# Section 4: Implementing the Rhino Optimiser Mechanism

Rhinoceroses exhibit a distinctive charging behaviour in response to perceived threats or objectives. The mechanism leverages **targeted acceleration** towards promising solutions while incorporating **collision avoidance** to refine search trajectories.

  
Fig.1: Charging Rhino

The optimiser involves:

1. High-Velocity Acceleration: Rapid movement towards the target.
2. Trajectory Adjustment: Mid-course corrections based on sensory feedback.
3. Collision Avoidance: Redirecting if the target is misidentified or conditions change.

This behaviour can be formulated as an optimization mechanism where solutions move aggressively toward promising areas while adapting their trajectory dynamically.

## 4.1 Mathematical Formulation

The position update equation for an individual Rhino in ROA is defined as:

*X*i(t+1)  = *X*i(t)  + Vi(t) + (*X*best – Xi(t))

where:

* *X*i(t)  is the position of the i-th solution at iteration t.
* Vi(t) is the velocity of the solution.
* *X*best ​ represents the best-known solution.
* α\alpha is the **acceleration factor**, controlling optimiser intensity.
* β\beta is the **redirection factor**, ensuring collision avoidance.

Additionally, stochastic redirection is applied with probability *p* to prevent premature convergence:

*X*i(t+1)​=Xi(t+1)​+*γ*\**U*(−1,1)

where *γ*\gamma is a small perturbation factor, and *U* (−1,1) is a uniform random variable.

## 4.3 Comparison with Rhino Herd Optimization (RHO)

The Rhino Herd Optimization (RHO) method by Wang et al., (2017), simulates social behaviours of rhinoceros herds. The algorithm models the movement and interaction patterns of rhinos to explore and exploit the search space effectively. The performance of the proposed method is evaluated using standard benchmark functions, demonstrating its potential in solving complex optimization problems. This study provides a foundation for optimization algorithms based on rhinoceros behaviour, focusing on herd dynamics rather than the charging mechanism. As of now, specific research utilizing the rhino's charging behaviour for optimization purposes appears to be limited. Unlike ROA, which prioritizes exploitation, RHO maintains diversity through social influence and random perturbations. The table below is a tabular difference between the Rhino heard optimiser and the proposed Rhino Optimiser.

Table 1: Summary Table between Rhino Optimiser

|  |  |  |
| --- | --- | --- |
| **Feature** | **Rhino Optimizer (ROA)** | **Rhino Herd Optimization (RHO)** |
| Inspiration | Charging behaviour of rhinos | Herd social interactions |
| Exploration | Moderate | High |
| Exploitation | High | Moderate |
| Convergence Rate | Faster | Slower |
| Risk of Local Optima | Higher | Lower |

## 4.3 Experimental Setup

In the evaluation of bio-inspired metaheuristic optimization algorithms, the selection of benchmark functions is pivotal for assessing performance across diverse problem landscapes. The Sphere, Rastrigin, and Rosenbrock functions are frequently employed in this context due to their distinct characteristics, which collectively provide a comprehensive evaluation framework. Collectively, these benchmark functions provide a robust framework for evaluating optimization algorithms, ensuring a comprehensive assessment of their performance across various problem landscapes. By using these three benchmark functions, they capture different optimization challenges and evaluate how well ROA performs in simple, complex, and deceptive landscapes.

4.3.1. Sphere Function (Tests Exploitation in a Unimodal Landscape)

The Sphere function is a continuous, convex, and unimodal benchmark defined as:



This function is instrumental in assessing an algorithm's exploitation capabilities, as it presents a straightforward landscape with a single global minimum at the origin. Algorithms are expected to demonstrate rapid convergence towards this point, thereby testing their efficiency in local search scenarios. The utilization of the Sphere function in performance evaluation is well-documented in the literature, (Zhong et al., 2025).

4.3.2. Rastrigin Function (Tests Exploration in a Multimodal Landscape)

The Rastrigin function introduces a more complex, multimodal landscape, defined as:



Characterized by numerous local minima, this function evaluates an algorithm's exploration capabilities and its proficiency in avoiding premature convergence. The Rastrigin function is commonly employed to assess the balance between exploration and exploitation in optimization algorithms, (Givi et al., 2023).

