




# A Python-based Algorithm for Production and Inventory Optimization

Héctor Cañas<sup>1</sup> ; Yakdiel Rodríguez-Gallo<sup>2</sup> ; Manuel Cardona<sup>3</sup> 




<sup>1</sup>*Faculty of Engineering, Department of Industrial Engineering, Universidad Don Bosco, El Salvador, [hector.canas@udb.edu.sv](mailto:hector.canas@udb.edu.sv)*

<sup>2</sup>*Faculty of Engineering, Universidad Don Bosco, El Salvador, [yakdiel.rodriguez@udb.edu.sv](mailto:yakdiel.rodriguez@udb.edu.sv),*

**Abstract—** Optimization challenges in industrial engineering, particularly in economic order quantity (EOQ) and materials requirement planning (MRP), have traditionally been complex. This research addresses critical limitations in existing production and inventory management models by addressing recent computational advancements. We propose a comprehensive approach to resolving large-scale industrial engineering optimization problems by integrating high-level programming languages and advanced optimization tools. The study focuses on developing a generic Python-based optimization algorithm using a reference optimization model and Gurobi solver, with primary contributions including: (i) systematic exploration of optimization methods in industrial engineering; (ii) development of a flexible, scalable optimization approach; (iii) demonstration of computational techniques' potential in solving complex production planning challenges. By bridging theoretical optimization models with practical implementation, this research offers a cost-effective solution that extends beyond traditional limitations of economic order quantity and production lot sizing methodologies.

**Keywords—** *Mathematical programming, gurobi, optimization, inventory control, production planning*

# A Python-based Algorithm for Production and Inventory Optimization

Héctor Cañas<sup>1</sup> ; Yakdiel Rodríguez-Gallo<sup>2</sup> ; Manuel Cardona<sup>3</sup> 

<sup>1</sup>*Faculty of Engineering, Department of Industrial Engineering, Universidad Don Bosco, El Salvador, hector.canas@udb.edu.sv*

<sup>2</sup>*Faculty of Engineering, Universidad Don Bosco, El Salvador, yakdiel.rodriguez@udb.edu.sv,*

<sup>3</sup>*Vice-presidency for Science and Technology, Universidad Don Bosco, El Salvador, manuel.cardona@udb.edu.sv*

**Abstract**— Optimization challenges in industrial engineering, particularly in economic order quantity (EOQ) and materials requirement planning (MRP), have traditionally been complex. This research addresses critical limitations in existing production and inventory management models by addressing recent computational advancements. We propose a comprehensive approach to resolving large-scale industrial engineering optimization problems by integrating high-level programming languages and advanced optimization tools. The study focuses on developing a generic Python-based optimization algorithm using a reference optimization model and Gurobi solver, with primary contributions including: (i) systematic exploration of optimization methods in industrial engineering; (ii) development of a flexible, scalable optimization approach; (iii) demonstration of computational techniques' potential in solving complex production planning challenges. By bridging theoretical optimization models with practical implementation, this research offers a cost-effective solution that extends beyond traditional limitations of economic order quantity and production lot sizing methodologies.

**Keywords**— *Mathematical programming, gurobi, optimization, inventory control, production planning*

## I. INTRODUCTION

The rapid advancement of information technologies (IT) and computing capabilities has transformed problem-solving into industrial engineering, particularly in production and inventory management. The economic order quantity (EOQ) model, pioneered by Harris Ford, remains a critical framework for optimizing production and inventory control [1].

Existing EOQ models face significant limitations, including simplified assumptions about production and demand [2], inability to address complex, dynamic manufacturing environments, and high implementation costs for small and medium enterprises. Successive researchers have expanded the original EOQ concept, with Taft introducing the economic production lot (EPL) model and Wagner-Within developing a dynamic lot size model addressing time-varying demand [3], [4]. These extensions significantly influenced materials requirement planning (MRP) development [2], [5].

MRP systems are the center of focus in our study, and their origins and development are crucial to providing a comprehensive understanding of this research. MRP has been formulated using optimization modeling methods and widely studied using different approaches. While MRP has been

adopted by enterprise resource planning (ERP) systems, this research does not focus on identifying modern software solutions for production planning and control systems. Readers interested in Industry 4.0 related production planning can refer to [6] for a conceptual proposal.

