VIETNAMESE-GERMAN UNIVERSITY

(CSE2020 AND ECE2021)

FINAL REPORT of GSA AND PSO FOR MPPT

Nguyễn Đình Bảo Khang - 10221062
Nguyễn Khắc Hoàng -18230
Phùng Quang Nhật - 10221071

Supervisor: Dr. Bui Minh Duong

Binh Duong Province, Viet Nam, February 2024

ABSTRACT

The rapid global population growth has caused a surge in electricity demand, driving increased research into renewable energy sources. Among these, solar cells are particularly popular. These cells have a characteristic curve with an optimum operating point for maximum power output, a point affected by sunlight intensity, temperature, and cell properties. To reduce solar electricity costs, real-time tracking of the Maximum Power Point (MPP) is crucial for efficient systems. Many techniques exist for this, and this paper investigates the Gravitational Search Algorithm (GSA) in a MATLAB environment for tracking the MPP of a solar cell array. The results are then compared to another evolutionary algorithm, Particle Swarm Optimization (PSO).

Keywords: *Gravitational Search Algorithm (GSA), Particle Swarm Optimization (PSO), Maximum Power Point Tracking (MPPT), Solar.*

PREFACE

In recent years, there has been a significant increase in the use of photovoltaic (PV) systems for electricity generation. This growth has been driven by the rising global demand for clean and sustainable energy sources. The environmental benefits, cost reductions, and advancements in solar energy technology have made PV technology the leading trend in the global energy transition. However, it's important to recognize that the performance and efficiency of PV systems are influenced by various environmental factors and operating conditions, which can have a significant impact on their power generation capabilities.

One of the main challenges in maximizing the output of PV systems is accurately tracking the Maximum Power Point (MPP). The MPP represents the operating point that yields the highest power output from the solar panels. As solar irradiance, temperature, shading, and other factors fluctuate throughout the day, the MPP of the PV system also changes. To address this challenge and optimize energy harvesting from PV arrays, efficient and accurate Maximum Power Point Tracking (MPPT) algorithms are necessary.

MPPT algorithms play a crucial role as the "brains" of PV systems. They continuously monitor operating conditions and adjust voltage and current to ensure that solar panels operate at their MPP. By flexibly adjusting the electrical load to match different environmental conditions, MPPT algorithms enable PV systems to extract the maximum available power, thereby improving overall energy conversion efficiency and output.

In this report, we focus on the development and evaluation of an improved MPPT system that utilizes the Gravitational Search Algorithm (GSA). GSA is an optimization algorithm inspired by the principles of gravitational force and mass interaction. It mimics the behavior of celestial bodies in space to search for optimal solutions in multidimensional search spaces. GSA has shown promising results in various optimization problems and has the potential to enhance the performance of MPPT in PV systems.

The main objective of this research is to introduce a GSA-based approach for MPPT and evaluate its effectiveness in achieving maximum power output from PV arrays under different operating conditions. The report begins by providing a comprehensive overview of MPPT algorithms, explaining their fundamental principles, and discussing the challenges associated with MPP tracking. It then delves into the GSA algorithm, describing its basic concepts, mathematical formulas, and search mechanisms.

Furthermore, the report presents detailed information about the design and implementation of the proposed GSA-based MPPT system. It highlights the key steps involved in integrating the GSA algorithm into the MPPT framework, including parameter setting, overall initialization, fitness evaluation, and search strategy. The hardware and software architecture of the system are also described, providing deep insights into its practical deployment.

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to Dr. Bui Minh Duong for his unwavering guidance throughout the project, and to Teaching Assistants Le Hoang Khang and Nguyen Thanh Tung for their invaluable assistance. I am also immensely grateful to all my friends and family who supported me throughout this study.

CONTENTS

ABSTRACT 2
PREFACE 2
ACKNOWLEDGEMENTS 4
CONTENTS 5
- I. Tables of Figure 6
- II. Tables of Tables 7
- III. Body of the report 7
  - 1. Introduction 7
  - 2. Methodology 8
  - 2.1 What is Maximum Power Point Tracking (MPPT)? 8
  - 2.2 MPPT System Model Overview 9
  - 2.3 Gravitational Search Algorithm (GSA) 9
  - 2.4 Particle Swarm Optimization (PSO) 11
- 3. Implementation 11
  - 3.1 The PV System 12
  - 3.2 PSO implementation 15
  - 3.3 GSA implementation 21
- 4. Analysis 28
  - 4.1 Finding 29
  - 4.2 Discussion 31
- 5. Conclusion 33

IV. Appendix 34
V. Reference 39

I. Tables of Figure

Figure	***Name***	***Page***
1	*Block diagram of PV system*	*12*
2	*PV system implementation*	*13*
3	*MPPT algorithms integration*	*14*
4	*Block design for Visualization*	*15*
5	*Flow of PSO algorithms*	*16*
6	*Initialization of PSO particles*	*17*
7	*Iteration loop implement (PSO)*	*18*
8	*Local optimum avoidance (PSO)*	*18*
9	*Locating global best (PSO)*	*19*
10	*Particle updates (PSO)*	*19*
11	*Velocity updates (PSO)*	*20*
12	*Duty updates (PSO)*	*20*
13	*Flow of GSA algorithms*	*21*
14	*Initialization of GSA particles*	*22*
15	*GSA iteration loop*	*23*
16	*Update best and worst fitness (GSA)*	*24*
17	*Mass function computation (GSA)*	*25*
18	*G calculation (GSA)*	*25*
19	*Force computation (GSA)*	*26*
20	*Particle position update (GSA)*	*26*
21	*Distance function (GSA)*	*27*
22	*Velocity function (GSA)*	*27*
23	*Update Duty function (GSA)*	*28*
24	*Power pattern between GSA and PSO*	*29*
25	*Duty pattern of GSA*	*30*
26	*Duty pattern of PSO*	*30*

II. Tables of Tables

Table	*Name*	*Page*
1	*Performance of GSA and PSO*	*31*
2	*GSA and PSO characteristics*	*32*

III. Body of the report

1. Introduction

In the past few years, the field of optimization algorithms has made significant progress, offering various efficient techniques for solving complex optimization problems. One such algorithm that has garnered attention is the Gravitational Search Algorithm (GSA). Inspired by the principles of Newton's law of gravity and motion, GSA demonstrates promise in effectively addressing a wide range of optimization challenges.

Researchers have conducted a thorough analysis of recent variants of the Gravitational Search Algorithm, focusing on three key parameters: Kbest, velocity, and position. By modifying these parameters, they aimed to enhance the algorithm's performance and adaptability across different optimization scenarios. The analysis involved comparing the performance of ten GSA variants on standard functions and CEC2015 functions, which cover diverse optimization categories such as unimodal, multimodal, and unconstrained functions.

Throughout the analysis, researchers evaluated each variant based on mean fitness value and convergence graph, providing valuable insights into their exploration and exploitation capabilities. Among the variants, one particular variant known as IGSA showcased exceptional precision and a balanced trade-off between exploration and exploitation. Additionally, the study examined the performance of GSA variants using a triple-negative breast cancer dataset, specifically for nuclei segmentation.[1]

To gain a comprehensive understanding of GSA's role in optimization algorithms, it is crucial to explore related concepts such as Maximum Power Point Tracking (MPPT) and Particle Swarm Optimization (PSO). MPPT is a technique utilized in photovoltaic systems to maximize power output by continuously tracking the optimal operating point [2]. On the other hand, PSO draws inspiration from the collective behavior of bird flocking or fish schooling and is a population-based optimization algorithm [3].

Exploring the fundamentals and uses of GSA, MPPT, and PSO allows us to uncover significant knowledge about the progress in optimization algorithms and their ability to tackle intricate issues in diverse fields. This piece aims to offer a thorough examination of the Gravitational Search Algorithm and its latest adaptations, offering insights into their efficiency and possible practical uses.

2. Methodology

2.1 What is Maximum Power Point Tracking (MPPT)?

Maximum Power Point Tracking, or MPPT, is a crucial technique in solar systems. Simply put, MPPT is like a smart controller for solar panels. It fine-tunes the panel settings to maximize power output, ensuring we get the most energy from sunlight.

Imagine your solar system as a smart, adaptable powerhouse. The technology continuously monitors and adjusts the panel settings, optimizing them to operate at or near their maximum power output. This is particularly valuable in fluctuating conditions like cloud cover or shading.

The real magic happens as MPPT is commonly used in solar charge controllers, grid-tied inverters, and other solar power applications. This not only enhances overall system efficiency but also ensures we harvest more energy from available sunlight.

In a nutshell, MPPT is about making our solar systems smarter and more effective, boosting efficiency and performance in various weather conditions."

2.2 MPPT System Model Overview

The Maximum Power Point Tracking (MPPT) system is a crucial component in photovoltaic (PV) systems, designed to optimize the efficiency of solar energy harvesting.

The MPPT system model overview encompasses a sophisticated algorithmic approach that continuously monitors and adjusts the operating point of the PV modules to extract the maximum available power from the solar panels.

This involves dynamically tracking the point on the current-voltage (I-V) curve where the power output is at its peak, regardless of environmental factors such as varying light conditions and temperature.

The MPPT system employs sensors and control mechanisms to ensure precise adjustments, allowing the PV system to operate at its highest efficiency and deliver optimal electrical output.

This model overview highlights the significance of MPPT technology in enhancing the overall performance and energy yield of solar power systems.

2.3 Gravitational Search Algorithm (GSA)

The Gravitational Search Algorithm (GSA) is a metaheuristic optimization algorithm inspired by the law of gravity and the behavior of celestial bodies in the universe. It was proposed by Rashedi, Nezamabadi-Pour, and Saryazdi in 2009. The algorithm is designed to solve optimization problems by simulating the gravitational forces between masses.