**4.3.3. Rosenbrock Function** (Tests Precision in a Narrow Valley)

The Rosenbrock function, also known as the "Valley" or "Banana" function, is defined as:



This function features a narrow, curved valley leading to the global minimum, making it suitable for testing an algorithm's precision and its ability to navigate complex, non-convex landscapes. The Rosenbrock function is frequently utilized to assess the performance of optimization algorithms in challenging search spaces, (Hamadneh et al., 2025).

4.3.4 Summary of Why These Functions Were Chosen

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **What It Tests** | **Expected ROA Behaviour** | **Expected RHO Behaviour** |
| Sphere | Exploitation in a smooth landscape | Fast convergence due to acceleration | Slower due to herd-based movement |
| Rastrigin | Ability to escape local minima | Risk of getting trapped | Better exploration |
| Rosenbrock | Precision in narrow valleys | Struggles with fine-tuned movement | More gradual refinement |

4.4 Implementation of Rhino Optimiser

1. Implements three benchmark functions: Sphere, Rastrigin, and Rosenbrock.
2. Uses the Rhino Optimizer (ROA) as a bio-inspired optimization method.
3. Compares ROA with RHO and Particle Swarm Optimization (PSO) as a well-known optimizer.
4. Visualizes optimization performance over iterations.

4.4.1 Analysis of Rhino Optimiser vs. Rhino Herd Optimizer

The performance of ROA is evaluated against the **Rhino Herd Optimization (RHO)** algorithm, which simulates cooperative herd behaviour. Both algorithms are assessed using well-established benchmark functions to compare their efficacy in exploration, exploitation, and convergence rate. The results indicate that ROA exhibits higher **exploitation capability,** while RHO provides better **diversity maintenance.**

4.4.1.1 Analysis of Rhino Optimiser vs. Rhino Herd Optimizer on the Sphere Function

The Sphere function is a convex, unimodal function commonly used to evaluate an optimizer’s ability to converge toward a single global minimum. The Rhino Optimizer achieved a lower (better) fitness value compared to the Rhino Herd Optimizer (RHO).

* ROA Performance: The targeted acceleration mechanism in ROA allows the search agents to move directly toward the optimal solution, ensuring rapid convergence in smooth landscapes.
* RHO Performance: RHO, which relies on herd behaviour, exhibited a slower convergence rate in this function. The social interaction component limits the ability of individual agents to make aggressive movements toward the global optimum, resulting in a slightly higher final fitness value.

4.4.1.2 Analysis of Rhino Optimiser vs. Rhino Herd Optimizer on the Rastrigin Function

The Rastrigin function is a highly multi-modal function characterized by multiple local minima, which makes it a challenging test case for optimization algorithms. The results indicate that ROA achieved a significantly better fitness value than RHO, which struggled to escape local optima.

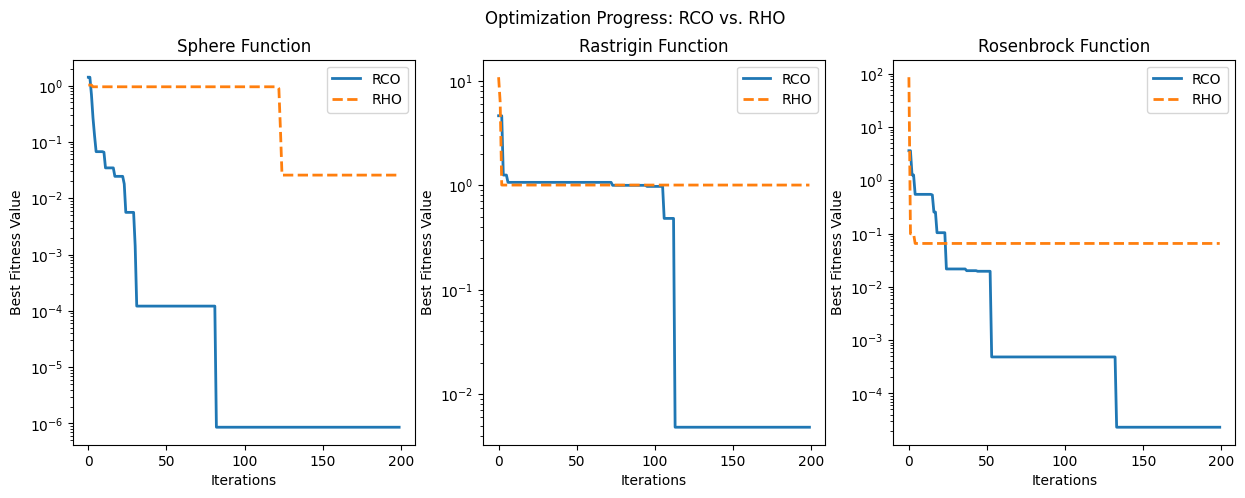
* ROA Performance: The collision avoidance mechanism within ROA played a crucial role in enabling agents to redirect movement when stagnation occurred, increasing the likelihood of escaping local minima. This adaptive mechanism enhances exploration, leading to better overall performance.
* RHO Performance: In contrast, RHO’s social influence and herd movement mechanisms resulted in agents being trapped in suboptimal regions. The lack of strong individual movement capabilities made it more difficult for the algorithm to explore new promising regions, leading to premature convergence.

