Optimization in Econometrics and Machine Learning: A Study of Particle Swarm Optimization (PSO)

Théo Linale
Victoria Vidal
Anastasiia Savelkova

Master 1 Econometrics Statistics

Paris I Panthéon-Sorbonne

Applied Econometrics

M. De Peretti

January 2025

Abstract

In econometrics and Machine Learning, optimizing an objective function (likelihood, cost function, etc.) is required. However, most algorithms return local optima, such as gradient descent or Newton-Raphson. In recent years, new classes of algorithms have emerged, including the particle Swarm optimization (PSO) algorithm. This paper aims to present the PSO algorithm and its recent developments, followed by an implementation in IML. The algorithms will then be applied to various functions, both linear and highly non-linear, for validation. A comparison with gradient descent and the SAS `PROC OPTMODEL` procedure is also provided, with the objective of outperforming the latter.

Keywords: proc iml, proc optmodel, optimization, nonlinear functions, computing time.

Contents

1 Introduction

Optimization plays an important role in econometrics and Machine Learning. It is used to adjust models in order to improve the accuracy of predictions. Finding the best parameters for a model involves minimising a cost function, such as the mean square error or the negative likelihood. Following this minimisation helps to obtain reliable and usable predictions.

In econometrics, optimization is involved in the estimation of the parameters of statistical models, particularly in linear regression where it is used to minimize the sum of the squares of the errors. For more complex models, such as neural networks or Bayesian models, there are suitable strategies to be able to efficiently explore the search space.

Classical optimization methods, such as gradient descent and the Newton-Raphson algorithm, have long dominated the field. Gradient descent will progressively adjust the parameters by following the downward slope of the cost function. However, one major problem stands out: the presence of local optima. The presence of local optima can prevent convergence towards the best solution. In addition, the choice of learning rate is crucial to avoid slow convergence or divergence. The Newton-Raphson algorithm uses second derivatives to speed up the search for the optimum. It offers faster convergence when the function is well conditioned. However, it depends heavily on the Hessian matrix, which can be a problem in the case of ill-defined or non-convex functions. It can also be trapped in local minima, making optimization as unstable as gradient descent.

Faced with these limitations, researchers have developed algorithms called metaheuristics. These algorithms are capable of exploring the search space more widely. Genetic algorithms, inspired by natural selection, will evolve a population of solutions by combining mutation and crossover. There is also the simulated annealing algorithm, which is based on the principle of gradual cooling. It temporarily accepts less optimal solutions to avoid premature deadlock.

PSO stands out for its balance between exploration and exploitation. It was inspired by the collective behaviour of birds and fish. It is based on a population of particles that adjust their position according to two influences: their own experience and that of the group. This dynamic will then enable a more efficient search for the global optimum while reducing the risk of stagnation.

Paper Objectives

In this paper, we will discuss optimization and its importance in econometrics and machine learning. Conventional methods such as gradient descent or the Newton-Raphson algorithm have a number of limitations. They can get stuck on solutions that are not the best. This is why we are looking for more efficient alternatives.

These new approaches include bio-inspired algorithms, which are inspired by the behaviour of nature. We will first present evolutionary algorithms, such as genetic algorithms and ant colonies, before introducing the PSO algorithm. We will explain why it is interesting and how it works. Next, we will look at its advantages and limitations. An important part of this work will consist of implementing PSO under `PROC IML` in SAS and applying it to different optimization functions, whether linear or non-linear. We will compare its results with those of gradient descent and the `PROC OPTMODEL` algorithm in SAS to see which is more efficient in

terms of speed and accuracy.

Our aim is to answer the following question :

Under what conditions can PSO outperform traditional optimization methods in econometrics and machine learning, and how can it be implemented effectively in a SAS environment ?

2 Importance of Optimization in Econometrics and Machine Learning

2.1 Importance of Optimization

The main goal of optimization is often to adjust statistical models to data to extract meaningful insights. In econometrics, optimization is used for parameter estimation through methods such as Ordinary Least Squares (OLS) and Maximum Likelihood Estimation (MLE), which help refine regression models and time series analysis. These techniques allow models to better describe relationships between economic or social variables.

In econometrics, advanced models such as ARCH (Autoregressive Conditional Heteroskedasticity) and GARCH (Generalized Autoregressive Conditional Heteroskedasticity) are widely used to model the volatility of financial time series. These models capture the temporal dependence of conditional variance and are particularly suited for financial market analysis.

However, they require efficient optimization algorithms to avoid getting stuck in local optima. Bayesian approaches also rely on optimization through algorithms like Markov Chain Monte Carlo (MCMC) and variational methods, which enable sampling from complex probability distributions by efficiently exploring high-dimensional spaces. These algorithms are essential for Bayesian statistical inference, but their performance heavily depends on the underlying optimization process.

Econometrics faces major challenges, particularly when dealing with nonlinear models or high-dimensional data. These complexities make parameter estimation significantly more difficult and require appropriate solutions to ensure both model convergence and accuracy. Machine Learning, on the other hand, heavily relies on optimization to fine-tune models based on data. For instance, in neural networks, gradient descent is used to adjust the weights of connections between neurons. Similarly, Support Vector Machines (SVM) use quadratic optimization techniques to optimally classify data. Clustering algorithms such as K-Means and reinforcement learning techniques also employ optimization procedures to maximize a reward function or improve data segmentation.

However, these approaches face several challenges, including the non-convexity of cost functions, sensitivity to initial conditions, and increasing computational complexity. These limitations have led to the rise of new approaches inspired by biological and social phenomena, offering more robust alternatives to traditional methods.

2.2 Classical Methods

2.2.1 Gradient Descent

Gradient descent is a method used for minimizing differentiable functions. It is based on an iterative update of parameters following the direction of the gradient of the objective function.

The equation is as follows :

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$$

where :

- θ_t : the model parameters at iteration t.
- $\nabla f(\theta_t)$: the gradient of the objective function f evaluated in θ_t, indicating the direction of the steepest slope.
- α : the learning rate, which controls the size of the update step.

Functioning

1. We start with an arbitrary initial point.
2. We compute the gradient of the function at this point, which represents the direction in which the function increases the fastest.
3. We adjust the position by taking a small step in the opposite direction of the gradient since we aim to minimize the function.
4. We repeat this process until reaching a point where the gradient is close to zero.
   This indicates that the algorithm has found a minimum.