The basic idea behind GSA is to represent potential solutions to an optimization problem as masses in a multidimensional space. The optimization process is then modeled as the movement of these masses under the influence of gravitational forces. Masses are attracted to each other, and their movements are guided by the gravitational forces they exert on one another.

The key components of the Gravitational Search Algorithm include

- Mass Representation: Each potential solution is represented as a mass in the search space.

- Gravitational Force: The gravitational force between two masses is calculated based on their masses and the distance between them. Higher masses experience stronger attractive forces, while greater distances result in weaker forces.

- Acceleration: The gravitational force is used to calculate the acceleration of each mass, which, in turn, determines its movement in the search space.

- Updating Positions: Masses update their positions based on the calculated accelerations, and the process is iteratively repeated to improve the solution.

The algorithm aims to converge toward optimal solutions by balancing the exploration of the search space and the exploitation of promising regions. GSA has been applied to various optimization problems, including engineering design, data clustering, and feature selection, among others.

Like other metaheuristic algorithms, the effectiveness of GSA depends on various factors such as parameter settings, problem characteristics, and the nature of the optimization landscape. Researchers continue to explore and enhance the performance of the Gravitational Search Algorithm and its variants for different types of optimization problems.

2.4 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a nature-inspired optimization algorithm that is used to find approximate solutions to optimization and search problems. It was developed based on the social behavior of birds and fish, where individuals in a group (particles) collaborate to find the optimal solution to a problem.

The basic idea behind PSO is to model a swarm of particles moving through a search space, where each particle represents a potential solution to the optimization problem. These particles adjust their positions and velocities based on their own experience and the experiences of their neighboring particles. The algorithm is guided by the principle of collective intelligence, where the particles communicate and share information to improve the overall performance of the swarm.

The movement of particles in the search space is influenced by their own experience (exploitation) and the experience of the entire swarm (exploration). This balance between exploration and exploitation helps PSO to efficiently search the solution space for optimal or near-optimal solutions.

PSO has been successfully applied to various optimization problems, including function optimization, parameter tuning, and neural network training, among others.

3. Implementation

3.1 The PV System

The following detailed overview of the MPPT system implementation using MATLAB's Simulink reflects the architecture and the operational dynamics of the system as captured in the provided schematics. The figure 1 below represents the structured approach to the inspiration and integration of the MPPT, focusing on the fundamental components and their interplay within the system.

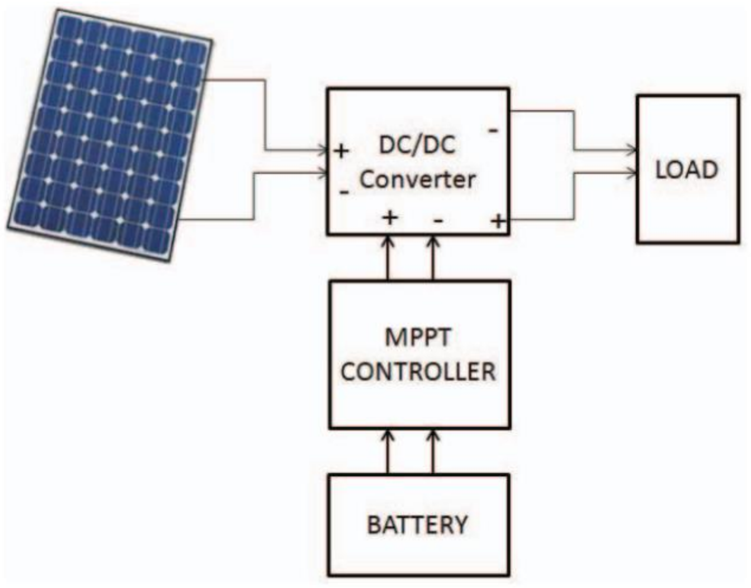


Figure 1: Block diagram of PV system

a. *Base PV Model*

At the heart of the MPPT system is the Base PV Model, a detailed representation of a solar panel that exhibits electrical characteristics akin to real-world PV cells. This simulation model is particularly sensitive to two key environmental inputs: irradiance and temperature. The Simulink model, as depicted in figure 2, is constructed with a granular level of detail, incorporating the nuanced physical processes of solar energy conversion.

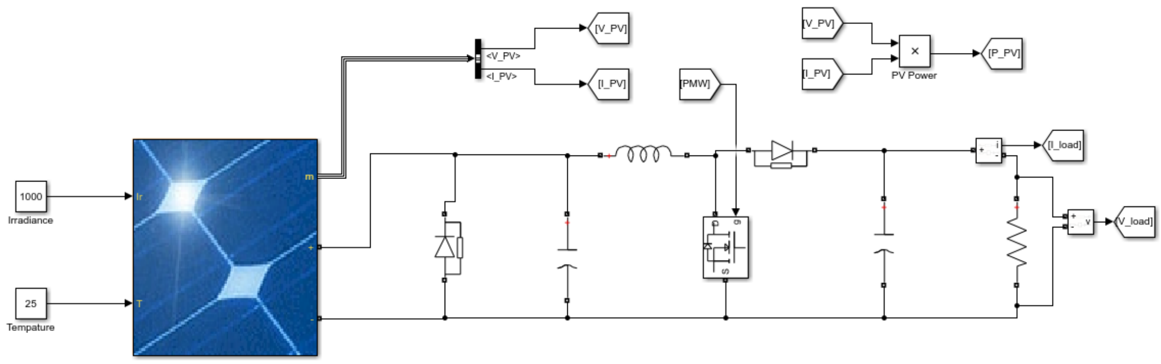


Figure 2: PV system implementation

This PV model is an assembly of several sub-components, including a source that simulates solar irradiance, a thermal model to account for temperature variations, and an equivalent circuit comprising diodes, resistors, and capacitors. These components collectively mimic the I-V (current-voltage) characteristics of a solar panel. By altering the irradiance and temperature inputs, one can observe the corresponding changes in the PV cell's output, thus enabling a dynamic analysis of the system's behavior.

Within Simulink, the model goes beyond mere simulation of solar cells—it extends to the encapsulation of an entire PV module or array, complete with the interconnection of multiple cells and bypass diodes. The entire setup is designed to emulate the power-generation process as accurately as possible, thereby laying a solid foundation for the implementation of MPPT algorithms.

b. <u>MPPT Algorithms integration</u>

The MPPT algorithms are implemented as distinct functional blocks within the Simulink model, each designed to interface seamlessly with the Base PV Model. The schematics depict the utilization of both the Particle Swarm Optimization (PSO) and Gravitational Search Algorithm (GSA) methods. These algorithms are represented in a way that abstracts the complex optimization processes into manageable blocks, focusing on the interaction between the PV system and the control logic as can be seen in the figure 3.

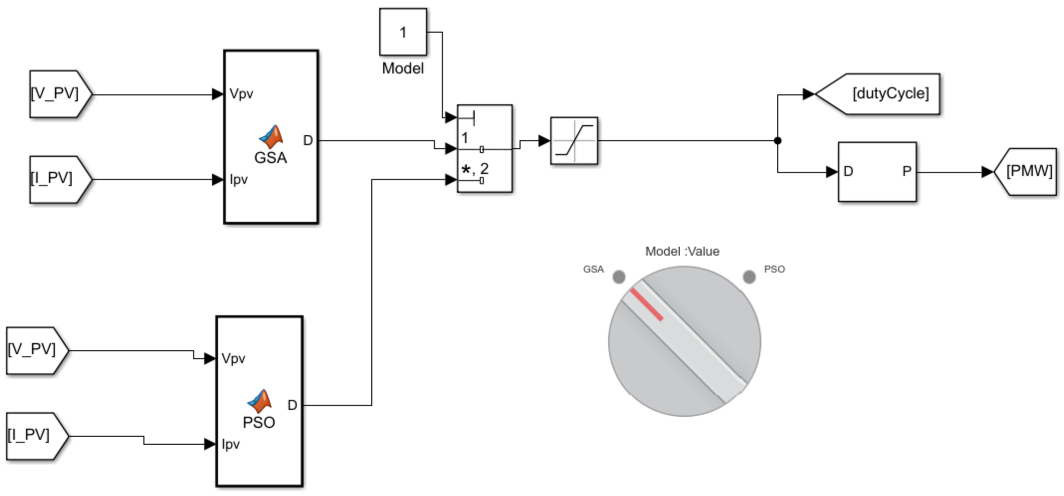


Figure 3: MPPT algorithms integration

The algorithms receive inputs directly from the PV model—such as the current and voltage—and apply their respective optimization techniques to iteratively adjust the duty cycle of a DC-DC converter. This process is inherently aimed at finding the optimal operating point that maximizes the power output from the PV module, irrespective of the static nature of the system's environmental inputs.The rotate switch allows to change which algorithms will be implemented inside the system.

c. *Visualization*

The visualization of simulation outcomes in a Simulink model of an MPPT system is pivotal to understanding the efficiency and effectiveness of the MPPT algorithms being used. The schematics provided suggest a systematic method of collecting and displaying data that highlights the dynamic performance of the system, particularly the power outputs from the PSO and GSA algorithms.