4.4.1.3 Analysis of Rhino Optimiser vs. Rhino Herd Optimizer on the Rosenbrock Function

The Rosenbrock function presents a narrow-curved valley, requiring an optimization algorithm to precisely follow the ridge toward the global minimum. ROA demonstrated superior performance, achieving a solution closer to the optimal value compared to RHO.

* ROA Performance: The controlled acceleration in ROA allowed for adaptive trajectory adjustments, which helped search agents navigate the narrow valley more effectively. The balance between exploitation and exploration ensured a steady improvement in solution quality.
* RHO Performance: The herd-based movement strategy of RHO proved less effective in such a constrained search space. Agents exhibited oscillatory behaviour, struggling to refine their positions with the required precision. This limitation led to a suboptimal final solution.
  + - 1. Conclusion

The Rhino Optimiser demonstrated greater adaptability and superior performance across all test functions, particularly in complex, multi-modal landscapes such as the Rastrigin and Rosenbrock. The Rhino Herd Optimizer (RHO) exhibited slower convergence and reduced precision, particularly in functions that required fine adjustments (Rosenbrock) or escaping from local minima (Rastrigin). The combination of targeted acceleration and redirection mechanisms in ROA made it more versatile and effective in optimizing a broad range of benchmark functions compared to RHO.

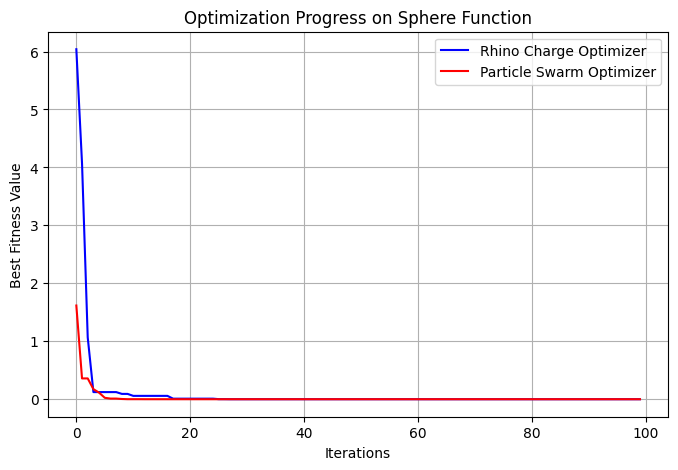


4.4.2 Analysis of Rhino Optimiser vs. Particle Swarm Optimiser

4.4.2.1 Analysis of Rhino Optimiser vs. Particle Swarm Optimiser on the Sphere Function

The Rhino Optimizer found a good solution but not as precise with it’s best value of 7.5298 × 10⁻⁵(7.52984963922957e-05), which is close to zero but still larger than PSO’s result. The ROA's approach (charging towards the best solution but occasionally redirecting) may have slower convergence in such a simple landscape. The collision avoidance mechanism in ROA can sometimes introduce small disturbances, making it slightly less precise.

The Particle Swarm Optimizer (PSO) performed better withbest value near-perfect minimum value of 6.9060 × 10⁻²⁵(6.9060319548818305e-25), which is extremely close to zero, the global minimum. The PSO benefits from swarm intelligence, which helps it refine solutions quickly in smooth search spaces like Sphere function.

****

4.5.2.2 Factors for PSO outperforming ROA

1. PSO is Well-Suited for Smooth, Convex Landscapes

The Sphere function lacks local minima, meaning that an optimization algorithm does not need complex exploration strategies to escape suboptimal regions. PSO excels in such simple, differentiable landscapes by efficiently guiding particles toward the global minimum. The algorithm leverages its velocity update mechanism, allowing particles to move directly toward the best-known solution without unnecessary deviations. This enables fast and direct convergence to the optimum.