Limitations

One of the main drawbacks of this method is its high sensitivity to initial conditions and the shape of the objective function's landscape. When the function to minimize has many local minima, gradient descent can get trapped in one of them, preventing it from reaching the global optimum.

This issue is particularly common in complex models such as :

- **Deep neural networks**
  The cost function is often highly non-convex, with multiple valleys and plateaus.
- **High-dimensional optimization problems**
  The gradients can become extremely small, significantly slowing down convergence.
- **Systems with noise or discontinuities**
  Gradient-based updates may become unstable or ineffective in these cases.

2.2.2 Newton-Raphson Algorithm

Optimization using the Newton-Raphson algorithm combines both the gradient and the curvature of the function. Instead of performing a standard parameter update, this algorithm applies a correction based on the second derivative. As a result, after several iterations, it approaches the solution, allowing much faster convergence when a quadratic model solution can be used as an approximation to the function.

Functioning

The algorithm follows an iterative approach :

1. **Initialization**
   It starts with an initial estimate of the solution, denoted as θ_t.

2. **Derivative computation**
   at each step, it evaluates:

   - The gradient $\nabla f(\theta_t)$, which indicates the direction of the steepest ascent of the function.

   - The Hessian matrix $H_f(\theta_t)$, which contains the second derivatives and describes the local curvature of the function.

3. **parameter update**
   The new value is computed using the formula:

$$\theta_{t+1} = \theta_t - H_f(\theta_t)^{-1} \nabla f(\theta_t)$$

4. **Repetition**
   The process is repeated until a stopping criterion is met, usually when the difference between θ_t and θ_{t+1} becomes negligible.

With this update, the Newton-Raphson algorithm adjusts the search trajectory and can reach an optimum in fewer iterations than gradient descent.

Why is this method fast ?

The Newton-Raphson algorithm is faster than gradient descent.

It takes into account the curvature of the function and, thanks to the Hessian matrix, adjusts both the direction and magnitude of the step. This avoids steps that are too short or inefficient. In addition, unlike gradient descent, the algorithm does not require a learning rate. The algorithm automatically adapts the step size at each iteration. This eliminates the risk of too large a step causing divergence or too small a step slowing convergence.

Finally, Newton-Raphson reaches the optimum in fewer iterations than gradient descent. Therefore, if the function is well approximated by a quadratic model, the algorithm quickly finds the

best solution.

It should be remembered that these properties make the Newton-Raphson algorithm effective for problems where the Hessian is easy to calculate and well conditioned.

Limitations

Despite its speed, the Newton-Raphson method has a number of limitations.

The first is the cost of calculating the Hessian, particularly for high-dimensional functions. Inverting this matrix requires significant resources and slows down the algorithm, which can then diverge. If the Hessian is poorly defined or if the initial estimate is too far from the optimum, the updates become unstable, preventing optimization.

The method is particularly sensitive to noisy or discontinuous functions. The uncertainty of the second derivative makes the fits less reliable. In addition, although this is not a specific drawback of Newton-Raphson, like many grid-based algorithms, it can also get stuck in a local minimum.

Ultimately, this method does not guarantee finding the best absolute optimum, but rather the best achievable optimum from the starting point.

3 Rise of Bio-Inspired Algorithms

3.1 Introduction to Evolutionary Algorithms

For a long time, numerical optimization has relied on classical methods such as gradient descent and the Newton-Raphson algorithm. Although these techniques are effective, they reach their limits in complex search spaces. They run the risk of getting stuck in local minima when there are several optimal solutions or when the gradients are difficult to exploit.

To overcome these difficulties, a new approach has emerged: bio-inspired algorithms. These algorithms imitate natural and social phenomena to explore the space of solutions. The principle is as follows: instead of refining a single solution, they work with an entire population of individuals who interact and evolve over the course of iterations.

Evolutionary algorithms have established themselves as powerful tools. They include genetic algorithms, inspired by natural selection, and ant colony optimization, which models the collective foraging behavior of ants. Another example is particle swarm optimization, which is inspired by the dynamics of schools of fish and flocks of birds.

These approaches take advantage of collective intelligence to efficiently explore the search space and avoid stagnating in sub-optimal solutions.

3.1.1 Genetic Algorithms and Ant Colonies

The idea of applying population-based approaches to optimization emerged in the 1960s and 1970s, at a time when many researchers were looking for optimization models inspired by life

and evolution. One of the first algorithms of this type was the genetic algorithm, formalised by John Holland in 1975. The central idea behind genetic algorithms is to artificially reproduce Darwinian principles such as natural selection, crossover and mutation, in order to move a 'population' of potential solutions towards the best overall optimum.

Ant colony optimization (ACO), developed by Marco Dorigo in 1992, is based on the way ants find the shortest route to a food source using pheromone trails. In 1995, James Kennedy and Russell Eberhart introduced PSO, a model inspired by the cooperative behaviour of individuals to efficiently explore a search space.

3.1.2 Introduction to PSO

PSO is based on a principle of collective intelligence inspired by the animal world. In nature, many groups of animals use cooperative strategies to improve their chances of survival. When a bird searches for a source of food, it adjusts its trajectory not only according to its own discoveries, but also those of the other members of the group. This analogy is repeated in numerical optimization algorithms.

Each particle represents a candidate solution and moves through the search space as a function of two factors. Firstly, it considers the best position it has found so far. Secondly, it is influenced by the most recent and most successful solution found by the group as a whole. At each iteration, the particle takes these two influences into account and adjusts its position accordingly.

For this reason, the solution tends to converge towards an optimum without the need for derivative calculations.

3.1.3 Why Use PSO?

PSO is particularly well suited to problems for which traditional methods reach their limits. It is effective for optimizing complex functions with multiple local minima and maxima, thus avoiding premature convergence. It is also useful for non-differentiable functions, where the absence of gradients makes methods such as gradient descent inapplicable. Finally, PSO is robust to noisy functions, where random perturbations can distort conventional calculations.

Unlike genetic algorithms, which rely on random mutations and recombinations, PSO updates positions in a smoother, more gradual way. This results in much faster convergence, improving its effectiveness in many situations.

3.1.4 Flexible and Efficient Approach

The rise of bio-inspired algorithms has introduced a completely different logic to traditional methods. Instead of progressively refining a single solution, these algorithms are based on the idea that collective intelligence can overcome any challenge, provided that a sufficiently wide range of solutions is explored.