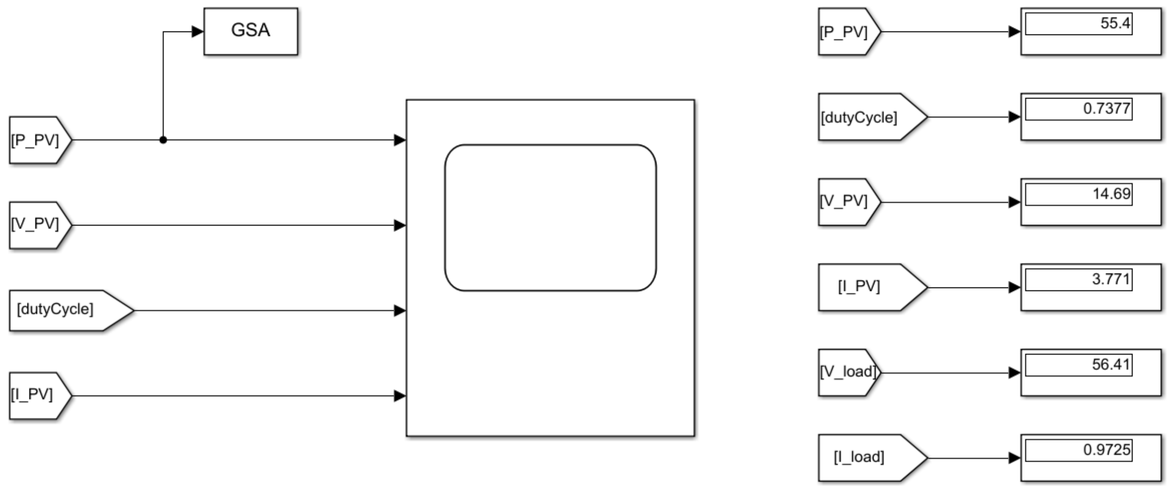


Figure 4: Block design for Visualization

In the Simulink environment (Figure 4), 'Display' blocks are utilized to provide immediate visual feedback of key performance indicators such as voltage, current, power, and the duty cycle. These blocks are connected directly to the outputs of the MPPT algorithms and the load, ensuring that any changes in the system's behavior are quickly reflected in the displayed values.

The use of a 'Workspace' block serves as a bridge between the Simulink model and the MATLAB workspace. This block is particularly useful for exporting simulation data for further analysis within the MATLAB environment. By feeding data from the MPPT algorithms into the Workspace block, it becomes possible to store the entire simulation dataset in a MATLAB variable.

3.2 PSO implementation

The implementation of the Particle Swarm Optimization (PSO) algorithm follows a well-defined process, as illustrated in the flowchart provided.

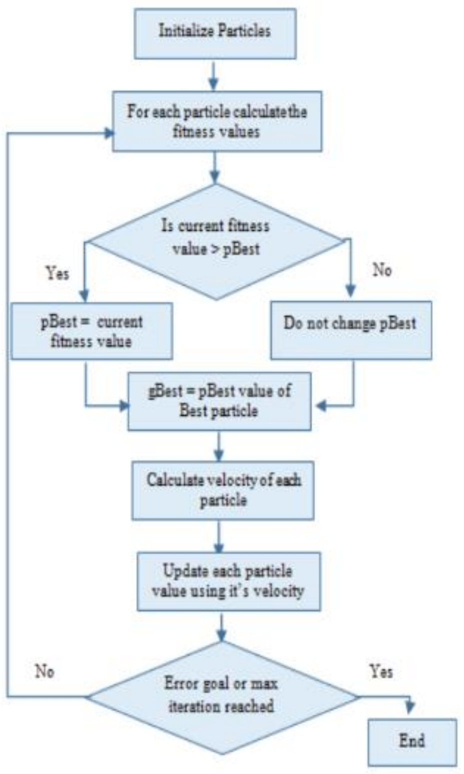

Initialize Particles
For each particle calculate the fitness values
Is current fitness value > pBest
Yes
No
pBest = current fitness value
Do not change pBest
gBest = pBest value of Best particle
Calculate velocity of each particle
Update each particle value using it's velocity
No
Error goal or max iteration reached
Yes
End

Figure 5: Flow of PSO algorithms

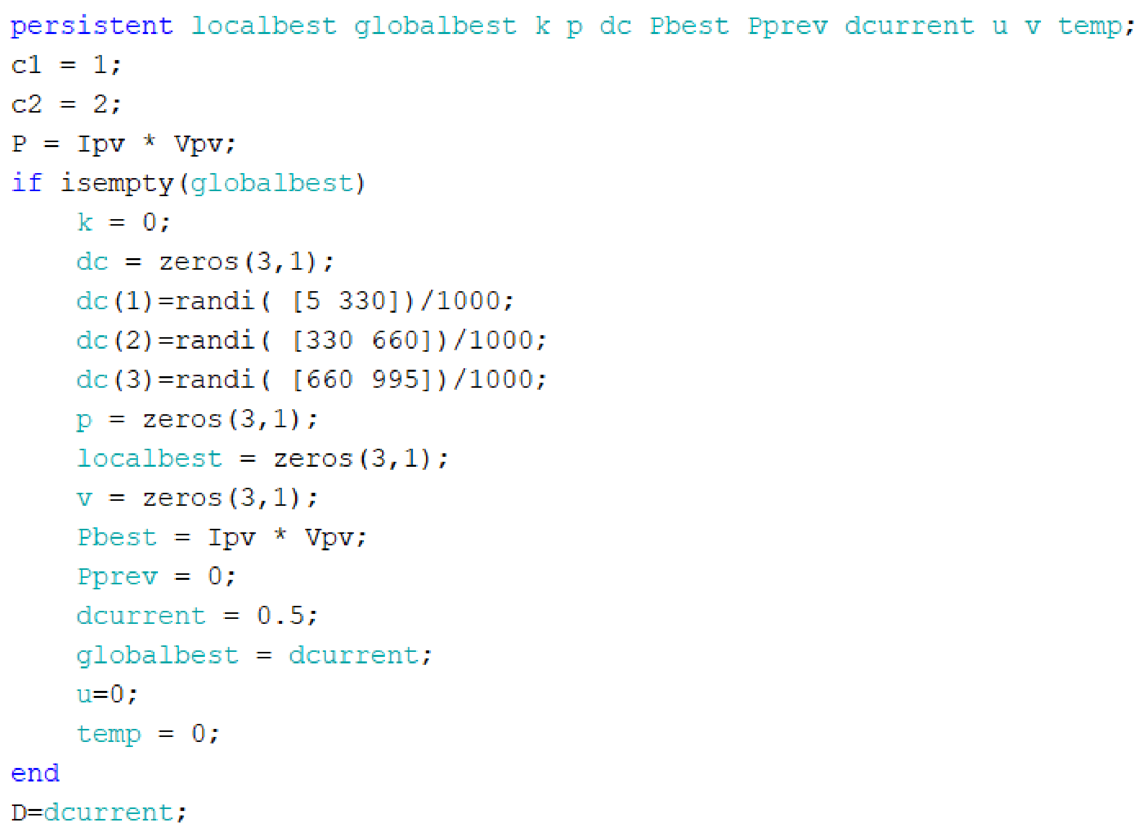
a. *Initialization*

When the PSO function is called, it initializes a set of persistent variables if they are not already set (checked via isempty(globalbest)). These variables store the state of the PSO algorithm between function calls:

- localbest: Stores the best position found by each particle.
- globalbest: Stores the best position found by any particle.
- k: A counter to limit the number of iterations the algorithm runs.
- p: Array to store the best power output found by each particle.
- dc: Array to store the duty cycle for each particle.
- Pbest: The best power output found by the algorithm so far.
- Pprev: The power output from the previous iteration.
- dcurrent: The current duty cycle being used.
- u: An index to cycle through each particle.

- v: Array to store the velocity of each particle.
- temp: A variable used as a conditional flag in the algorithm.

The duty cycles (dc) are initialized with random values within specific ranges to ensure a diverse starting point for each particle's exploration. The variable dcurrent is set to an initial guess of 0.5, and globalbest is assigned this value as the initial best-known position.

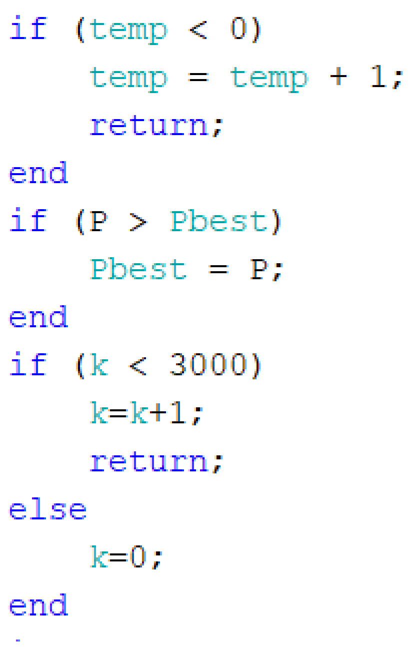


```
persistent localbest globalbest k p dc Pbest Pprev dcurrent u v temp;
c1 = 1;
c2 = 2;
P = Ipv * Vpv;
if isempty(globalbest)
    k = 0;
    dc = zeros(3,1);
    dc(1)=randi( [5 330])/1000;
    dc(2)=randi( [330 660])/1000;
    dc(3)=randi( [660 995])/1000;
    p = zeros(3,1);
    localbest = zeros(3,1);
    v = zeros(3,1);
    Pbest = Ipv * Vpv;
    Pprev = 0;
    dcurrent = 0.5;
    globalbest = dcurrent;
    u=0;
    temp = 0;
end
D=dcurrent;
```