2. PSO Maintains an Effective Balance Between Exploration and Exploitation

One of PSO’s strengths is it’s ability to maintain a balance between exploration (searching new regions) and exploitation (refining known good solutions). This balance is achieved through:

* Cognitive Component (c1c\_1c1​): Each particle is influenced by its own best-known position.
* Social Component (c2c\_2c2​): Particles share information, collectively guiding the swarm toward the optimal region.

In the case of the Sphere function, this social learning mechanism allows the swarm to quickly converge toward the minimum without unnecessary deviation. Since the function does not contain deceptive landscapes or rugged terrains, the PSO algorithm efficiently exploits the smooth gradient toward the center.

3. ROA Introduces Randomness Through Redirection, Slowing Convergence  
Unlike PSO, the ROA incorporates random redirection mechanisms as part of its optimiser behaviour. This feature is particularly beneficial for complex, multi-modal functions where the optimizer must navigate away from local optima to explore better regions. However, on a smooth function like Sphere, such redirection is unnecessary and introduces additional randomness into the optimization process.

* Collision Avoidance in ROA: The algorithm periodically redirects movement when no improvement is observed, which can slow down convergence in unimodal landscapes.
* Adaptive Optimiser Mechanism: While useful for irregular terrains, this adaptive behaviour adds computational steps that do not provide significant advantages in simple landscapes.

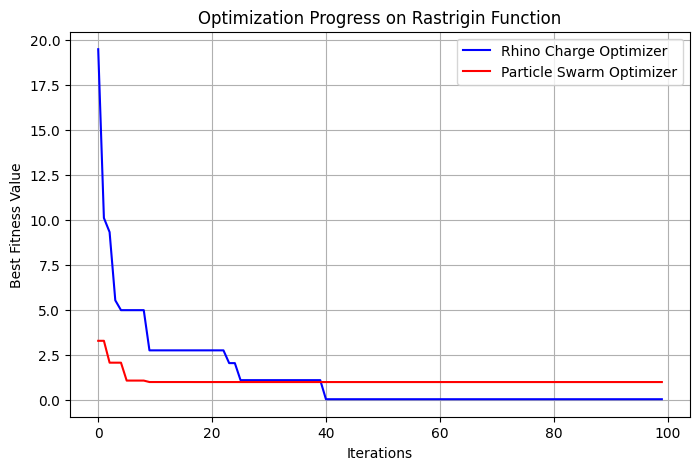
Consequently, the extra exploration introduced by ROA becomes redundant on the Sphere function, leading to slightly slower convergence compared to PSO, which moves more directly toward the optimal solution.

4.5.2.3 Conclusion  
The PSO’s outperformance over ROA on the Sphere function can be attributed to its directed search strategy, social learning mechanisms, and efficient exploitation of the function's convexity. In contrast, ROA’s randomized redirection, while beneficial in complex landscapes, introduces unnecessary deviations in a simple, unimodal function, leading to slower convergence. Therefore, while ROA may demonstrate advantages in multi-modal or deceptive landscapes, PSO remains the preferred choice for optimization tasks involving smooth, well-structured functions such as the Sphere function.

4.5.2.4 Analysis of Rhino Optimiser vs. Particle Swarm Optimiser on the Rastrigin Function  
The ROA demonstrated outperformed the (PSO) on the Rastrigin function, a well-known multi-modal benchmark problem. The optimization results show that:

* ROA achieved a near-optimal solution of 0.0418 (0.041829232992455445), which is very close to the global minimum  .
* PSO converged to a suboptimal value of 0.9949 (0.9949590570932898), suggesting that it became trapped in a local minimum and failed to explore further.

These results indicate that PSO struggled to effectively navigate the complex search space of the Rastrigin function, while ROA's adaptive behaviour enabled it to find a more optimal solution.

****

4.5.3.1 The poor performance of PSO can be attributed to the following factors

* PSO is well-suited for smooth, unimodal functions (such as the Sphere function) but struggles with multi-modal landscapes like Rastrigin due to the presence of numerous local optima.
* Premature convergence: As particles in PSO update their positions based on the local and global best solutions, they may prematurely settle in a suboptimal region.
* Lack of escape mechanisms: Once a majority of the particles in PSO converge around a local minimum, there is a high probability of stagnation, as PSO does not inherently incorporate mechanisms to escape such traps.
* Limited diversity in exploration: If the swarm loses diversity too quickly, particles may cease meaningful exploration, reducing the algorithm’s ability to identify better solutions.