This philosophy has earned PSO and its counterparts a prominent place in artificial intelligence, engineering, finance, and scientific research in general. Evolutionary approaches offer

unmatched flexibility and adaptability when optimizing solutions for complex problems.

3.2 Operating Principle of PSO

PSO is a group-based approach to solving optimization problems. The model is inspired by the behaviour of natural bird swarms, where each individual adjusts its trajectory according to its own experience. Instead of using a deterministic model that relies on gradients for final tracking, PSO dynamically updates particle positions based on the best results.

The algorithm works iteratively, balancing exploration and exploitation. Exploration allows the search to cover different areas of the solution space, preventing the algorithm from getting stuck in a local minimum, while exploitation allows promising solutions to be refined. Several key parameters control this equilibrium, in particular the inertia factor w, which regulates the influence of past movements on the current trajectory.

One of PSO's strengths lies in its adaptive mechanism. Instead of generating mutated and mutated and recombined solutions like genetic algorithms, PSO continually adjusts the trajectories of particles based on their past performance. This makes it a potentially fast and efficient optimization method.

3.2.1 Definition of Key Elements

Each particle represents a candidate solution in the search space and has several fundamental attributes :

- **Position** (x_i)
  Represents the particle's current state in the solution space.

- **Velocity** (v_i)
  Determines the direction and magnitude of movement at each iteration.

- **Best individual position** (p_i)
  The best solution found by the particle since the start of the algorithm.

- **Best global position** (g)
  The best solution identified by the entire swarm.

One of the fundamental principles of the PSO is the collective influence exerted by the best overall particle. This phenomenon, similar to the social behaviour observed in nature, encourages rapid convergence towards optimal solutions. However, if the diversity within the group decreases too quickly, the algorithm runs the risk of getting stuck in a local optimum.

To avoid this problem, some variants introduce specific strategies :

- Adding random perturbationsto reintroduce diversity into the search.

- Local neighbourhood PSO, where each particle is only influenced by a subset of neighbours rather than the whole swarm, which improves exploration.

3.2.2 Algorithm Structure

The algorithm follows an iterative structure and consists of three main steps :

1. **Initialization**

   - The initial positions and velocities of the particles are randomly generated.
   - The objective function is evaluated for each particle.
   - The initial values of p_i (best individual position) and g (best global position) are set.

2. **Particle Update (Main Loop)**

   At each iteration, the particles adjust their positions based on the best solutions found. This process consists of two components:

   - **Individual influence**: each particle is attracted to its best previous position p_i.
   - **Collective influence**: each particle is influenced by the best solution found by the swarm g.

   These forces are combined using the following equations:

   Velocity Update

   $$v_i^{(t+1)} = wv_i^{(t)} + c_1 r_1 (p_i - x_i^{(t)}) + c_2 r_2 (g - x_i^{(t)})$$

   where:

   - w is the **inertia factor**, which controls the influence of the previous movement.
   - c_1 and c_2 are acceleration coefficients that weight the influence of the best individual and collective solutions.
   - r_1 and r_2 are random numbers between 0 and 1, introducing a stochastic factor to avoid premature convergence.

   Position Update

   $$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$$

   After each update, the new positions are evaluated, and the values of p_i and g are adjusted if better solutions are found.

3. **Stopping Criterion**

   The algorithm stops if:

- A maximum number of iterations is reached.
- The variation of g becomes negligible, indicating convergence.

3.2.3 PSO Parameters and Convergence

Consequently, the balance between exploration and exploitation is crucial for the PSO.

Exploration allows particles to explore the solution space and avoid local minima, while exploitation refines solutions by focusing research on promising areas. This balance can be achieved by fine-tuning the parameters to ensure that the PSO algorithm converges efficiently. Incorrect parameter settings can slow down the algorithm, limit exploration to an insufficient level or increase the risk of error. The algorithm, limit exploration to an insufficient level or increase the risk of sub-optimal convergence.

Inertia Factor w

The inertia factor w regulates the influence of the previous velocity on the current movement of the particles.

- A high value maintains strong dynamics and encourages wide exploration.
- A low value slows down the particles and favours the exploitation of the best solutions found.

Optimal adjustment

Dynamic adjustment of w often improves convergence.
For example, a gradual decrease from 0.9 to 0.4 allows a wide exploration at first, followed by a concentration on the optimal solutions. Some PSO variants adjust w as a function of particle dispersion :

- If they are too dispersed, w decreases.
- If they are too concentrated, w increases to boost exploration.

Acceleration Coefficients c_1 and c_2

These coefficients play a key role in the movement of particles by influencing their attraction to their own best position and the best global position.

- The c_1 coefficient increases local exploration by encouraging each particle to refine its own discoveries. its own discoveries.
- On the other hand, the c_2 coefficient encourages faster convergence by pulling the particles towards the best solution found by the swarm as a whole.

Effects of incorrect parameter settings

- A high c_1 leads to excessive exploration, which slows down the convergence of the algorithm.
- A high c_2 can lead to premature convergence, limiting exploration and increasing the risk of missing the global optimum.

A good balance is often $c_1 = 2.0$ and $c_2 = 2.0$ ensuring an effective synergy between individual autonomy and collective learning.

Particle Population Size

The number of particles influences both the diversity of the exploration and the speed of convergence.

- With fewer than 20 particles, execution is faster, but the risk of premature convergence to a local optimum is higher.
- With more than 50 particles, the exploration is deeper, but the computation time increases considerably.

How to Choose the Right Size?

The optimum range is generally between 20 and 50 particles for most problems. A rule of thumb suggests that :

$$N = 10D$$

where D is the number of dimensions in the problem.

A moderate population size provides a good balance between diversity and efficiency.

Shutdown criteria

PSO stops according to two types of criteria: :

- Convergence: the improvement in the best overall solution becomes negligible.
- Number of iterations: the algorithm reaches a predefined threshold.

Balance Between Accuracy and Calculation Time

Finding the right balance between accuracy and computation time is essential to guarantee optimal results.

- A strict stopping criterion can terminate the algorithm too early, leading to a sub-optimal solution.

- Conversely, a criterion that is too flexible needlessly prolongs the calculation time without any significant gain in performance.

To solve this problem, some adaptive strategies dynamically adjust these criteria to optimise the trade-off between execution speed and solution quality.