Mathematical programming continues to evolve, demanding robust methodological frameworks that can serve as foundational tool for advanced computational research. This research introduces a novel algorithmic approach that provides a test bed for future complexity development, offering researchers a systematic baseline for exploring more sophisticated optimization strategies.

The proposed algorithm is a generic optimization model [7] integrating modern computational tools with industrial engineering principles, and demonstrates the potential of a Python-based open-source software package such as Pyomo [8].

The study offers a flexible, scalable approach to production and inventory management by addressing the gap between theoretical optimization models and practical implementation. The primary research objectives include: (i) developing a comprehensive, adaptable optimization model; (ii) providing a cost-effective solution for production and inventory control; (iii) showcasing the potential of computational techniques in industrial optimization.

The paper is structured to progressively explore these concepts: Section 2 provides an overview of optimization methods and the Pyomo framework. Section 3 presents the Python-based production and inventory control algorithm. Section 4 presents the results of the optimization model. Finally, Sections 5 and 6 discuss limitations, conclusions, and future research lines.

## II. MATERIALS AND METHODS

### A. Mathematical optimization

Engineering optimization problems can be comprehensively classified based on their variable characteristics and mathematical structures. Biegler and Grossmann's seminal classification distinguishes continuous and discrete optimization approaches with subcategories and distinct computational challenges [9].

Continuous optimization encompasses linear programming (LP) and non-linear programming (NLP), subdivided into sophisticated problem types. Linear complementary problems

(LCP) and quadratic programming (QP) represent LP variants, while semidefinite programming (SP) addresses more complex NLP scenarios. NLPs problems introduce additional complexity through potential convex or nonconvex configurations, which can yield locally optimal solutions depending on their mathematical properties.

Discrete optimization presents alternative modeling strategies through mixed integer programming (MILP) and mixed integer nonlinear programming (MINLP). When all variables are integers, the approach transforms into pure integer programming (IP), offering powerful modeling capabilities for complex engineering challenges. These methodologies effectively address real-world problems like resource allocation, routing optimization, and strategic decision-making.

The mathematical formulation for optimization problems, regardless of continuous or discrete nature, follows a generalized framework that allows systematic problem representation and computational solution strategies. This versatile approach enables engineers to model intricate systems, balance competing constraints, and develop optimal solutions across diverse industrial and technological domains [9].

By providing a structured approach to optimization, researchers and engineers can systematically analyze, model, and resolve complex computational challenges, bridging theoretical mathematical concepts with practical engineering applications. A general formulation for a MIP can be given as follows:

$$\min(Z) = f(x, y) \text{ s.t. } \begin{cases} h(x, y) = 0 \\ g(x, y) \leq 0 \\ x \in X, y \in \{0,1\} \end{cases} \quad (1)$$

Where  $f(x, y)$  is the objective function,  $h(x, y) = 0$  is, the equation describing the performance of the system, and  $g(x, y) \leq 0$  are the equations describing the constraints of the modeled system. The  $x$  variables, are continuous and generally refer to the state variables, while the  $y$  variables are the discrete variables, generally these take values between 0 and 1.

An essential extension in these MIP models is that they can be converted into dynamic models, where a time variable makes them discrete-time models, and continuous-time models give rise to optimal control problems. Another essential variable is the inclusion of uncertainty; these models give rise to stochastic optimization problems. On the other hand, for an optimization model with continuous variables (i.e., NLPs), we have the following formulation:

$$\min(Z) = f(x) \text{ s.t. } \begin{cases} h(x) = 0 \\ g(x) \leq 0 \end{cases} \quad (2)$$

These problems have the characteristic of being convex and non-convex. Where convex regions require  $g(x)$  to be convex and  $h(x)$  to be linear. If an NLP is of the type convex, then any local solution is a global solution to the NLP problem. On the other hand, if the objective function is strictly convex, it has a unique solution. The Karush Kuhn Tucker conditions can

only satisfy local optimality for non-convex problems, and more rigorous search methods are needed to find optimal global solutions.

If a problem is linear, then it can be solved by the standard algorithm, the simplex method [10]. In this sense, optimization problems have different solving techniques. For example, some solvers rely on algorithm construction to solve NLPs, but this is not within the scope of this article.

#### B. Python-based optimization modelling (Pyomo)

Python is a high-level object-oriented programming (OOP) language that researchers and developers have widely adopted for creating open-source libraries and contributing to the Python ecosystem. In the domain of mathematical optimization, several powerful frameworks have emerged, including SciPy [11], GEKKO [12], and Pyomo [8].