Figure 6: Initialization of PSO particles

b. Iterative Loop

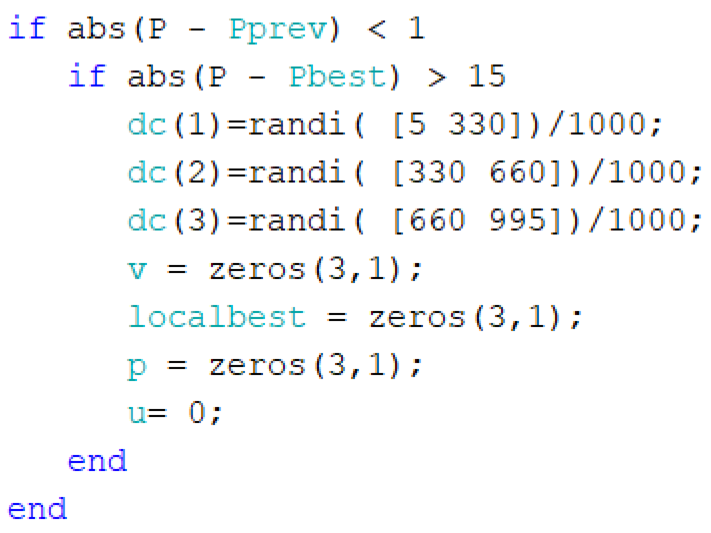
The main loop of the function starts by setting D to the current duty cycle (dcurrent). It then performs a series of checks and updates. If temp is negative, it increments temp and exits early. This mechanism could serve as a delay or a precondition check before the algorithm proceeds. If the current power P is greater than Pbest, it updates Pbest. This step is equivalent to evaluating a particle's fitness and updating its personal best. If the iteration counter k is less than 3000, it increments k and exits early, indicating that the algorithm should continue for more iterations.



```
if (temp < 0)
    temp = temp + 1;
    return;
end
if (P > Pbest)
    Pbest = P;
end
if (k < 3000)
    k=k+1;
    return;
else
    k=0;
end
```

Figure 7: Iteration loop implement (PSO)

If the change in power from the previous iteration is negligible, it suggests the possibility of being stuck at a local optimum. Therefore, the algorithm re-randomizes the duty cycles (dc) to potentially escape and explore new areas of the search space

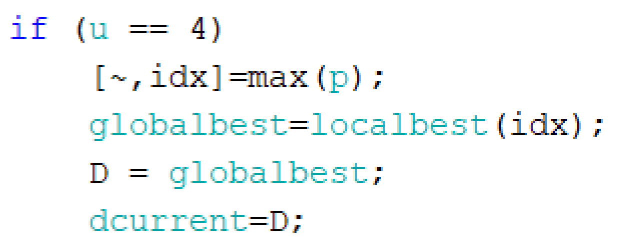


```
if abs(P - Pprev) < 1
   if abs(P - Pbest) > 15
      dc(1)=randi( [5 330])/1000;
      dc(2)=randi( [330 660])/1000;
      dc(3)=randi( [660 995])/1000;
      v = zeros(3,1);
      localbest = zeros(3,1);
      p = zeros(3,1);
      u= 0;
   end
end
```

Figure 8: Local optimum avoidance (PSO)

c. *Particle Updates*

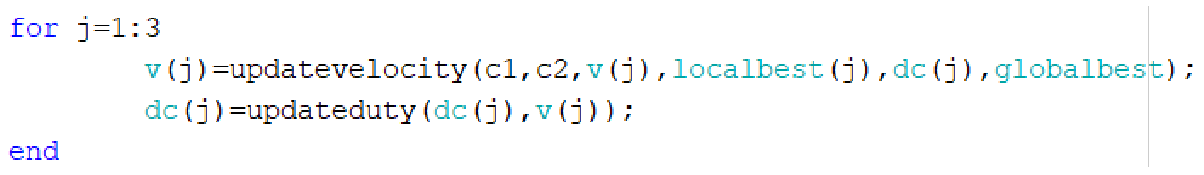
Next, the algorithm iterates through each particle (indexed by u) to update its personal best (p) and its best-known position (localbest). If u equals 4, it signifies the end of a complete cycle through the particles, prompting an update of globalbest to the best position found in this cycle.



```
if (u == 4)
    [~,idx]=max(p);
    globalbest=localbest(idx);
    D = globalbest;
    dcurrent=D;
```

Figure 9: Locating global best (PSO)

For each particle, the function calculates the new velocity using the updatevelocity function, which is influenced by the particle's current velocity, the distance from its personal best, and the distance from the global best. The new velocity is then used to update the particle's position (duty cycle) using the updateduty function. This position update is constrained to ensure the duty cycle remains between 0 and 1.

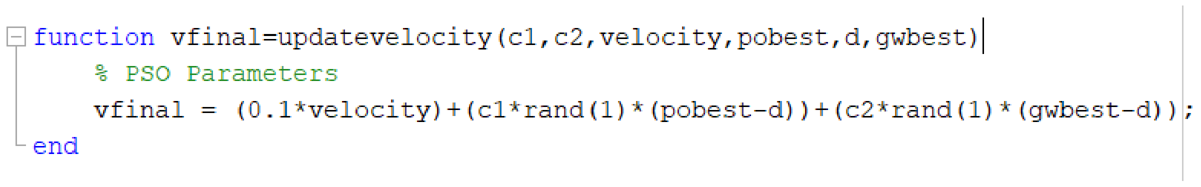


```
for j=1:3
        v(j)=updatevelocity(c1,c2,v(j),localbest(j),dc(j),globalbest);
        dc(j)=updateduty(dc(j),v(j));
end
```

Figure 10: Particle updates (PSO)

d. *Velocity Update*

This function computes the new velocity for a particle. It incorporates inertia (via a fraction of the current velocity), cognitive component (attraction to the particle's personal best), and social component (attraction to the global best). Random coefficients c1 and c2 introduce stochastic elements to the cognitive and social components, allowing the particles to explore the search space effectively.

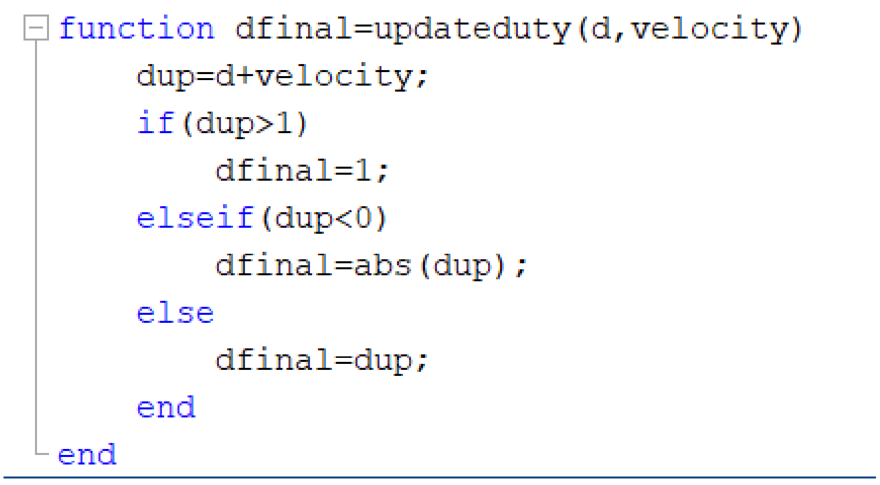


```
function vfinal=updatevelocity(c1,c2,velocity,pobest,d,gwbest)
    % PSO Parameters
    vfinal = (0.1*velocity)+(c1*rand(1)*(pobest-d))+(c2*rand(1)*(gwbest-d));
end
```

Figure 11: velocity updates (PSO)

e. Duty Update

This function updates the duty cycle of a particle based on its velocity. It ensures that the new duty cycle is within the valid range (0 to 1). If the updated value (dup) goes beyond this range, it is corrected to the closest bound (set to 1 if it's above 1, or taken as the absolute value if it's below 0).



```
function dfinal=updateduty(d,velocity)
    dup=d+velocity;
    if(dup>1)
        dfinal=1;
    elseif(dup<0)
        dfinal=abs(dup);
    else
        dfinal=dup;
    end
end
```

Figure 12: Duty updates (PSO)

By the end of the PSO function, D contains the duty cycle corresponding to the best solution found by the swarm (when u equals 4) or the next value to try (when u is less than 4). This value will be used by the MPPT system to adjust the converter's operation, aiming to reach the maximum power point of the PV system. The loop continues with the next call to the PSO function, allowing the algorithm to refine its search for the optimal duty cycle over successive iterations

3.3 GSA implementation

The MATLAB system provided the environments for implementing the Gravitational Search Algorithm (GSA) for optimizing the duty cycle in a Maximum Power Point Tracking (MPPT) system. The GSA is inspired by the laws of physics and utilizes the metaphor of gravitational attraction and mass interactions to navigate the search space towards the optimal solution, which in this case is the maximum power point of a PV system. The function in MATLAB closely follows the flow of the GSA algorithm as described in the following figure 7.