Influence of Parameters on Convergence

Incorrect parameter settings can reduce the efficiency of the PSO and degrade its performance. An incorrect inertia factor can either slow down exploration or, on the contrary, prevent convergence towards the optimal solution.

Unbalanced values of c_1 and c_2 can either cause particles to wander aimlessly or to cluster too quickly, reducing the chances of finding an optimal solution. In addition, too large a population can overload the calculations without necessarily improving performance.

Consequently, a gradual adjustment of the adaptive parameters can help to maintain better convergence and exploration in the PSO.

3.3 Advantages and Limitations of PSO

Advantages

The success of PSO is due to several factors. It is easy to implement and highly adaptable. Moreover, it is gradient-free, allowing it to be applied to various types of problems, including continuous, discrete, or combinatorial optimization. Its fast exploration helps overcome the limitations of traditional methods, such as overlapping issues or singular eigenvalues in stationary cycles. It also avoids the common pitfalls where gradient-based methods fail due to the presence of multiple local optima of different sizes.

Unlike deterministic approaches, PSO incorporates a random component, enabling it to explore the search space more widely and escape the influence of the environment. This enables it to explore the search space more widely and to escape local minima more effectively. This characteristic makes it particularly beneficial for problems where the objective function has multiple sub-optimal solutions.

In addition, PSO is less sensitive to initial conditions than other algorithms such as gradient descent. In gradient descent, a poor choice of starting point can lead to convergence towards a low-quality local minimum. In contrast, PSO takes advantage of the collective dynamics of its swarm to explore different regions of the search space, gradually identifying the most promising areas.

Limitations

PSO also has limitations that can affect its performance if not properly considered.
The efficiency of PSO strongly depends on the tuning of hyperparameters, mainly the inertia

factor w and the acceleration coefficients c_1 and c_2. Poor parameter settings will lead to suboptimal performance. A high inertia factor causes excessive dispersion of particles, delaying convergence, while a low inertia factor limits exploration, increasing the risk of trapping the algorithm in a local minimum.

Although PSO is effective in avoiding some local minima, it does not always guarantee finding the global optimum. When applied to complex problems with many local minima, the algorithm may converge to a suboptimal solution without fully exploring the search space.
To address this, PSO variants such as random perturbations or restricted neighborhood strategies introduce additional diversification mechanisms.

PSO is often less efficient than gradient-based methods for well-defined and differentiable functions. In such cases, methods like gradient descent with adaptive learning rates or Newton-Raphson algorithms generally converge faster to the optimal solution. Due to its stochastic nature and collective update process, PSO may require more iterations to achieve a solution of equivalent quality.

Finally, the cost of PSO can become a limiting factor when the particle population size is too large. A small population may restrict exploration, while a large population increases computation time without necessarily improving accuracy. Thus, balancing convergence quality and computational complexity is essential when applying PSO to real-world problems.

4 Implementation of PSO in SAS with PROC IML

Intuitively, the particle has inertia in its motion through the solution space and is drawn towards the best position previously visited by that particle and the best position visited by any particle (global best). Particles influence each other through the global best state, which attracts all particles and is updated as soon as any particle visits a position better than the previous global best. Figure 1 graphically presents one iteration of this process for a single particle.

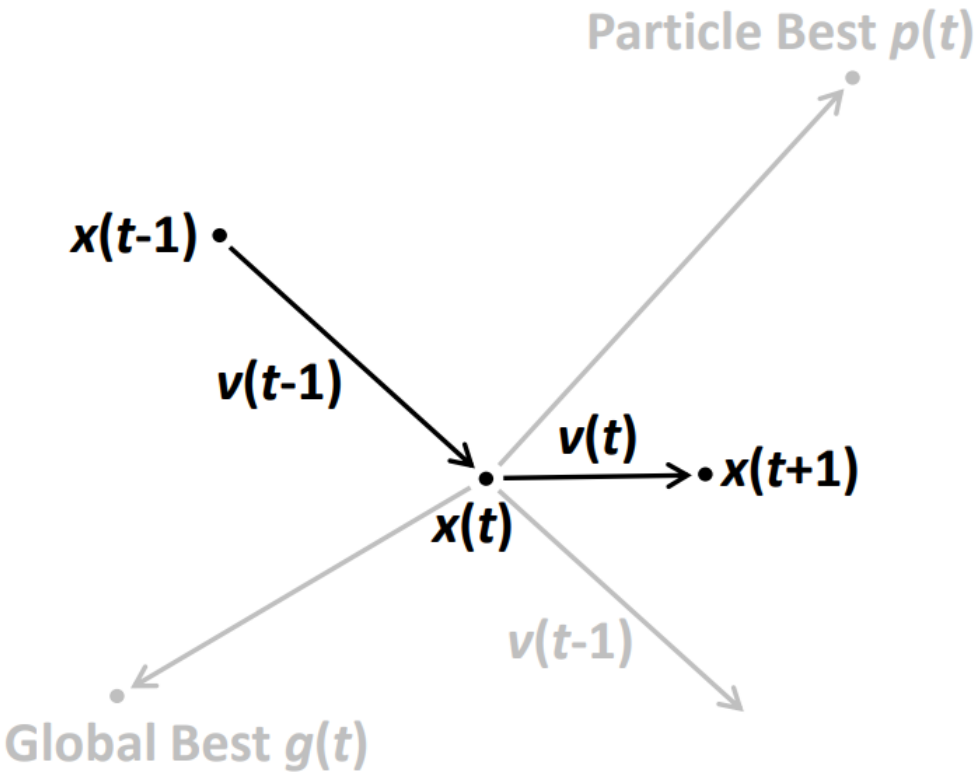

Figure 1: Updating Particle Position and Velocity

It is crucial to develop PSO algorithm by its implementation under PROC IML in SAS. Simply put, an illustration of this approach should start with a simple function to optimize. The chosen function is quadratic with a shift, making it an example of a convex function suitable for optimization tests. This approach allows to study the convergence and robustness of the algorithm before applying it to more complex problems. This function is defined as follows:

```
start f(X);
    decalage = 5;
    return ((X - decalage)[,##]);
finish;
```

This function allows studying the convergence of the algorithm towards an optimized solution.