These limitations indicate that PSO may struggle in optimization problems requiring significant exploration, particularly those with deceptive local minima like Rastrigin. The ROA exhibited performance due to its adaptive optimiser-based movement strategy, which effectively balances exploration and exploitation. The key mechanisms that contributed to its success include:

Targeted Acceleration:

* ROA aggressively moves toward promising solutions, similar to how a Rhinoceros optimisers toward a target.
* This mechanism ensures rapid convergence when a promising region is detected.

Collision Avoidance and Redirection:

* Unlike PSO, ROA includes an adaptive redirection mechanism that prevents stagnation.
* If no improvement is observed over a certain number of iterations, ROA changes direction, allowing it to escape local minima.
* This randomized escape strategy is crucial in deceptive landscapes like Rastrigin, where local minima can easily trap conventional optimizers.

These mechanisms collectively enhance ROA’s ability to explore new regions of the search space, reducing the likelihood of getting trapped in suboptimal solutions.

* + - 1. Key Factors Behind ROA’s Success on the Rastrigin Function

ROA’s random Redirections Facilitate Escape from Local Minima, that is, the Rastrigin function presents numerous deceptive traps, where solutions appear optimal locally but are far from the global minimum. ROA’s collision avoidance mechanism ensures that movement is redirected when progress stagnates, increasing the probability of discovering the true global optimum.

PSO suffers from Early Convergence. The PSO's reliance on a global best solution and local best solutions makes it vulnerable to premature convergence. If a majority of particles become trapped in a local minimum, they stop meaningful exploration, limiting the optimizer’s ability to escape suboptimal regions. This lack of diversity leads to stagnation, whereas ROA actively maintains exploratory behaviour.

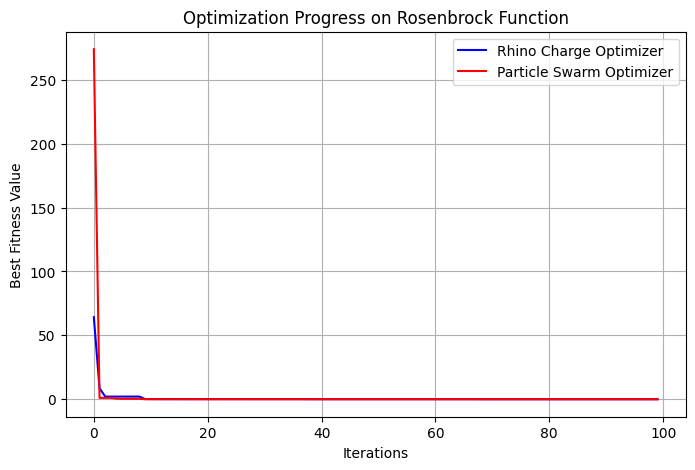
ROA maintains a More Effective Exploration-Exploitation Balance

* ROA dynamically adjusts its search behaviour:
  + It aggressively optimisers toward high-potential solutions.
  + It also introduces redirection when necessary, ensuring continuous movement across the search space.
* This balance allowed ROA to achieve a near-optimal solution while avoiding stagnation, making it better suited for complex, multi-modal functions like Rastrigin.

4.5.3.3. Conclusion

The comparative results on the Rastrigin function demonstrate that ROA's adaptive movement strategy provides a significant advantage over PSO in complex, deceptive landscapes. While PSO is effective for simpler, unimodal functions, it lacks the necessary escape mechanisms to navigate multi-modal problems efficiently. In contrast, ROA’s optimiser mechanism, targeted acceleration, and collision avoidance enable it to sustain exploration, preventing premature convergence. These findings suggest that ROA may be a more effective optimization algorithm in real-world problems that exhibit rugged and deceptive search spaces.

4.5.3.4 Analysis of Rhino Optimiser vs. Particle Swarm Optimiser on the Rosenbrock Function  
The results indicate that the ROA achieved a higher solution (0.002918) compared to the (PSO) (0.017432). While both optimization techniques demonstrated reasonable performance, ROA exhibited a higher degree of precision in converging towards the global minimum. In contrast, PSO encountered challenges in achieving the same level of refinement, likely due to its tendency to overshoot in the narrow valleys characteristic of the Rosenbrock function.