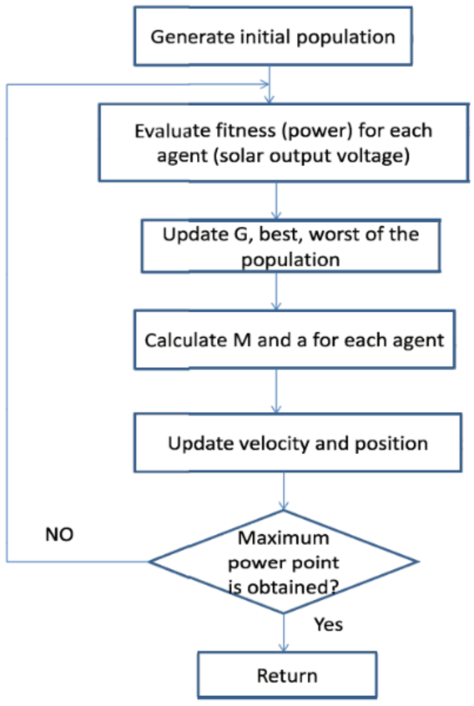
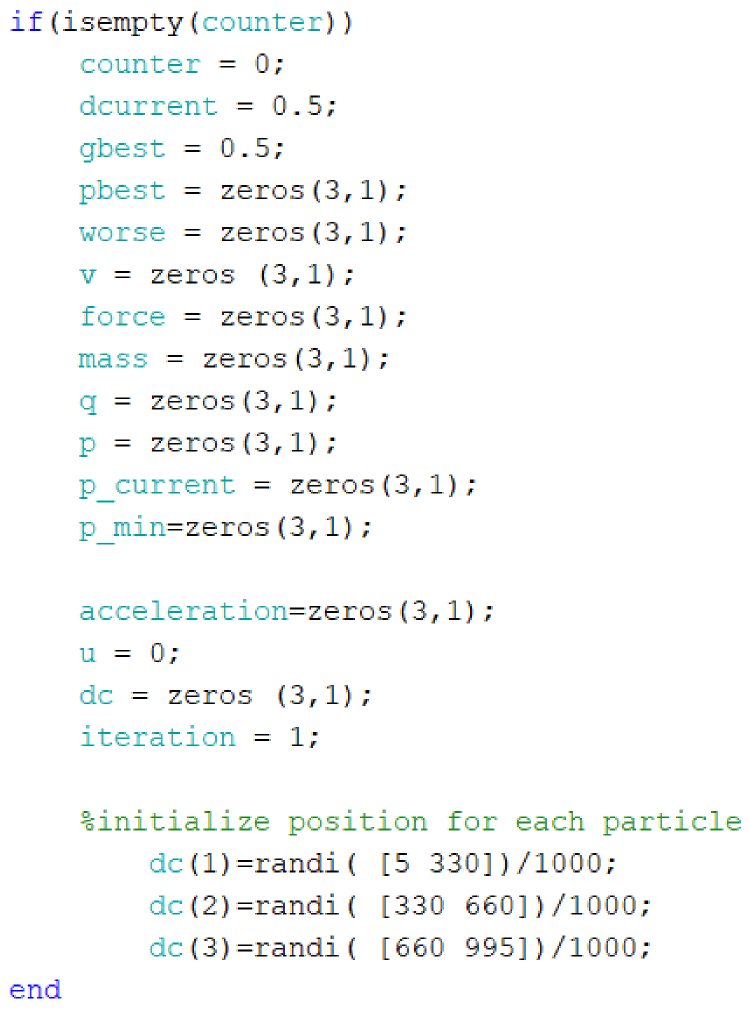

Generate initial population
Evaluate fitness (power) for each agent (solar output voltage)
Update G, best, worst of the population
Calculate M and a for each agent
Update velocity and position
NO
Maximum power point is obtained?
Yes
Return

Figure 13: Flow of GSA algorithms

a. *Initialization*

The commencement of the GSA function is marked by the inspection of the counter variable. The absence of its instantiation, determined by an empty state, signals the inaugural execution of the algorithm. This phase is characterized by the initialization of an array, dc, which encapsulates the positional data of each agent within the search space. These initial positions, derived through a randomized allocation, span a vast spectrum of potential solutions, thereby facilitating comprehensive exploration.

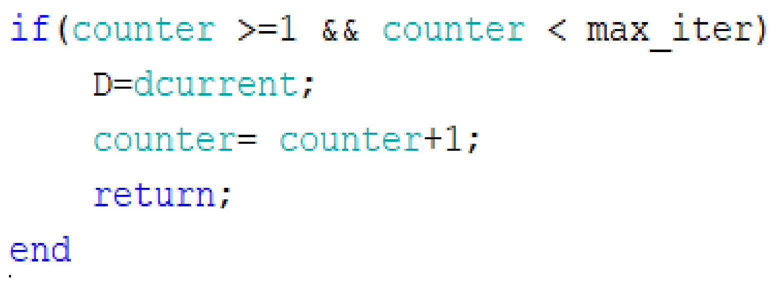


```
if(isempty(counter))
    counter = 0;
    dcurrent = 0.5;
    gbest = 0.5;
    pbest = zeros(3,1);
    worse = zeros(3,1);
    v = zeros (3,1);
    force = zeros(3,1);
    mass = zeros(3,1);
    q = zeros(3,1);
    p = zeros(3,1);
    p_current = zeros(3,1);
    p_min=zeros(3,1);

    acceleration=zeros(3,1);
    u = 0;
    dc = zeros (3,1);
    iteration = 1;

    %initialize position for each particle
        dc(1)=randi( [5 330])/1000;
        dc(2)=randi( [330 660])/1000;
        dc(3)=randi( [660 995])/1000;
end
```

Figure 14: Initialization of GSA particles

b. Iterative Loop

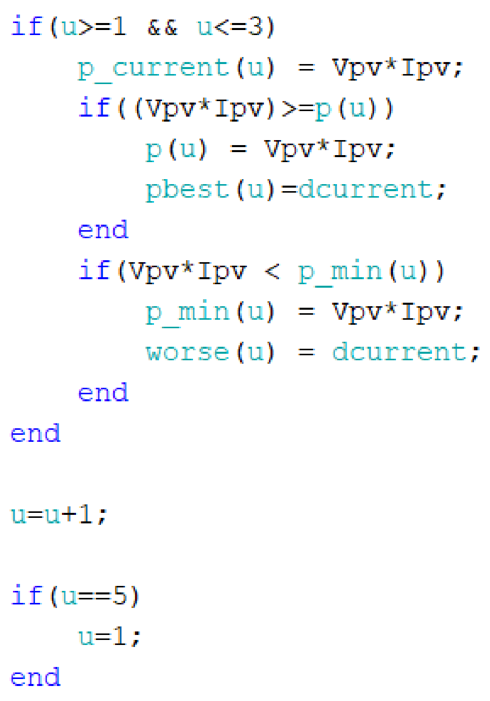
Post-initialization, the agents embark on their iterative journey through the search space, propelled by the GSA's algorithmic impetus. Each iteration represents an incremental time step, advancing the agents' search for the optimal solution. A predefined maximum number of iterations, max_iter, serves as the sentinel, ensuring the algorithm's termination upon fulfillment of its exploratory mandate.



```
if(counter >=1 && counter < max_iter)
    D=dcurrent;
    counter= counter+1;
    return;
end
```

Figure 15: GSA iteration loop

During each iteration, indexed by the variable u, the algorithm engages in a meticulous evaluation of fitness, denoted by the power output of each agent. This process entails the comparison of the current output with the previously established personal best (pbest) and worst (worse) records. The discovery of a new extremum instigates an update to these records, reflecting a continual refinement of the agents' performance metrics

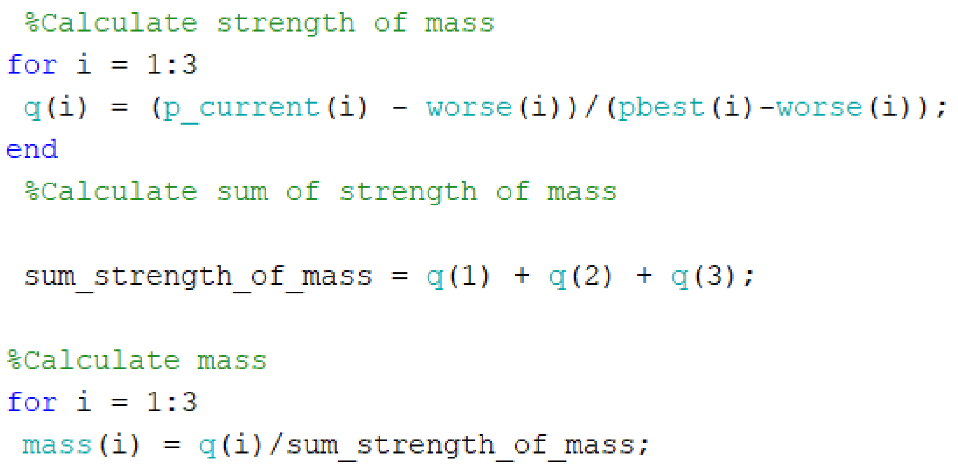


```
if(u>=1 && u<=3)
    p_current(u) = Vpv*Ipv;
    if((Vpv*Ipv)>=p(u))
        p(u) = Vpv*Ipv;
        pbest(u)=dcurrent;
    end
    if(Vpv*Ipv < p_min(u))
        p_min(u) = Vpv*Ipv;
        worse(u) = dcurrent;
    end
end

u=u+1;

if(u==5)
    u=1;
end
```

Figure 16: Update best and worst fitness (GSA)

c. *Mass Calculation*

Subsequent to the assessment of fitness, the GSA algorithm proceeds to compute the relative mass (mass) of each agent. This computation incorporates the variable q, emblematic of the agents' mass strength, formulated as a ratio of their current fitness against the collective extremities of the population. Such a translation of fitness into gravitational mass is pivotal to the GSA framework, conferring a physical dynamism to the agents' interactions.