Each particle in the swarm is initialized with a random position and velocity within the search space defined by the limits `rangeMin` and `rangeMax`.

```
start Initializeparticles(p, v, pB, cB, ffB, fPB, rangeMin, rangeMax, sizeP,
paramsP);
    p = j(sizep, paramsP, .);
    v = j(sizep, paramsP, .);
    do j = 1 to paramsP;
        p[,j] = rangeMin[j] + (rangeMax[j] - rangeMin[j]) * rand('Uniform', sizep);
        v[,j] = (rangeMin[j] - rangeMax[j]) * 0.1 + (rangeMax[j] - rangeMin[j]) * 0.2
        * rand('Uniform', sizeP);
    end;
    pB = p;
    fPB = j(sizep, 1, 1E308);
    cB = p[1,];
    ffB = f(cB);
finish;
```

This approach ensures a uniform distribution of initial particles and the search space to be covered efficiently.

At each iteration, the particles adjust their velocity and position based on their own best position and the best position found by the entire swarm. The update follows the classic PSO rule:

```
start particleMovement(p, v, pB, cB, inertia, selfBoost, groupBoost, rangeMin,
rangeMax, iter, nbIter);
    sizep = nrow(p);
    paramsP = ncol(p);
    do i = 1 to sizep;
        do j = 1 to paramsp;
            rp = rand('Uniform');
            rg = rand('Uniform');
            v[i,j] = inertia * v[i,j]
                    + selfBoost * rp * (pB[i,j] - p[i,j])
                    + groupBoost * rg * (cB[j] - p[i,j]);
            max_velocity = (rangeMax[j] - rangeMin[j]) * 0.1 * (1 - iter/nbIter);
            v[i,j] = sign(v[i,j]) * min(abs(v[i,j]), max_velocity);
            p[i,j] = p[i,j] + v[i,j];
        end;
    end;
finish;
```

The use of `inertia`, `selfBoost` and `groupBoost` makes it possible to control the balance between exploration and exploitation, thus avoiding premature convergence towards a local minimum.

Each particle stores the best solution encountered, and the swarm as a whole updates the best global solution at each iteration:

```
start UpdateBestpositions(p, pB, cB, ffB, fPB);
    fValues = f(p);
    update = loc(fValues < fPB);
    if ncol(update) > 0 then do;
```

```
        pB[update,] = p[update,];
        fPB[update] = fValues[update];
    end;
    minpBIndex = fPB[>:<];
    if fPB[minpBIndex] < ffB then do;
        ffB = fPB[minpBIndex];
        cB = pB[minpBIndex,];
    end;
finish;
```

This update ensures that the algorithm gradually converges towards a global optimum.

A stopping criterion is defined, which is based on a tolerance threshold and a maximum number of iterations without improvement in order to avoid unnecessary computations when the algorithm has converged:

```
 if noImprovementCount >= 1000 then do;
    print "Convergence atteinte après" iter "itérations";
    leave;
 end;
```

This mechanism speeds up the process by stopping the optimization as soon as the improvements become negligible.

At the end of the process, several indicators are displayed to evaluate the efficiency of the algorithm:

```
print "Global minimum trouvé:" ffB[format=best8.];
print "Solution optimale:" cB[format=8.4];
print "Itération finale:" iter;
print "Temps machine écoulé:" (t1 - t0) "secondes";
```

The display of execution time is a crucial element for evaluating the computational efficiency of the algorithm. In optimization, it is important to measure not only the quality of the obtained solution but also the time required to achieve it. This aspect becomes even more relevant when comparing PSO with other algorithms such as `PROC OPTMODEL` or gradient descent, where the speed of convergence can be a determining factor in choosing an algorithm for a specific application.

This implementation ensures that the PSO algorithm operates efficiently under `PROC IML` and can be compared with other optimization techniques to assess its performance in terms of accuracy and speed.

5 Models Comparison

This section is dedicated to the comparison of the results of optimization algorithms using two distinct functions:

A simple quadratic function: This function is convex and differentiable, which allows optimization algorithms to converge easily towards the global optimum. It serves as a reference

for observing the efficiency of different methods in an ideal setting where there are no multiple local minima.

The Schwefel function: Unlike the first one, this function is highly nonlinear and has numerous local minima. It presents a challenge for optimization algorithms, especially those, based on iterative updates like gradient descent.

The aim is to analyse the performance of three methods - Particle Swarm optimization (PSO), Gradient Descent (GD) and `PROC OPTMODEL` in these two contexts and to understand how they react to increasing complexity. The review highlights the details of some results obtained and explains the differences in the behavior of each algorithm faced to the particularities of the functions studied.

The test function used in the first subsection is a shifted quadratic function. It is defined as follows:

$$f(X) = \sum_{i=1}^{n} (X_i - 5)^2$$

where each X_i represents an optimization variable.

As this function is strictly convex, it guarantees the existence of a unique global minimum. This property is particularly advantageous for `PROC OPTMODEL` and gradient descent, which effectively exploit convexity to converge rapidly on the optimal solution. On the other hand, it does not highlight the advantages of PSO, whose strength lies more in the global search on non-convex and complex functions.

The function is continuous and differentiable over its entire domain. This mathematical regularity makes it particularly suitable for analytical optimization methods, especially those based on the gradient, which require first derivatives to guide the search for the optimum.

5.1 `PROC OPTMODEL` Results Analysis

La procédure OPTMODEL

Résumé de la solution	
Solveur	NLP
Algorithme	Interior Point Direct
Fonction objective	f
Statut de la solution	Optimal
Valeur de l'objectif	1.836559E-16
Erreur d'optimalité	9.0909662E-8
Infaisabilité	0
Itérations	5
Presolve Time	0.00
Durée de la solution	0.04

[1]	X
1	5
2	5
3	5
4	5
5	5

	f
Valeur minimale :	1.8366E-16

Figure 2: `PROC OPTMODEL` Procedure

`PROC OPTMODEL` well suited for simple, convex, and well-structured functions. In fact, this algorithm is based on mathematical programming techniques that allow it to calculate quickly, without requiring a prolonged exploration process. In this case, the result is obtained in just 5 iterations.

This result is made possible by the exact nature of the algorithm used (Interior point Direct), which allows near instantaneous convergence by exploiting the mathematical structure of the problem. Its execution time (0.04 seconds) is extremely low, making it an optimal choice for functions where a single global minimum can be unambiguously identified.

Analysis of the results obtained shows that `prox optmodel` provides a very accurate optimal solution with a numerical error of the order of 1.8366E-16, indicating that the algorithm achieves a near perfect convergence in this type of configuration.