While each library has its merits, Pyomo has distinguished itself through its comprehensive documentation and extensive support for high-level modeling constructs, including differential equations and logical disjunctions. GEKKO offers a robust framework for optimization modeling and machine learning, supporting various operational modes such as parameter regression, data reconciliation, real-time optimization, dynamic simulation, and non-linear predictive control. These Python libraries comprehensively support optimization formulations, including LP, QP, NLP, MILP, and MINLP.

Developing optimization frameworks in Python has seen significant evolution over the past decade. Watson et al. introduced PySP, a framework specifically designed for modeling and solving stochastic programs, featuring innovative approaches to model specification, extensive form generation, and scenario-based decomposition [13]. This work has influenced subsequent research, as demonstrated by Fan et al., who leveraged PySP alongside Gurobi and Pyomo to address flexible supply chain planning under stochastic disruptions [14].

Significant advances in differential equation optimization came with Nicholson et al.'s introduction of `pyomo.dae`, a framework integrated into Pyomo offers unprecedented flexibility in handling differential and algebraic equations without restricting users to predefined forms [15]. In parallel, Andersson et al. developed `CaSADi`, an open-source framework specializing in nonlinear optimization and optimal control problems, which, while implemented in C++, provides efficient interfaces for Python, MATLAB, and Octave users [16].

Recent years have seen increasingly sophisticated applications of these tools. Soraya Rawlings et al. demonstrated the practical application of various optimization formulations (NLP, MINLP, and GDP) in chemical engineering, specifically for optimizing Kaibel column design [17]. Jusevičius et al. conducted a comprehensive comparison of algebraic modeling languages, finding that while commercial solutions like AMPL and GAMS offer particular advantages in formulation and performance, open-source frameworks such as Pyomo and

JuMP demonstrate comparable capabilities [18], [19], [20], [21].

The field continues to evolve with specialized extensions and applications. Wiebe and Misener extended Pyomo's capabilities with RModel, facilitating robust optimization problem-solving [22]. Kneeven et al. advanced the field of parallel computing in optimization with mpi-sppy, an open-source library for parallel stochastic program solving [23]. Table I provides a comparison of the optimization modelling frameworks.

TABLE I  
OPTIMIZATION MODELLING FRAMEWORKS

Framework	Features	Problem Type	Language
GEKKO	Machine learning and optimization for dynamic systems	LP, QP, NLP, MIP, MILP	Python
Pyomo	Intuitive syntax for modelling optimization problems in Python	LP, NLP, MIP	Python
SciPy	Part of a scientific Python ecosystem with various optimization algorithms	Unconstrained/constrained minimization, global optimization, among others.	Python
PySP	Stochastic programming extension for Pyomo	Stochastic optimization	Python
Pyomo.dae	Extension for differential algebraic equations in Pyomo	Optimization with differential and algebraic equations	Python
CaSADi	Symbolic framework with automatic differentiation	Nonlinear optimization and algorithmic differentiation	Python, MATLAB, C++
AMPL	Intuitive and expressive syntax, well documented	LP, QP, NLP, MIP, MILP, among others.	AMPL
GAMS	Extensive Integrated Development Environment for the industry	LP, QP, NLP, MIP, MILP, among others	GAMS
JuMP	Performance benefits, cutting edge packages	LP, QP, NLP, MIP, MILP, among others	Julia
RModel	Extends Pyomo for robust optimization	Robust optimization	Python
Mpi-sppy	Scalable framework for stochastic programming	Stochastic optimization	Python

### III. GENERIC PYTHON-BASED ALGORITHM FOR PRODUCTION AND INVENTORY OPTIMIZATION

#### A. Problem definition

In this section, a generic production and inventory optimization model is taken as the basis for the Python algorithm, specifically the work proposed by McDonald and Karimi. In this model, it is assumed that (i) all parameters are deterministic; (ii) it is a dynamic model with discrete time periods; (iii) multiple products belong to three kinds of sets: raw materials, intermediate products, finished products; (iv) intermediate products are used as raw materials for a second processor, this gives rise to internal material flow; (v) the production system is semi-continuous such that operations are continuous and by batch processing; (vi) a production ratio is given, the continuous production runs with starts and stops that must be made between products families (setup). In these production systems, processing times must be longer since a production campaign must be the longest possible to reduce the frequency of product families' changes. Fig. 1 shows a graphical representation of the production and inventory optimization model as a network [7].