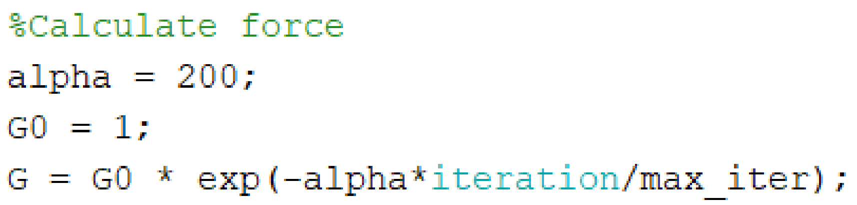


```
 %Calculate strength of mass
for i = 1:3
 q(i) = (p_current(i) - worse(i))/(pbest(i)-worse(i));
end
 %Calculate sum of strength of mass

 sum_strength_of_mass = q(1) + q(2) + q(3);

%Calculate mass
for i = 1:3
 mass(i) = q(i)/sum_strength_of_mass;
```

Figure 17: Mass function computation (GSA)

d. *Force Calculation*

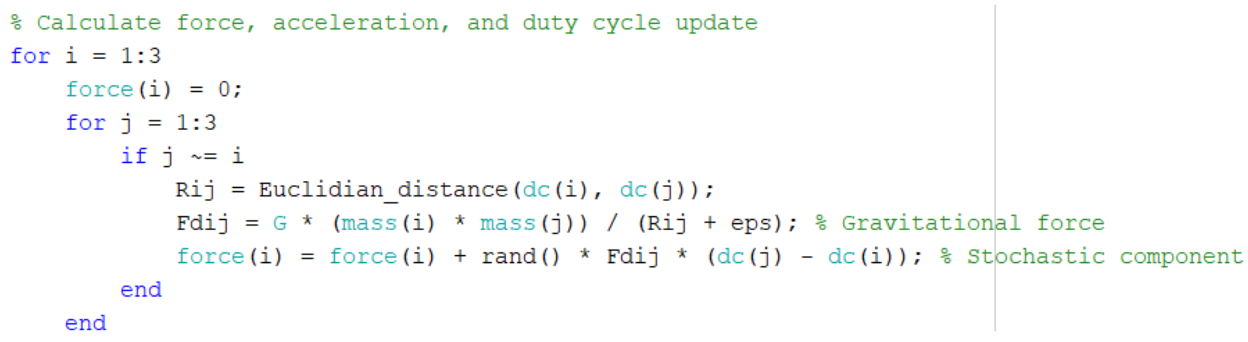
With the agents' masses thus ascertained, the algorithm advances to the force computation stage. Here, the gravitational constant G is subjected to an exponential decay over time, embodying the gradual convergence of the algorithm.



```
%Calculate force
alpha = 200;
G0 = 1;
G = G0 * exp(-alpha*iteration/max_iter);
```

Figure 18: G calculation (GSA)

The forces acting between pairs of agents are calculated, incorporating a random element to imbue the system with stochastic behavior. This randomness is fundamental to the algorithm's capacity to navigate complex search spaces, avoiding premature convergence to local optima.

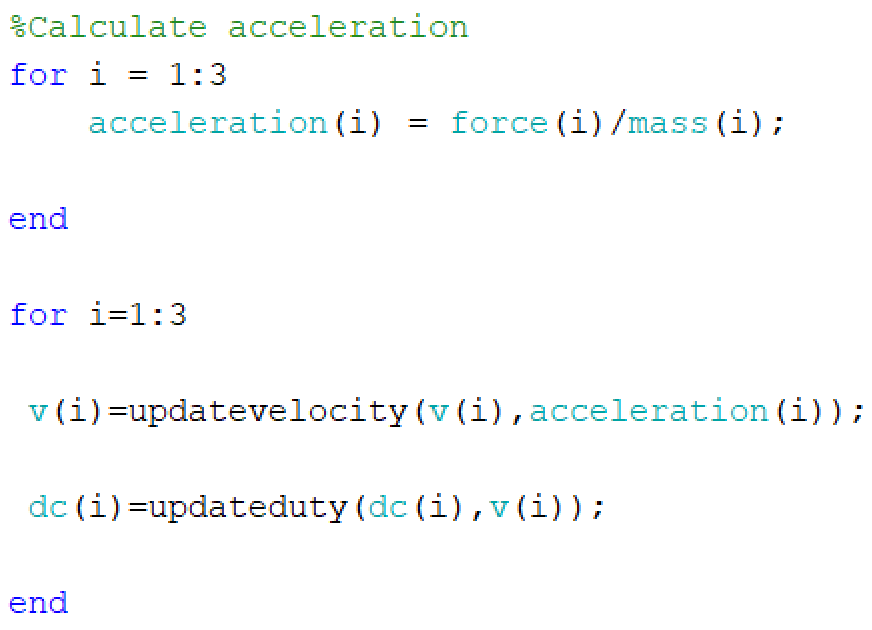


```
% Calculate force, acceleration, and duty cycle update
for i = 1:3
    force(i) = 0;
    for j = 1:3
        if j ~= i
            Rij = Euclidian_distance(dc(i), dc(j));
            Fdij = G * (mass(i) * mass(j)) / (Rij + eps); % Gravitational force
            force(i) = force(i) + rand() * Fdij * (dc(j) - dc(i)); % Stochastic component
        end
    end
```

Figure 19: Force computation (GSA)

e. Position Update

The calculated forces precipitate an update in the agents' velocities and subsequently, their positions—conceptualized as duty cycles in the MPPT context. The update mechanism is infused with randomness, maintaining a portion of the previous velocity while assimilating the new acceleration induced by gravitational interactions. The resultant movement of the agents is a confluence of their inertia and the collective gravitational pull, steering them towards regions of the search space characterized by superior power outputs.

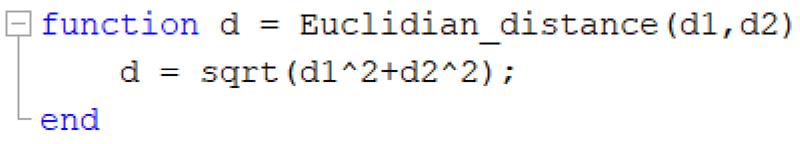


```
%Calculate acceleration
for i = 1:3
    acceleration(i) = force(i)/mass(i);

end

for i=1:3

 v(i)=updatevelocity(v(i),acceleration(i));

 dc(i)=updateduty(dc(i),v(i));

end
```

Figure 20: particle position update (GSA)

f. Distance Function

The Euclidian_distance function calculates the Euclidean distance between two points, d1 and d2, which represent the positions (duty cycles) of two agents in the search space. The Euclidean distance is a measure of the straight-line distance between two points in a multi-dimensional space. In the context of GSA, this distance plays a role in the force calculation between agents, as the gravitational force is inversely proportional to the distance between the masses (agents).

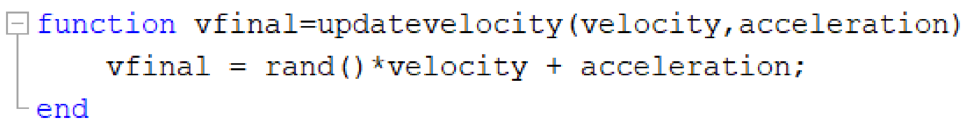


```
function d = Euclidian_distance(d1,d2)
    d = sqrt(d1^2+d2^2);
end
```

Figure 21: Distance function (GSA)

g. Velocity Update

The updatevelocity function updates the velocity of an agent based on its current velocity and the acceleration due to the gravitational force. The function introduces a stochastic element by multiplying the current velocity with a random number (rand()), which adds variability to the agent's movement. The acceleration, a result of the gravitational force divided by the agent's mass, is then added to this product to compute the new velocity (vfinal).

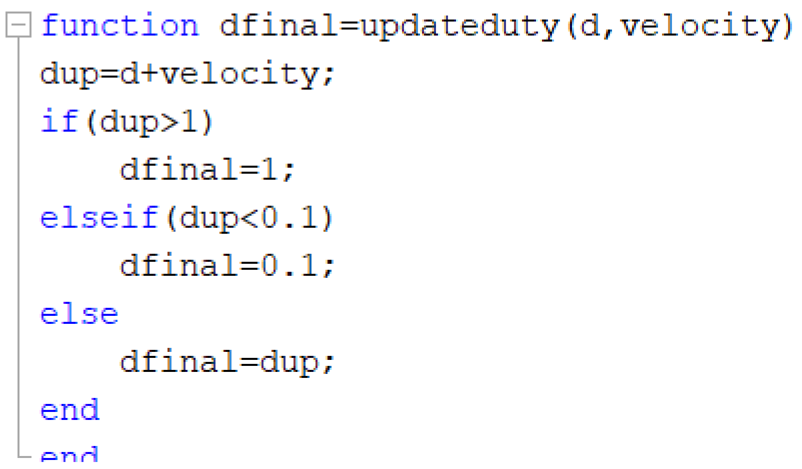


```
function vfinal=updatevelocity(velocity,acceleration)
    vfinal = rand()*velocity + acceleration;
end
```

Figure 22: Velocity function (GSA)

h. Duty Update

The updateduty function calculates the new position of an agent by adding the velocity to the current position (duty cycle d). The resulting duty cycle dup is then checked to ensure that it is within the valid range for the system. If dup exceeds 1, it is set to 1, and if it is less than 0.1, it is set to 0.1. These bounds ensure that the duty cycle remains within a physically realizable range for the PV system.



```
function dfinal=updateduty(d,velocity)
dup=d+velocity;
if(dup>1)
    dfinal=1;
elseif(dup<0.1)
    dfinal=0.1;
else
    dfinal=dup;
end
end
```

Figure 23: Update Duty function (GSA)

By harnessing the principles of gravitational interaction and mass relationships, the GSA provides an effective and powerful means of tracking the maximum power point in PV systems. It reflects a fine-tuned process that adapts to the unique characteristics of the search space, methodically progressing towards the peak power output, which is the ultimate goal of the MPPT system.

4. Analysis

In a comprehensive academic exploration, we delve into the algorithmic efficacy of two predominant strategies in the domain of photovoltaic system optimization: Gravitational Search Algorithm (GSA) and Particle Swarm Optimization (PSO). The investigation employs an experimental setup where both methodologies deploy three agents over a span of 3000 iterations in the span of time of 0.3 seconds.