5.2 Gradient Descent Results Analysis

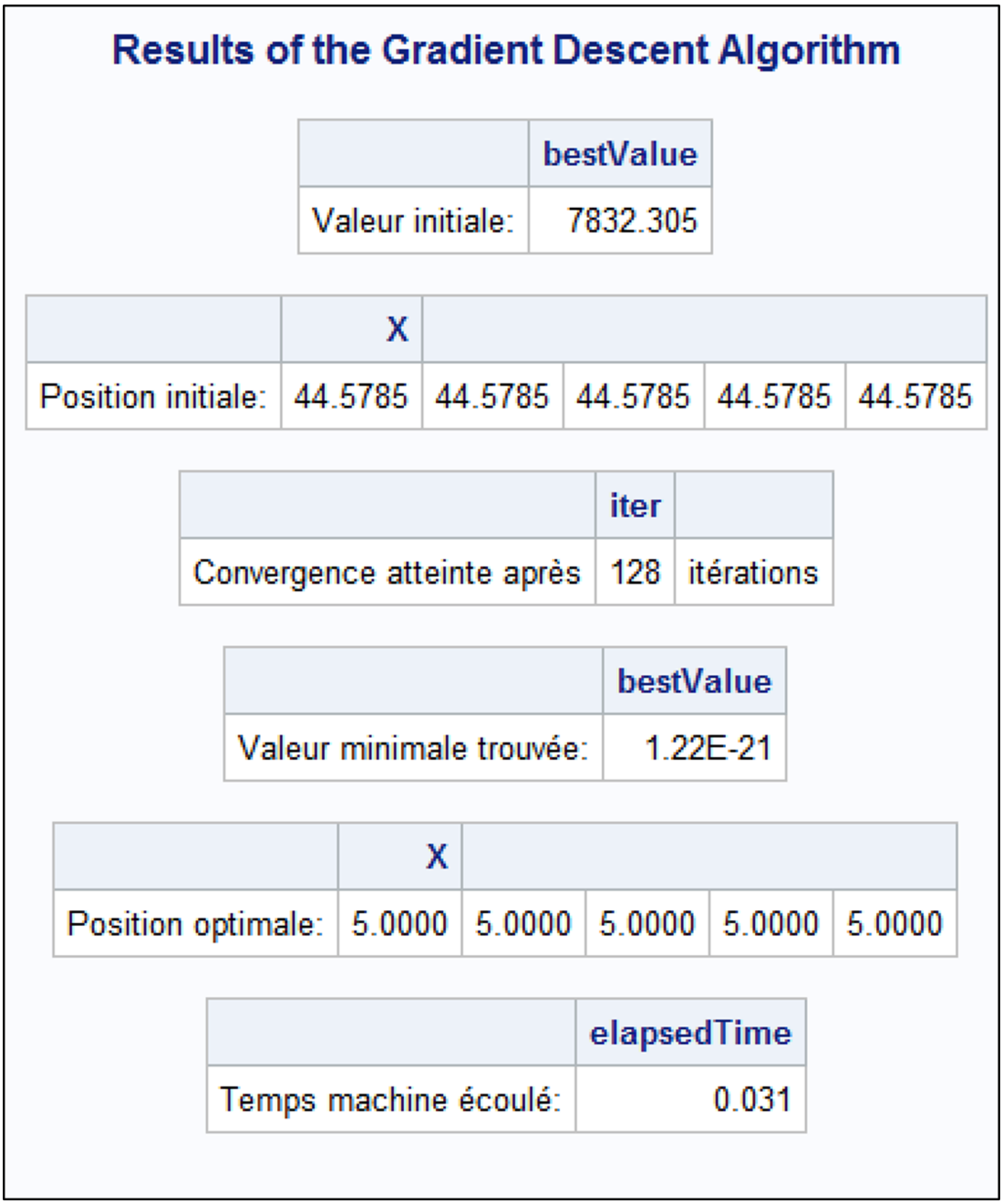

Figure 3: Gradient Descent Procedure

As with the `PROC OPTMODEL`, excellent results are obtained for this function.

In this test, the gradient descent reached the optimum in just 128 iterations, which is very fast. What is more, the execution time (0.031 seconds) is also very low. This speed is explained by the fact that the quadratic function used is convex and differentiable, allowing the algorithm to converge efficiently without unnecessary exploration.

Analysis of the results shows that the solution obtained is accurate with an error of the order of 1.22E-21, which is well below the typical numerical errors of numerical methods.

5.3 PSO Algorithm Results Analysis

Results of the PSO Algorithm

	iter
Itération finale	501

	ffB
Meilleur score obtenu	4.89E-25

Solution optimale trouvée

cB				
Paramètre 1	Paramètre 2	Paramètre 3	Paramètre 4	Paramètre 5
5.0000	5.0000	5.0000	5.0000	5.0000

Temps machine :	0.5939999	secondes

Figure 4: PSO Procedure

Unlike analytical approaches such as `PROC OPTMODEL`, PSO does not perform direct gradient calculations but relies on a heuristic approach that balances exploration and exploitation. This characteristic makes it particularly suitable for complex and non-convex optimization problems, but it may be less efficient for simple quadratic convex functions where more direct methods are available.

In this test, PSO required 501 iterations to converge, which is longer than `PROC OPTMODEL`, which finds the solution in just a few iterations due to its deterministic mathematical approach. However, it is comparable to gradient descent in terms of the number of iterations, though the latter remains more efficient for a simple quadratic function.

In terms of performance, PSO's execution time is 0.59 seconds, whereas `PROC OPTMODEL` converges almost instantly by leveraging the mathematical structure of the problem. Gradient descent also converges quickly, though slightly slower than `PROC OPTMODEL`.

However, the solution obtained by PSO is extremely precise, with a numerical error of 4.89E-25, confirming that the algorithm has reached optimal convergence. This highlights PSO's ability to provide highly accurate solutions, even though its heuristic approach makes it less effective for convex functions where `PROC OPTMODEL` and gradient descent are more appropriate.

5.4 Schwefel Function Presentation

The Schwefel function is a test function widely used in optimization to evaluate the robustness of algorithms faced with complex landscapes. It is defined by:

$$f(X) = 418.9829n - \sum_{i=1}^{n} X_i \sin(\sqrt{|X_i|})$$

where X_i represent the parameters to be optimized and n is the dimension of the problem.

