = 0.3	Clamped K = 0.5
Exponential	4	1	2	3
Schwefel	4	1	2	3
Qing	4	1	2	3
Rosenbrock	4	1	2	3
Brown	4	1	2	3

D. *Percentage of particles that left the valid search space*

The results show that under suboptimal parameters, unclamped particle velocity leads to particles leaving the

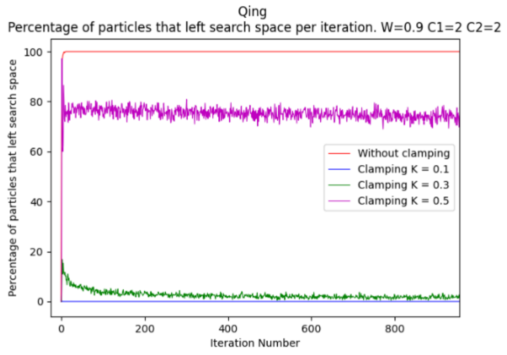

FIGURE 6

search space almost immediately. As can be observed with the Brown Function (Figure 6), only the tightest clamping resulted in less than 95% of leaving the search space. Even under these conditions, the tightest clamping does not always stay within the search space. For certain functions that are easier for the PSO to solve, all forms of clamping show an effect (see Figure above) and can lead to some particles never leaving the search space. More inconsistency is showed with regards to clamping, as it cannot be trusted to behave the same for every function.

E. *Summary of non-converging paramater tests*

The unclamped PSO obtained the worst solution for every single function, and since it is known that it constantly diverges, a conclusion can be drawn that for most functions, an unclamped PSO will perform worse than a clamped one - when parameters do not satisfy the convergence condition. The degree of clamping is also significant. Bad parameter values lead to rapid acceleration of a particle, thus making it leave the search space almost immediately and removing the possibility of finding better solutions. Therefore, clamped particles are forced to remain and explore within the search space, and this leads to better solutions.

TEST RESULTS FOR CONVERGING PARAMETERS

The main measure of the quality of a PSO is that it can find the best solution to the problem, in the smallest number of iterations. So ideally these two aspects need to be minimized with regards to PSO execution. In this section of the results, a comparison will be made between clamped and non-clamped PSOs, when the parameters satisfy the convergence condition.

A. *Particle Diversity*

With converging PSOs, an indicator that a solution has been found is when the particle diversity is near zero, which translates to all particles being in the same position. In theory, when the convergence conditions are met, particles should converge to a single position for every function in most cases (Figure 7). Over the functions tested, a clear pattern arises that points to the effect of clamping on the diversity of particles. Clamped particles can be seen to

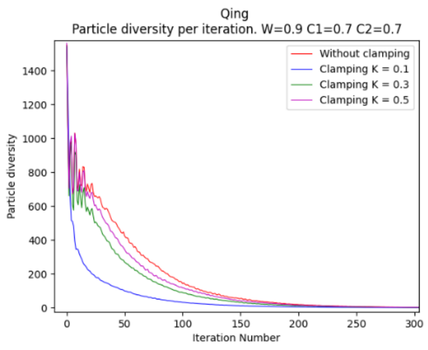

FIGURE 7

converge faster, due to limited search space exploration. The Schwefel function is seen to provide strange results, with unclamped particles sometimes never converging. The Schwefel function is notoriously difficult for PSOs to solve, since it has many local minima which create opportunity for particles to get stuck. However, the average behavior

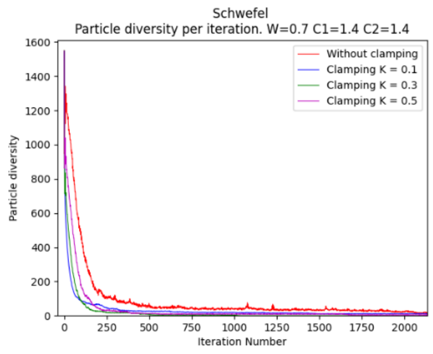

FIGURE 8

between the functions is that clamping K = 0.1 converges first, there-after K = 0.3, K = 0.5 and lastly unclamped. An interesting observation is the secondary peak a few iterations into the experiment. This illustrates the explorative behavior of PSOs since the particles do not always converge and may move away to search for better solutions.

B. *Particle Magnitude*

As previously noted, a near zero value magnitude with respect to converging PSOs implies that particles have

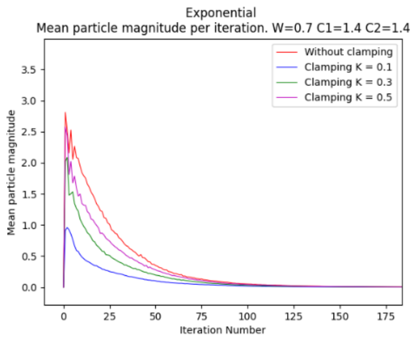

FIGURE 9

stopped moving i.e., no longer making significant velocity updates.

Figure 9 sufficiently illustrates the effect of clamping since the clamped particles do not reach the same levels as unclamped, as well as reaching lower peaks depending on how tight the clamping is applied. These graphs look nearly identical for every single function that has been tested.