4.1 Finding

The data are visually represented across several key performance indicators: power, duty cycle, voltage, and current

The power graphs indicate that GSA quickly stabilizes to an optimal power level, exhibiting minor fluctuations, suggestive of a robust and effective tracking approach. In contrast, PSO's power graph is marked by a more varied performance, with power levels taking a longer time to stabilize, reflecting its expansive exploration of the search space.

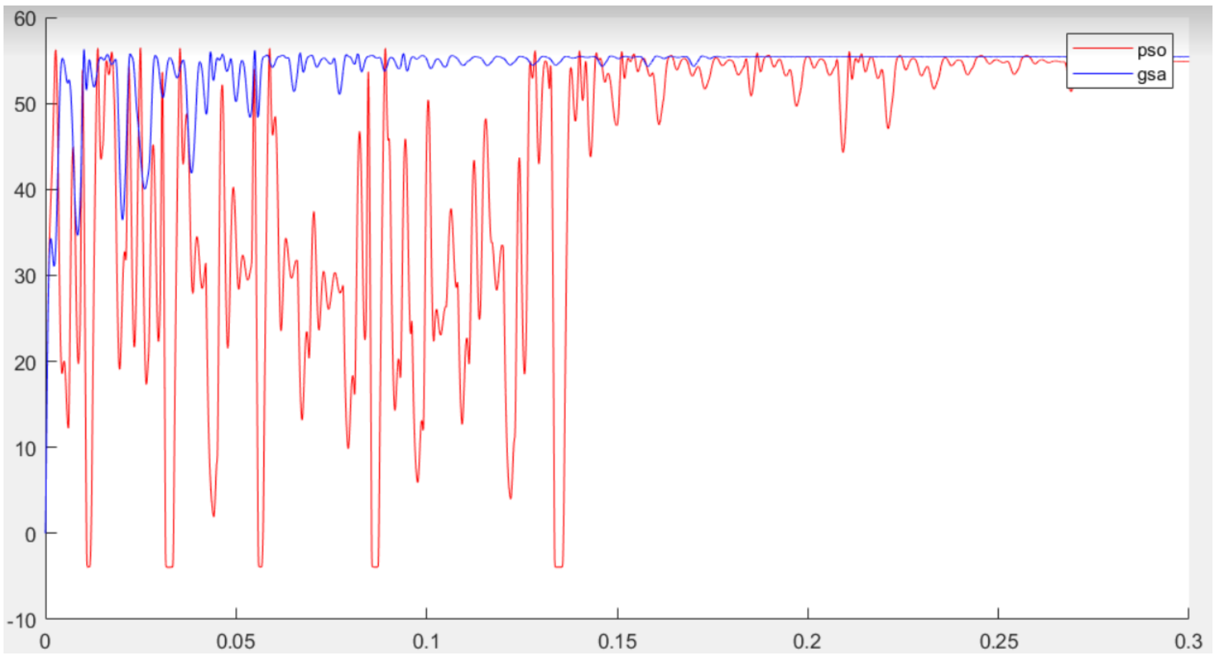


Figure 24: Maximal Power convergent pattern between GSA and PSO

Examining the duty cycle adjustments, GSA demonstrates a swift approach to identifying an optimal duty cycle, as evidenced by fewer and smaller adjustments. On the other hand, PSO shows larger and more frequent changes in duty cycle, implying a more extensive search behavior.

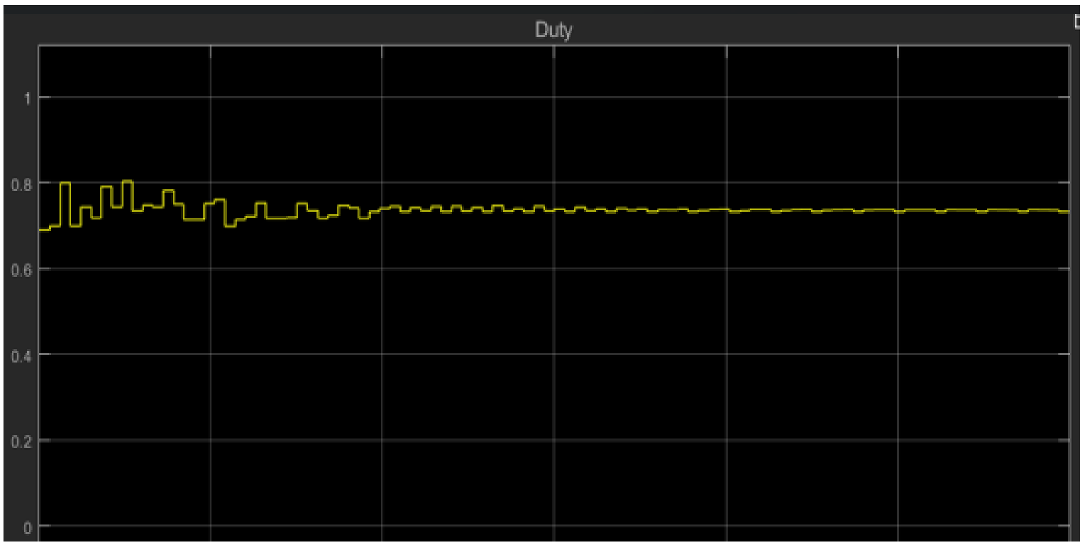


Figure 25: Duty pattern of GSA

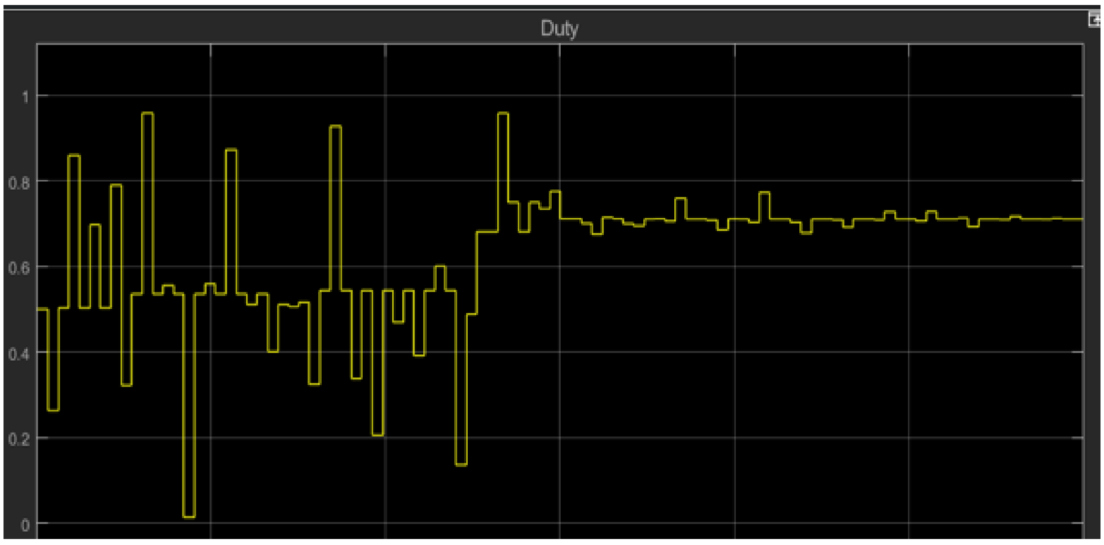


Figure 26: Duty pattern of PSO

The findings of this experimental analysis resonate with the results presented in the paper titled "A GSA Based Improved MPPT System for PV Generation." This affirms the superior performance of GSA in terms of faster convergence and less oscillatory behavior.

	GSA	**PSO**
Max Power (Watt)	55.4	54.32
Stabilize Time (Sec)	0.173	0.278
Oscillation	less	more
Convergence	faster	slower

Table 1: Performance of GSA and PSO

4.2 Discussion

The comparative finding of Gravitational Search Algorithm (GSA) and Particle Swarm Optimization (PSO) within the realm of Maximum Power Point Tracking (MPPT) unveils distinct operational profiles for each algorithm.

GSA's proficiency is evident in the data, which highlights its rapid convergence to the maximum power point, suggesting an innate capability to efficiently home in on the optimal solution. The consistent maintenance of a steady state following convergence further exemplifies GSA's stability, marking it as an effective tool for MPPT where rapid and reliable tracking is indispensable. Such performance is advantageous in dynamic environments where swift adaptation to changing solar irradiance and temperature conditions is essential. In contrast, PSO's performance is defined by its exploratory nature. The algorithm's oscillatory convergence pattern, as reflected in the experimental data.

However, GSA is not without its disadvantages. The algorithm becomes computationally demanding as the number of agents increases due to the necessity to calculate gravitational forces between all pairs of particles, potentially limiting its scalability. There's also a risk of premature convergence, which could lead to suboptimal solutions if the gravitational constant is not adequately managed. Moreover, GSA's performance may be significantly affected by the initial distribution of agents, highlighting its sensitivity to initial conditions.

On the other hand, PSO is praised for its simplicity and ease of implementation, characterized by fewer parameters that need adjusting. This simplicity makes PSO accessible for a variety of applications, supported by its adaptability to different problem domains, including discrete and combinatorial optimization tasks. Nevertheless, PSO faces challenges, such as the risk of getting stuck in local optima in complex landscapes, which necessitates maintaining swarm diversity. The algorithm's performance is also highly sensitive to parameter settings, including inertia weight and acceleration coefficients, which can significantly impact outcomes.