This function has the following properties:

- It has many local minima, which complicates convergence for gradient-based methods.
- Its global minimum is located at X= 420.9687 in each dimension.
- It is non-convex and non-quadratic, which makes it particularly difficult to optimize with conventional algorithms.

The 3-dimensional representation is shown below:

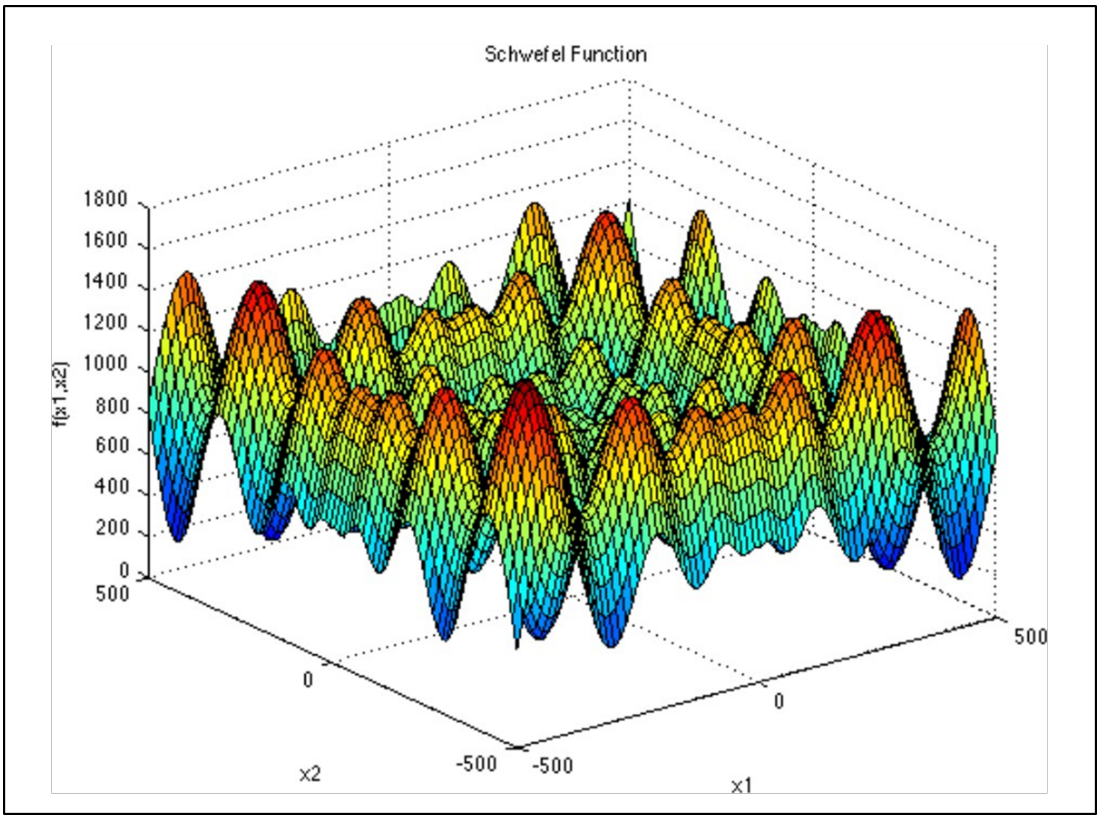

Figure 5: Schwefel Function 3D

This representation shows that the function has many local minima, which makes it very difficult to optimize.

To compare the results, we performed the calculations in 1-dimensional and 5-dimensional spaces.

5.4.1 Results in 1-Dimensional Space

`PROC OPTMODEL`

- **Number of iterations:** 5
- **Solution obtained:** 5.2392
- **objective function value:** 415.04
- **Execution time:** 0.01 seconds

Gradient Descent

- **Number of iterations:** 26 810
- **Solution obtained:** 421.0940
- **Best score:** 0.0019923
- **Execution time:** 0.625 seconds

PSO

- **Number of iterations:** 501
- **Solution obtained:** 420.9687
- **Best score:** 0.000013
- **Execution time:** 0.282 seconds

The evaluation of the performance of the three algorithms on a 1-dimensional space of the Schwefel function reveals contrasting behavior in terms of convergence and robustness to local minima.

`PROC OPTMODEL` demonstrates extremely fast convergence, reaching a solution in just 5 iterations. However, the solution obtained (5.2392) is far from the expected global optimum (420.97). This suggests that this algorithm, although effective on convex problems, is unsuitable for functions with several local minima. Indeed, it seems to be trapped in a local minimum, which considerably impairs its performance.

Gradient descent achieves a solution much closer to the global optimum, with a value of 421.0940. However, this result is obtained at the cost of a very high number of iterations (26 810). This behavior highlights the high sensitivity of this algorithm to local minima, requiring a large number of iterations to escape the pitfalls of the optimization landscape and converge on a satisfactory solution.

PSO stands out for its efficiency, reaching a solution very close to the global optimum in just 501 iterations and with a very fast execution time of 0.282 seconds. This performance is a testament to PSO's ability to navigate efficiently through a complex search space, combining exploration and exploitation to avoid local minima while maintaining good convergence speed.

5.4.2 Results in 5-Dimensional Space

`PROC OPTMODEL`

- **Number of iterations:** 5
- **Solution obtained:** 5.2392 (for each dimension)
- **Final value:** 2075.2
- **Final position:** [5.2392, 5.2392, 5.2392, 5.2392, 5.2392]
- **Execution time:** 0.01 seconds

Gradient Descent

- **Number of iterations:** 10.001
- **Solution obtained:** 770.59336
- **Final position:** [66.869854, 428.50273, 67.296009, 414.15486, 401.56179]
- **Execution time:** 0.189 seconds

PSO

- **Number of iterations:** 501
- **Solution obtained:** 420.9687 (for each dimension)
- **Final value:** 0.000064
- **Final position:** [420.9687, 420.9687, 420.9687, 420.9687, 420.9687]
- **Execution time:** 2.969 seconds

`PROC OPTMODEL` fails to find a good solution and gets stuck far from the best possible result. This shows that it is not suitable for complex functions with multiple traps (local minima).

Gradient Descent performs better than `PROC OPTMODEL`, but it gets stuck in a local minimum. after 10.001 iterations, it reaches 770.59336, which is still far from the expected global minimum (0). Its final position varies across dimensions, meaning that it did not successfully find the best solution.