The generic optimization model aims to determine the optimal balancing of inventory and transition costs due to the multi-product processed in parallel semi-continuous operations.

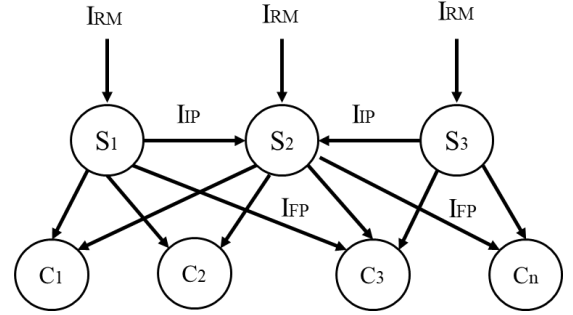


Fig. 1 Inventory optimization model as a network. Adapted from [7].

#### B. Mathematical formulation

The model's mathematical notation is given in Table II. It contains the parameters and variables and descriptions of the mathematical notations.

TABLE II  
MODEL'S MATHEMATICAL NOTATION

Symbol	Definition
$F$	Set for the product families $f \in F$
$J$	Set for machine processors $j \in J$
$S$	Set for the production plants $s \in S$
$C$	Set for the market zones $c \in C$
$T$	Set for the periods $t \in T$
$I$	Set for the products $I \in \{RM, IP, FP\}$
$P_{ijst}$	Quantity to produce of products $i$ in machine $j$ in plant $s$ at time $t$
$RL_{ijst}$	Corresponding production run-length for product $i$
$RL_{fjst}$	Corresponding production run-length for product family $f$
$C_{ist}$	Raw material consumption or intermediate products for product $i$ in plant $s$ during period $t$
$I_{ist}$	Inventory levels for product $i$ held at plant $s$ in period $t$
$S_{isct}$	Supply of finished products for product $i$

$\sigma_{iss't}$	The material flow of intermediate products for product $i$
$I_{ict}^-$	Quantity of shortage of finished product $i$
$I_{ist}^\Delta$	Quantity of deviation below desired safety stock levels for product $i$
$Y_{fst}$	Binary variable, 1 if product family $f$ is produced in plant $s$ in period $t$ , 0, otherwise
$h_{ist}$	Inventory holding cost of product $i$
$\mu_{ic}$	Benefits per unit sold of product $i$
$P_{is}$	Raw material price for product $i$
$\xi_{is}$	Penalty cost for below desired safety stock levels for product $i$ in plant $s$
$v_{ijs}$	The variable cost for production of a unit of product $i$ in machine processor $j$ in plant $s$
$t_{ss'}/t_{sc}$	Unitary shipping costs for products $i$
$fc_{fjs}$	Fixed cost for transitions between family products $f$
$MRL_{fst}$	Minimum run-length for product families $f$
$R_{ijst}$	The effective rate for product $i$
$\beta_{i'is}$	Bill of materials (BOM) where $i'$ th quantity is needed to produce product $i$
$H_{jst}$	Quantity of available time for production in machine $f$ in plant $s$ in period $t$
$d_{ict}$	Demand of finished product $i$ for client $c$ in period $t$
$I_{ist}^L$	Desired safety stock levels for product $i$
$I_{iso}$	Initial inventory levels at the beginning of the planning horizon

The model's mathematical formulation is given as follows:

$$\begin{aligned}
& \min \sum_i \sum_j \sum_s \sum_t v_{ijs} P_{ijst} \\
& + \sum_i \sum_s \sum_t P_{is} C_{ist} \\
& + \sum_i \sum_s \sum_t h_{ist} I_{ist} \\
& + \sum_i \sum_s \sum_c \sum_t t_{sc} S_{isct} \\
& + \sum_i \sum_s \sum_{s'} \sum_t t_{ss'} \sigma_{iss't} \\
& + \sum_i \sum_s \sum_t \xi_{is} I_{ist}^\Delta \\
& + \sum_i \sum_c \sum_t \mu_{ic} I_{ict}^-
\end{aligned} \quad (3)$$

$$P_{ijst} = R_{ijst} RL_{ijst} \quad \forall i, j, s, t \in I, J, S, T \quad (4)$$