	GSA	PSO
Global Search Capability	Strong, due to gravitational interaction mechanism	Moderate, depends on swarm diversity maintenance
Parameter Complexity	High, with more adjustable parameters	Low, simple parameters but sensitive to settings
Versatility	Moderate, suitable for nonlinear, multimodal, problems	Higher, adaptable to a range of problem domains
Computational Efficiency	Lower, due to intensive force calculations among agents	Higher, more straightforward particle updates

Risk of Premature Convergence	Present, especially if gravitational constant is mismanaged	Present, particularly in complex landscapes
Sensitivity to Initial Conditions	High, initial agent distribution impacts performance	Moderate, initial swarm setup can affect outcomes
Scalability	Challenged by large agent numbers	Potentially limited by problem dimensionality

Table 2: GSA and PSO characteristics

5. Conclusion

In conclusion, the application of Particle Swarm Optimization (PSO) and Gravitational Search Algorithm (GSA) to Maximum Power Point Tracking (MPPT) systems has provided a profound and insightful comparison between these two optimization algorithms. Each algorithm demonstrates unique strengths and limitations when applied to the task of optimizing MPPT systems, highlighting the importance of choosing the appropriate algorithm based on the specific requirements of the application. The implementation of GSA for MPPT systems, in particular, has shown significant potential for future applications. Its robust global search capability, derived from the principles of gravity and mass interactions, makes it exceptionally effective for thoroughly exploring the search space associated with MPPT optimization problems.

It is hoped that this study will serve as a valuable resource for future research in the field of MPPT optimization. As the demand for efficient renewable energy systems continues to grow, future research can build upon these findings, exploring hybrid approaches, refining algorithm parameters, and investigating the scalability and adaptability of these algorithms to different MPPT system configurations, thereby enhancing the efficiency and effectiveness of renewable energy harvesting.

IV. Appendix

a) PSO code

```
function D = PSO(Vpv, Ipv)
  coder.extrinsic('randi')
  persistent localbest globalbest k p dc Pbest Pprev dcurrent u v temp;
  c1 = 1;
  c2 = 2;
  P = Ipv * Vpv;
  if isempty(globalbest)
      k = 0;
      dc = zeros(3,1);
      dc(1)=randi( [5 330])/1000;
      dc(2)=randi( [330 660])/1000;
      dc(3)=randi( [660 995])/1000;
      p = zeros(3,1);
      localbest = zeros(3,1);
      v = zeros(3,1);
      Pbest = Ipv * Vpv;
      Pprev = 0;
      dcurrent = 0.5;
      globalbest = dcurrent;
      u=0;
      temp = 0;
  end
  D=dcurrent;

  if (temp < 0)
      temp = temp + 1;
      return;
  end
  if (P > Pbest)
      Pbest = P;
  end
  if (k < 3000)
      k=k+1;
      return;
  else
      k=0;
  end
  if abs(P - Pprev) < 1
     if abs(P - Pbest) > 15
        dc(1)=randi( [5 330])/1000;
        dc(2)=randi( [330 660])/1000;
        dc(3)=randi( [660 995])/1000;
        v = zeros(3,1);
        localbest = zeros(3,1);
        p = zeros(3,1);
        u= 0;
     end
  end

  if(u>=1 && u<=3)
      if(P>p(u))
          p(u)=P;
          localbest(u)=dcurrent;
      end
  end

  u=u+1;

  if (u > 4)
      u=1;
  end
  if (u == 4)
      [~,idx]=max(p);
      globalbest=localbest(idx);
      D = globalbest;
      dcurrent=D;
      for j=1:3
              v(j)=updatevelocity(c1,c2,v(j),localbest(j),dc(j),globalbest);
              dc(j)=updateduty(dc(j),v(j));
      end
  else
      D=dc(u);
      dcurrent=dc(u);
      Pprev = P;
  end


end

function vfinal=updatevelocity(c1,c2,velocity,pobest,d,gwbest)
  % PSO Parameters
  vfinal = (0.1*velocity)+(c1*rand(1)*(pobest-d))+(c2*rand(1)*(gwbest-d));
end

function dfinal=updateduty(d,velocity)
  dup=d+velocity;
  if(dup>1)
      dfinal=1;
  elseif(dup<0)
      dfinal=abs(dup);
  else
      dfinal=dup;
  end
end
```

b) GSA code

```
function D = GSA(Vpv,Ipv)
  coder.extrinsic('randi')
  persistent u;
  persistent dcurrent;%store current duty cycle
  persistent pbest;%store local best dc for power
  persistent force; %store force
  persistent acceleration; %store acceleration
  persistent mass; % mass
  persistent q; %  strength of mass
  persistent p; %  power for each particle
  persistent p_current; %  power current for each particle
  persistent p_min; %  power min for each particle
  persistent worse;   %store best worse of each particle
  persistent dc; %store duty cycle ~ position
  persistent v;  %velocity
  persistent counter; %delay iteration
  persistent iteration;
  persistent gbest;%store global best dc for power
  %initialization

  max_iter = 3000;
  if(isempty(counter))
      counter = 0;
      dcurrent = 0.5;
      gbest = 0.5;
      pbest = zeros(3,1);
      worse = zeros(3,1);
      v = zeros (3,1);
      force = zeros(3,1);
      mass = zeros(3,1);
      q = zeros(3,1);
      p = zeros(3,1);
      p_current = zeros(3,1);
      p_min=zeros(3,1);

      acceleration=zeros(3,1);
      u = 0;
      dc = zeros (3,1);
      iteration = 1;

      %initialize position for each particle
          dc(1)=0.69;
          dc(2)= 0.7;
          dc(3)=0.8;

  end
```

```
if(counter >=1 && counter < 3000)
    D=dcurrent;
    counter= counter+1;
    return;
end

if(u>=1 && u<=3)
    p_current(u) = Vpv*Ipv;
    if((Vpv*Ipv)>=p(u))
        p(u) = Vpv*Ipv;
        pbest(u)=dcurrent;
    end
    if(Vpv*Ipv < p_min(u))
        p_min(u) = Vpv*Ipv;
        worse(u) = dcurrent;
    end
end
u=u+1;
if(u==5)
    u=1;
end
if(u >= 1 && u <= 3)
    %Avoid over shooting
    if(iteration < max_iter)
        D=dc(u);
        dcurrent=D;
        counter=1;
    return;
    else
        D = dcurrent;
        return
    end
elseif(u==4)
    iteration = iteration +1;
    [~,i]=max(p);
    gbest=pbest(i);
    D=gbest;
    dcurrent=D;
    counter=1;
     %Calculate strength of mass
    for i = 1:3
     q(i) = (p_current(i) - worse(i))/(pbest(i)-worse(i));
    end
     %Calculate sum of strength of mass

     sum_strength_of_mass = q(1) + q(2) + q(3);

    %Calculate mass
    for i = 1:3
     mass(i) = q(i)/sum_strength_of_mass;
```

```
      end
      %Calculate force
      alpha = 200;
      G0 = 1;
      G = G0 * exp(-alpha*iteration/max_iter);
      %G = 6.67430 * 10^-13; %gravitational constant
      e = 2.2204*10^-16;
      for i = 1:3
          force(i) = 0;
          for j = 1:3
              if j ~= i
                  Rij = Euclidian_distance(dc(i), dc(j));
                  Fdij = G * (mass(i) * mass(j)) / (Rij + e); % Gravitational
force
                  force(i) = force(i) + rand() * Fdij * (dc(j) - dc(i)); %
Stochastic component
              end
          end
       end
      %Avoid over shooting
      if(iteration == max_iter)

          D=dcurrent;
          return;
      end
      %Calculate acceleration
      for i = 1:3
          acceleration(i) = force(i)/mass(i);

      end

      for i=1:3

       v(i)=updatevelocity(v(i),acceleration(i));

       dc(i)=updateduty(dc(i),v(i));

      end
      return;

  else
      D=0.5;
  end
  end


  function d = Euclidian_distance(d1,d2)
      d = sqrt(d1^2+d2^2);
  end
```

```
function vfinal=updatevelocity(velocity,acceleration)
    vfinal = rand()*velocity + acceleration;
end

function dfinal=updateduty(d,velocity)
dup=d+velocity;
if(dup>1)
    dfinal=1;
elseif(dup<0.1)
    dfinal=0.1;
else
    dfinal=dup;
end
end
```

c) Matlab visualization code

```
data(:,1) = PSO.data;
data(:,2) = GSA.data;
data(:,3) = GSA.time;
time = data(:,3);
pso = data(:,1);
gsa = data(:,2);

figure;
hold on;

for i = 1:2
    p1 = plot(time, pso,'r');
    p2 = plot(time, gsa,'b');
end
hold off
legend([p1(1) p2(1)],'pso', 'gsa');
ah1 = axes('position',get(gca,'position'),'visible','off');
```

V. Reference

[1] A Comprehensive Survey on Gravitational Search Algorithm by Esmat Rashedi, Elaheh Rashedi, Hossein Nezamabadi-pour: https://www.sciencedirect.com/science/article/am/pii/S2210650217303577

[2] A Gentle Introduction to Particle Swarm Optimization by Adrian Tam on October 12, 2021:

https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/

[3] Maximizing photovoltaic system power output with a master-slave strategy for parallel inverters by Mohamed Zaki , Ahmed Shahin , Saad Eskander , Mohamed A. Elsayes , Vladimír Bureš : https://www.sciencedirect.com/science/article/pii/S2352484723016128

[4] A maximum power point tracking method for PV system with improved gravitational search algorithm by Ling-Ling Li , Guo-Qian Lin , Ming-Lang Tseng , Kimhua Tan , Ming K. Lim https://www.sciencedirect.com/science/article/abs/pii/S156849461830036X

[5] Artificial Neural Networks in MPPT Algorithms for Optimization of Photovoltaic Power Systems: A Review by César G. Villegas-Mier, Juvenal Rodriguez-Resendiz, José M. Álvarez-Alvarado, Hugo Rodriguez-Resendiz, Ana Marcela Herrera-Navarro, and Omar Rodríguez-Abreo: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8541603/