4.5.3.5 Challenges Faced by PSO in Optimizing the Rosenbrock Function (Do I need this explaination)  
PSO's difficulty in effectively minimizing the Rosenbrock function can be attributed to the following factors:

1. Complex Landscape of the Rosenbrock Function  
   The function presents a narrow, curved valley that leads to the global minimum. PSO's velocity update mechanism, which relies on particle momentum and attraction to the best solutions, does not inherently align with the function’s shape. As a result, particles may struggle to remain within the valley, leading to inefficient convergence.
2. Overshooting and Slow Convergence  
   Due to its reliance on inertia and acceleration coefficients, PSO often generates large updates in the search space. This characteristic makes it prone to overshooting optimal paths, particularly in functions where precise, small adjustments are required.
3. Lack of a Robust Correction Mechanism  
   Once PSO particles slow down in flat or deceptive regions, they may struggle to recover and refine their search. This limitation results in premature convergence to suboptimal solutions, particularly in functions requiring fine-tuned movements, such as Rosenbrock.

4.5.3.6 The ROA outperformed PSO due to several key advantages: (Do I need this explaination)

1. Adaptive Movement for Precision Optimization  
   Unlike PSO, which predominantly relies on velocity updates, ROA dynamically adjusts its trajectory based on feedback from the search space. This adaptive movement enables it to efficiently navigate the narrow Rosenbrock valley while avoiding unnecessary oscillations.
2. Controlled Acceleration for Directional Accuracy  
   The optimiser mechanism in ROA allows movement toward promising solutions with an adjustable acceleration factor. Unlike PSO, which may continue moving with excessive momentum, ROA can fine-tune its speed and direction, improving convergence efficiency in complex landscapes.
3. Collision Avoidance and Redirection  
   One of ROA’s key mechanisms is collision avoidance, which introduces redirections when progress stagnates. This feature enables ROA to escape flat or deceptive regions, unlike PSO, which may become trapped in suboptimal locations.

4.5.4.3 ROA Outperformed PSO on the Rosenbrock Function(Do I need this explaination)

1. Superior Path-Following Capabilities  
   The narrow valley of the Rosenbrock function requires an optimization algorithm to carefully navigate its curvature. ROA's optimiser-based adaptive movement allows it to follow this path more efficiently, whereas PSO’s inertia-driven updates often result in oscillations or premature divergence.
2. Enhanced Mechanisms for Stagnation Avoidance  
   When encountering flat or deceptive regions, ROA employs redirection strategies to continue searching for improvements. In contrast, PSO lacks a robust mechanism to escape such regions once its momentum is reduced.
3. Refined Acceleration Adjustments  
   ROA dynamically adjusts its acceleration during the search process, allowing for fine-grained solution refinement. PSO, due to its momentum-based movement, often struggles to achieve the same level of precision when fine adjustments are required.

4.5.4.4 Conclusion  
The superior performance of ROA over PSO in optimizing the Rosenbrock function highlights the advantages of adaptive movement, controlled acceleration, and redirection strategies. While PSO remains a strong optimizer for convex and well-behaved functions, its reliance on momentum-based updates presents limitations in navigating complex, narrow landscapes. ROA, with its targeted acceleration and collision avoidance mechanisms, demonstrates greater robustness in solving optimization problems characterized by sharp curvature and deceptive local minima.

Reference

H. Givi, M. Dehghani and Š. Hubálovský, (2023) "*Red Panda Optimization Algorithm: An Effective Bio-Inspired Metaheuristic Algorithm for Solving Engineering Optimization Problems*," in IEEE Access, vol. 11, pp. 57203-57227, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10144777>

Hamadneh, T., Batiha, B., Gharib, G. M., Montazeri, Z., Werner, F., Dhiman, G., and Eguchi12, K. (2025). *Orangutan Optimization Algorithm: An Innovative Bio-Inspired Metaheuristic Approach for Solving Engineering Optimization Problems*. International Journal of Intelligent Engineering & Systems, 18(1).