$$RL_{fst} - H_{jst} Y_{fst} \leq 0 \quad \forall f, j, s, t \in F, J, S, T \quad (5)$$

$$RL_{fst} - MRL_{fst} Y_{fst} \leq 0 \quad \forall f, j, s, t \in F, J, S, T \quad (6)$$

$$RL_{fst} = \sum_{i \in \Lambda_{if}} RL_{ijst} \quad \forall j, s, t \in J, S, T \quad (7)$$

$$\sum_f RL_{fst} \leq H_{jst} \quad \forall j, s, t \in J, S, T \quad (8)$$

$$C_{ist} = \sum_{i' \ni \beta_{i'is} \neq 0} \beta_{i'is} \sum_j P_{i'jst} \quad \forall i, j, s, t \in I, J, S, T \quad (9)$$

$$C_{ist} = \sum_{s'} \sigma_{iss't} \quad \forall i, s, t \in I, S, T \quad (10)$$

$$I_{ist} = I_{is(t-1)} + \sum_j P_{ijst} - \sum_{s'} \sigma_{iss't} - \sum_c S_{isct} \quad \forall i, s, t \in I, S, T \quad (11)$$

$$I_{ict}^- \geq I_{ic(t-1)}^- + d_{ict} - \sum_s S_{isct} \quad \forall i, c, t \in I, C, T \quad (12)$$

$$\sum_{s, t' \leq t} S_{isct'} \leq \sum_{t' \leq t} d_{ict'} \quad \forall i, c \in I, C \quad (13)$$

$$I_{ist}^\Delta \geq I_{ist}^L - I_{ist} \quad \forall i, s, t \in I, S, T \quad (14)$$

$$P_{ijst} \leq R_{ijst} H_{jst} \quad \forall i, s, t \in I, S, T \quad (15)$$

$$S_{isct} \leq \sum_{t' \leq t} d_{ict'} \quad \forall i, s, c \in I, S, C \quad (16)$$

$$I_{ict}^- \leq \sum_{t' \leq t}^{T'} d_{ict'} \quad \forall i, c \in I, C \quad (17)$$

$$I_{ist}^\Delta \leq I_{ist} \quad \forall i, s, t \in I, S, T \quad (18)$$

$$P_{ijst}, RL_{fst}, RL_{ijst}, C_{ist}, \quad \forall i, c, f, j, s, s', t \in I, C, F, J, S, S', T \quad (19)$$

$$I_{ist}, I_{ict}^-, \sigma_{iss't}, I_{ist0}^\Delta \geq 0$$

The model's cost minimization objective function is given by (3). This equation includes production costs, raw materials provisioning costs, inventory holding costs, internal demand costs, inventory penalty costs for inventory deviation, and shortage costs.

The production amount corresponding to production runs through a ratio given by (4). For this specific problem, longer production times lead to high inventory and safety stock levels in a semi-continuous production system. At the same time, high-capacity utilization is achieved, and low transition costs are incurred.

An upper limit to production campaigns (run-lengths) is given by (5).

The family production campaigns have a required lower limit  $MRL_{fst}$  is given by (6).

A parameter  $\Lambda_{if}$  defines the relationship between products belonging to each product family; this is given by (7).

A capacity constraint is defined so the total production time cannot be violated, as given by (8).

Consumption of raw materials or intermediate products using BOM. For raw materials, the quantity that should be purchased from an external supplier is given by (9). In the model, it is assumed that these materials are available on demand. In the case of intermediate products consumed in the plants, supply must come from the production at the same site, or this supply must come from another supplier  $s'$ .

All material shipped to a plant must be consumed at the exact location and in the same period, as given by (10). This implies that inventory is held only where the product is manufactured. This avoids redundancy in the material flow network.

The inventory balance equation establishes that inventory held at the end of a period  $t$  is equal to the inventory at the end of a previous period plus the production during that period minus the quantity of product that must be shipped to a customer. This is given by (11).

The drop in customers is the cumulative difference between demand and supply. Supply drops move from one period to the next, as given by (12). Shortages will be zero when supply meets demand.

Supply in the current period to satisfy orders from the previous period, subject to the upper limit of the total accumulated demand up to that period, is given by (13).

If the inventory levels exceed the desired safety stock level, the safety stock shortage takes the value of zero due to the positive  $I_{ist}^\Delta$ , this is given by (14). Otherwise, it is equal to the deviation from the desired level, whose maximum level is  $I_{ist}^L$ .