C. *Particle Quality*

The previous section shows that clamped particles converge faster than unclamped particles under the specific conditions, but these results do not imply that velocity clamping is more optimal. For this, a PSO needs to obtain a solution of the highest quality. After testing particle quality,

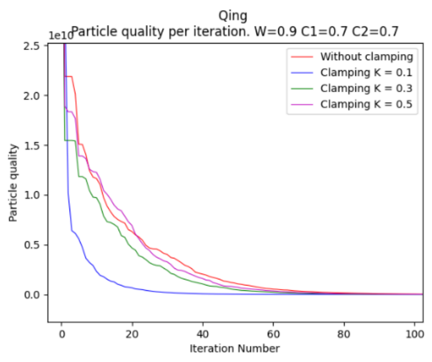

FIGURE 10

Figure 10 shows that the Qing Function optimal solution is obtained roughly 50% faster by the tightest clamped PSO, and that the other clamping values along with unclamped

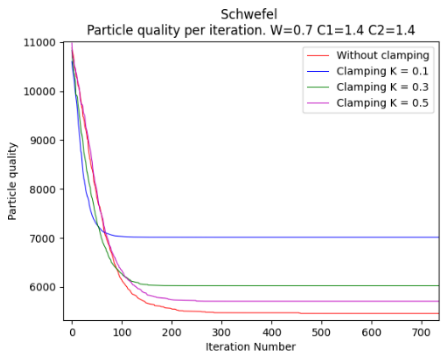

FIGURE 11

perform the same. But Figure 11 illustrates completely different results, with the unclamped PSO obtaining the best minimum over 20 runs. This points to clamped PSOs performing better when optimization problems have fewer local minima. The difference between the Qing and Schwefel function shows this:

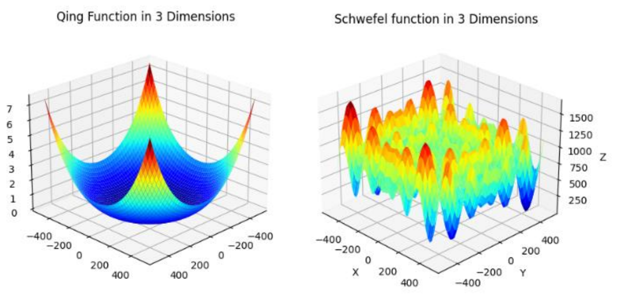
Qing Function in 3 Dimensions
Schwefel function in 3 Dimensions
X
Y
Z

FIGURE 12
QING AND SCHWEFEL FUNCTIONS PLOTTED IN PYTHON

The Schwefel Function is clearly a lot more complex, and requires as much exploration possible to find the optimal point. Neither the unclamped nor clamped PSOs find the global minimum of the Schwefel in 30 dimensions, which further enforces the difficulty of solving the problem.

D. *Percentage of particles that left the valid search space*

Ideally, converging PSOs should in theory have close to zero particles that have left the search space after a solution is found.

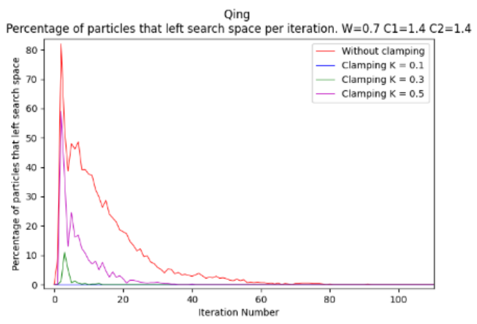

Figure 13 proves the statement above, as the percentage of

FIGURE 13

particles that left the search space after sufficient iterations is near zero. The unclamped particles tend to leave the search space in high numbers at the start of the simulaation, and then return to converge to a feasible point. The clamped particles tend to behave exactly the same, but the initial spike is less harsh. For some functions the clamped PSO with K = 0.1 never has particles that leave the search space.

E. *Summary of test results for converging parameters*

TABLE 5. CLAMPED AND UNCLAMPED PERFORMANCE

	Clamped	Unclamped
Obtained minimum (Out of 10)	8	10
Converged the fastest (Out of 10)	10	0

When observing the number of times that each PSO obtained the minimum and which PSO converged the fastest, the result is that a PSO that makes use of velocity clamping, converges faster than an unclamped PSO under all test cases. However, clamped PSOs do not always find the best quality solution, and working with high complexity functions, usually amplify this effect. It is therefore recommended that in order to find the best feasible solution, to not make use of velocity clamping when the convergence conditions are met.

VI. CONCLUSION

Is velocity clamping necessary? After simulating several different PSOs with various parameters and clamping values, we find that in most cases, velocity clamping is not necessary when the convergence conditions are met. The utility of velocity clamping, however, is not completely redundant, as in some cases, it allows the PSO to converge faster. This is not seen as a sufficient condition to rule velocity clamping as optimal but is worthy of note. When the convergence conditions are met, particles are allowed to perform the necessary exploration in order to find the global minimum, as well as to regulate velocity in such a manner that divergence is rare. Thus, we conclude that velocity clamping is not necessary for Particle Swarm Optimization.

REFERENCES

[1] Bonyadi, M. R Michalewicz, Z. (2017). "Particle swarm optimization for single objective continuous space problems: a review". Evolutionary Computation. 25 (1): 1–54.

[2] A.P. Engelbrecht, “Computational Intelligence, An Introduction”, Second Edition, “Particle Swarm Optimization”, pp. 323-333, 2007.

[3] C.W. Cleghorn and A.P. Engelbrecht, “Particle Swarm Variants: Standardized Convergence Analysis”, Equation 18, pp. 8, 2006.

[4] M. Jamil and X. Yang, “A Literature Survey of Benchmark Functions For Global Optimization Problems”, pp. 5-28, 2013.

[5] Indusmic Private Limited, “Schwefel Function Plot”, https://www.indusmic.com/post/schwefel-function, 2021