PSO is once again the most effective method. It finds a solution much closer to the global minimum in only 501 iterations. Unlike the other methods, it avoids local traps and gets much closer to the best result. Even though its computation time is longer, it is much more reliable for solving this type of complex problem.

6 Conclusion and Perspectives

A comparative study of these three optimization algorithms on functions of increasing complexity represents the strengths and limitations of each approach depending on the nature of a problem studied.

On the one hand, `PROC OPTMODEL` proves to be a fast and efficient method. However, its performance worsens considerably in the presence of multiple local minima, as is the case with the Schwefel function. The algorithm is then unable to explore the search space efficiently and converges on a sub-optimal solution, demonstrating its limitations in a global optimization framework.

On the other hand, gradient descent appears as a more robust alternative. It is capable in approaching of the global optimum, but at the cost of a high number of iterations. Gradient descent is still very sensitive to local minima, which significantly lengthens its convergence time when applied to non-convex and highly non-linear functions. This characteristic makes it an acceptable choice for relatively simple optimizations, although potentially ineffective in solid optimization landscapes.

Finally, PSO has established itself as the best-performing method for highly complex functions. Unlike analytical approaches and gradient-based algorithms, PSO is able to navigate efficiently in a complex search space, avoiding local minima. This algorithm allows to reach solutions very close to the global optimum with a relatively fast execution time. although it takes longer to run than `PROC OPTMODEL` on convex functions, its efficiency on non-convex problems is clearly superior.

This paper reveals several important advantages of swarm-based algorithm. Firstly, PSO is particularly effective for non-convex functions with several local minima, where gradient-based approaches are comparatively trapped. It does not require the derivatives calculations, which allows it to be applicable to a wider range of problems, especially when the objective function is discontinuous or non-derivable. Eventually, this algorithm adapts well to changes to the response surface, due to this flexibility, it has the potential to solve a wide range of real-world optimization problems in many disciplines, especially in engineering and Machine Learning.

However, although PSO performs well in these contexts, it also has certain limitations. Its validity decreases when the constraints are too strong, as it does not involve as strict a treatment of constraints as some analytical methods do. At very high dimensionality, Particle Swarm Optimization can become computationally expensive to ensure efficient convergence. Finally, if particle diversity is not managed well, it may stagnate at the local optima, which requires additional mechanisms to prevent this.

In summary, this paper provided a detailed exposition of the PSO algorithm and its implementation in SAS. The results obtained show that the choice of an optimization algorithm depends largely on the nature of the function. `PROC OPTMODEL` excels on well-defined, convex problems, while gradient descent can be considered when the differentiability of the function is assured. Nevertheless, for complex, multi-modal problems, PSO appears as a relevant solution, due to its robustness and adaptability. This underlines the importance of a methodical choice of algorithms depending on the specifics of the optimization problem.

7 References

[1] Eberhart R., Kennedy J. Particle Swarm Optimization, Proceedings of the IEEE International Conference on Neural Networks, 1995, pp. 1942–1948. URL: https://ieeexplore.ieee.org/document/488968.

[2] Cleghorn C.W., Engelbrecht A. P. Particle Swarm Convergence: An empirical investigation, 2014, pp. 1-7. URL: http://www.cmap.polytechnique.fr/~nikolaus.hansen/proceedings/2014/WCCI/CEC-2014/PROGRAM/E-14451.pdf.

[3] Karpenko A. P., Seliverstov E. Ju. A review of particle swarm methods for the global optimization problem, 2009, pp. 2. (In Russian).

[4] Cazzaniga P., Nobile M. S., Besozzi D. The impact of particles initialization in PSO: parameter estimation as a case in point, 2015, vol. 94, pp. 1-8. URL: https://ieeexplore.ieee.org/document/7300288.

[5] Djellali H., Ghoualmi N. Improved chaotic initialization of particle swarm applied to feature selection, 2019, pp. 1-5. URL: https://ieeexplore.ieee.org/document/8807837.

[6] Ruder S. An overview of gradient descent optimization algorithms, 2017, pp. 1-3. URL: https://arxiv.org/abs/1609.04747.

[7] Karen E. Walker, Walker Consulting LLC Chandler arizona. Gradient Descent Using SAS™ for Gradient Boosting, 2020, pp. 3-5. URL: https://www.lexjansen.com/pharmasug/2020/AI/PharmaSUG-2020-AI-025.pdf.

[8] MQL5 Community: MetaTrader 5. Algorithmes d'optimization de la population: Essaim de particules, 2023. URL: https://www.mql5.com/fr/articles/11386 (Online forum reference).

[9] Le Tallec R. Techniques alternatives de modélisation de la probabilité de défaut pour une banque de détail, 2019, pp. 15-60. URL: https://www.institutdesactuaires.com/docs/mem/f08ad555ca5f0b533589e74cc1dd8067.pdf.

[10] Watson G., UCLA Department of Biostatistics, Los Angeles, CA. Particle Swarm Optimization in SAS, pp. 1-7. URL: https://www.lexjansen.com/wuss/2014/63_Final_Paper_PDF.pdf.

[11] Srivastava A., Kumbharvadiya S. Developing the Code: Executing Particle Swarm Optimization in SAS, 2014, pp. 1-4. URL: https://support.SAS.com/resources/papers/proceedings14/2030-2014.pdf.

[12] SAS Institute Inc. SAS/OR 13.2 User's Guide: Mathematical programming The OPTMODEL Procedure, 2014, pp. 30-170. URL: https://support.SAS.com/documentation/onlinedoc/or/132/optmodel.pdf.

[13] Surjanovic S., Bingham D. Virtual Library of Simulation Experiments: Schwefel Function, 2013. URL: https://www.sfu.ca/~ssurjano/schwef.html.

[14] Oliveira I. SAS/OR User's Series: Getting Started with SAS/OR OPTMODEL, 2013. URL : https://video.SAS.com/detail/video/2565868829001/SAS-or-user-s-series:-getting-started (SAS Video).

[15] Pratt R. PROC OPTMODEL : Solving Optimization Problems with Hybrid Approaches, 2018. URL : https://video.SAS.com/detail/video/5839970485001/proc-optmodel:-solving-optim (SAS Video).

[16] Hughes E. SAS Optimization on SAS Viya Using PROC OPTMODEL, 2018. URL : https://video.SAS.com/detail/video/5802740521001/SAS-optimization-on-SAS-viya-using-proc-optm (SAS Video).

b n