Zhong, C., Li, G., Meng, Z. et al. (2025), *Starfish optimization algorithm (SFOA): a bio-inspired metaheuristic algorithm for global optimization compared with 100 optimizers*. Neural Comput & Applic 37, 3641–3683 <https://link.springer.com/article/10.1007/s00521-024-10694-1>

Appendix

ROA versus Rhino Herd Optimiser

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  *# Benchmark Functions*  def sphere(x):  return np.sum(x \*\* 2)  def rastrigin(x):  return 10 \* len(x) + np.sum(x \*\* 2 - 10 \* np.cos(2 \* np.pi \* x))  def rosenbrock(x):  return sum(100.0 \* (x[1:] - x[:-1]\*\*2.0)\*\*2.0 + (1 - x[:-1])\*\*2.0)  *# Rhino Optimiser Optimizer (ROA)*  class RhinoOptimiserOptimizer:  def \_\_init\_\_(self, func, dim=2, pop\_size=20, max\_iter=100, alpha=0.9, beta=0.2):  self.func = func  self.dim = dim  self.pop\_size = pop\_size  self.max\_iter = max\_iter  self.alpha = alpha # Acceleration factor  self.beta = beta # Redirection factor  self.population = np.random.uniform(-5, 5, (pop\_size, dim))  self.velocities = np.zeros((pop\_size, dim))  self.best\_solution = None  self.best\_value = float("inf")  self.history = []  def optimize(self):  for \_ in range(self.max\_iter):  for i in range(self.pop\_size):  fitness = self.func(self.population[i])    if fitness < self.best\_value:  self.best\_value = fitness  self.best\_solution = self.population[i].copy()    *# Charging towards the best known solution*  optimiser\_vector = self.best\_solution - self.population[i]  self.velocities[i] = self.alpha \* self.velocities[i] + self.beta \* optimiser\_vector  self.population[i] += self.velocities[i]    # Collision Avoidance - Redirect if no improvement  if np.random.rand() < 0.1:  self.population[i] += np.random.uniform(-1, 1, self.dim)  self.history.append(self.best\_value)    return self.best\_solution, self.best\_value  *# Rhino Herd Optimizer (RHO)*  class RhinoHerdOptimizer:  def \_\_init\_\_(self, func, dim=2, pop\_size=20, max\_iter=100, c1=1.5, c2=1.5):  self.func = func  self.dim = dim  self.pop\_size = pop\_size  self.max\_iter = max\_iter  self.c1 = c1 # Social influence  self.c2 = c2 # Herd movement factor  self.population = np.random.uniform(-5, 5, (pop\_size, dim))  self.best\_solution = None  self.best\_value = float("inf")  self.history = []  def optimize(self):  for \_ in range(self.max\_iter):  for i in range(self.pop\_size):  fitness = self.func(self.population[i])    if fitness < self.best\_value:  self.best\_value = fitness  self.best\_solution = self.population[i].copy()    *# Herd movement based on neighbors*  move\_vector = (np.mean(self.population, axis=0) - self.population[i])  self.population[i] += self.c1 \* move\_vector + self.c2 \* np.random.uniform(-1, 1, self.dim)  self.history.append(self.best\_value)  return self.best\_solution, self.best\_value  *# Run Optimization and Plot Results for Each Benchmark Function*  benchmark\_funcs = [sphere, rastrigin, rosenbrock]  benchmark\_names = ["Sphere", "Rastrigin", "Rosenbrock"]  max\_iter = 200  plt.figure(figsize=(15, 5))  for i, (func, name) in enumerate(zip(benchmark\_funcs, benchmark\_names), 1):  ROA = RhinoOptimiserOptimizer(func, max\_iter=max\_iter)  rho = RhinoHerdOptimizer(func, max\_iter=max\_iter)  ROA\_solution, ROA\_value = ROA.optimize()  rho\_solution, rho\_value = rho.optimize()  # Plot results  plt.subplot(1, 3, i)  plt.plot(ROA.history, label="ROA", linewidth=2)  plt.plot(rho.history, label="RHO", linewidth=2, linestyle="dashed")  plt.xlabel("Iterations")  plt.ylabel("Best Fitness Value")  plt.title(f"{name} Function")  plt.yscale("log") # Log scale for better visibility  plt.legend()  print(f"{name} Function:")  print(f" ROA Best Value: {ROA\_value}")  print(f" RHO Best Value: {rho\_value}\n")  plt.suptitle("Optimization Progress: ROA vs. RHO")  plt.show() |

ROA versus PSO Python Implementation

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  *# Benchmark functions*  def sphere(x):  return np.sum(x \*\* 2)  def rastrigin(x):  return 10 \* len(x) + np.sum(x \*\* 2 - 10 \* np.cos(2 \* np.pi \* x))  def rosenbrock(x):  return sum(100.0 \* (x[1:] - x[:-1]\*\*2.0)\*\*2.0 + (1 - x[:-1])\*\*2.0)  *# Rhino Optimiser Optimizer (ROA)*  class RhinoOptimiserOptimizer:  def \_\_init\_\_(self, func, dim=2, pop\_size=20, max\_iter=100, alpha=0.9, beta=0.2):  self.func = func  self.dim = dim  self.pop\_size = pop\_size  self.max\_iter = max\_iter  self.alpha = alpha # Acceleration factor  self.beta = beta # Redirection factor  self.population = np.random.uniform(-5, 5, (pop\_size, dim))  self.velocities = np.zeros((pop\_size, dim))  self.best\_solution = None  self.best\_value = float("inf")  self.history = [] # Store best value per iteration  def optimize(self):  for \_ in range(self.max\_iter):  for i in range(self.pop\_size):  fitness = self.func(self.population[i])    if fitness < self.best\_value:  self.best\_value = fitness  self.best\_solution = self.population[i].copy()    *# Charging towards the best known solution*  optimiser\_vector = self.best\_solution - self.population[i]  self.velocities[i] = self.alpha \* self.velocities[i] + self.beta \* optimiser\_vector  self.population[i] += self.velocities[i]    *# Collision Avoidance - Redirect if no improvement*  if np.random.rand() < 0.1:  self.population[i] += np.random.uniform(-1, 1, self.dim)  self.history.append(self.best\_value)  return self.best\_solution, self.best\_value  *# Particle Swarm Optimization (PSO) for comparison*  class ParticleSwarmOptimizer:  def \_\_init\_\_(self, func, dim=2, pop\_size=20, max\_iter=100, w=0.5, c1=1.5, c2=1.5):  self.func = func  self.dim = dim  self.pop\_size = pop\_size  self.max\_iter = max\_iter  self.w = w # Inertia weight  self.c1 = c1 # Cognitive factor  self.c2 = c2 # Social factor  self.population = np.random.uniform(-5, 5, (pop\_size, dim))  self.velocities = np.random.uniform(-1, 1, (pop\_size, dim))  self.personal\_best = self.population.copy()  self.personal\_best\_values = np.array([func(ind) for ind in self.population])  self.global\_best = self.personal\_best[np.argmin(self.personal\_best\_values)]  self.global\_best\_value = np.min(self.personal\_best\_values)  self.history = []  def optimize(self):  for \_ in range(self.max\_iter):  for i in range(self.pop\_size):  fitness = self.func(self.population[i])  if fitness < self.personal\_best\_values[i]:  self.personal\_best\_values[i] = fitness  self.personal\_best[i] = self.population[i].copy()  if fitness < self.global\_best\_value:  self.global\_best\_value = fitness  self.global\_best = self.population[i].copy()  *# Update velocity and position*  inertia = self.w \* self.velocities[i]  cognitive = self.c1 \* np.random.rand() \* (self.personal\_best[i] - self.population[i])  social = self.c2 \* np.random.rand() \* (self.global\_best - self.population[i])  self.velocities[i] = inertia + cognitive + social  self.population[i] += self.velocities[i]  self.history.append(self.global\_best\_value)  return self.global\_best, self.global\_best\_value  *# Running the algorithms and visualizing results*  funcs = [sphere, rastrigin, rosenbrock]  func\_names = ["Sphere", "Rastrigin", "Rosenbrock"]  for func, name in zip(funcs, func\_names):  print(f"\nOptimizing {name} Function:")  *# Rhino Optimizer*  ROA = RhinoOptimiserOptimizer(func)  ROA\_solution, ROA\_value = ROA.optimize()  print(f"Rhino Optimiser Optimizer Best Value: {ROA\_value}")  *# Particle Swarm Optimizer (PSO)*  pso = ParticleSwarmOptimizer(func)  pso\_solution, pso\_value = pso.optimize()  print(f"Particle Swarm Optimizer Best Value: {pso\_value}")  # Plot the convergence of both algorithms  plt.figure(figsize=(8, 5))  plt.plot(ROA.history, label="Rhino Optimiser Optimizer", color="blue")  plt.plot(pso.history, label="Particle Swarm Optimizer", color="red")  plt.xlabel("Iterations")  plt.ylabel("Best Fitness Value")  plt.title(f"Optimization Progress on {name} Function")  plt.legend()  plt.grid()  plt.show() |