Lower and upper constraints are given by (15-19).

### C. Python-based optimization algorithm using Pyomo

Pyomo provides a robust framework for implementing mathematical optimization models in Python. This section details the implementation process, from model initialization to solution generation.

The implementation uses a concrete model instance suitable for working with known datasets. While abstract models are also possible in Pyomo, concrete models offer direct data integration capabilities.

The model leverages multiple data management approaches:

- Primary data source: Microsoft Excel sheets, integrated via Pandas.
- Data bridge: Pandas DataFrame to Pyomo-compatible dictionary conversion.
- Additional supported formats: CSV, JSON, and database connections via SQL.

The model's foundation relies on carefully defined sets, with special attention to hierarchical relationships. It is noteworthy that Python sets were used to define sets  $F, J, S, C, T$  and  $I$ . The most intricate set implementation involves mapping products to their respective product families and establishing crucial dependencies for the optimization process. To this end, a Python dictionary  $i\_of\_f = \{ 'F1': \{1,2,3\}, 'F2': \{4, 5\}, \dots \}$  defines the product families and the dependency of each product to each family is defined as parent products and child products (intermediate products).

Parameters are implemented as indexed components, populated from the preprocessed data. The conversion process follows this workflow: (i) load raw data via Pandas; (ii) transform DataFrames into dictionaries using `to_dict()`; (iii) initialize Pyomo parameters with the processed data.

Constraints are implemented using a functional programming approach. Each constraint follows this structure (See Fig. 2).

```
def prod_amount(model, *indices):
    # constraint logic
    return expression # type: ignore

model.constraint_name = Constraint(
    index_set, rule=prod_amount # type: ignore
)
```

Fig. 2 Functional programming approach. Source: Authors.

The objective function implementation uses Pyomo's `Objective()` method, incorporating optimization direction via the `sense` parameter and mathematical expression through the `expr` parameter, and complex objective functions can be defined using nested expressions.

The model employs Gurobi as the primary solver, leveraging its state-of-the-art algorithms for efficient solution

generation [24]. The solver integration is streamlined through Pyomo's unified solver interface. Fig 3 shows the flowchart of the Pyomo algorithm. Fig 4 shows graphically the programming logic of the algorithm for every set, parameter, variable and objective function using Pyomo.

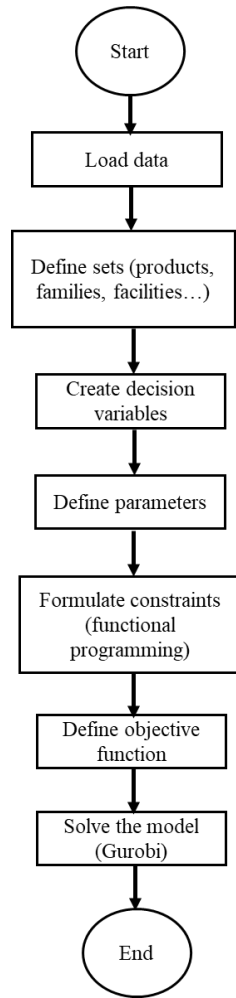


Fig. 3 Flowchart of the Python-based algorithm implementation in Pyomo. Source: Authors

#### IV. RESULTS

The optimization model examines two chemical production facilities with distinct operational characteristics. The primary facility, Plant  $s_1$ , manages 23 products (labeled  $I_1$  through  $I_{23}$ ) organized into 11 product families ( $F_1$  through  $F_{11}$ ). This plant operates with a single machine configuration, setting the set  $J$  to 1. The secondary facility, plant  $s_2$ , functions as a dependent operation, producing 11 products (labeled  $I_{24}$  through  $I_{34}$ ) and requiring one unit from each product family manufactured at the primary facility.

The model operates across a twelve-month planning horizon with deterministic demand patterns. A notable characteristic of the system is that plant  $s_2$  operates without capacity constraints, with its demand structured to be precisely 50 percent of the supply available from plant  $s_1$ . The dataset

encompasses comprehensive information regarding production rates, minimum run lengths, and fixed charge costs, with detailed specifications available in McDonald and Karimi's work.

The implementation generated a substantial mathematical model of 8,544 continuous variables and 264 integer variables. The optimization process achieved an objective value of 13,281.5, maintaining a precise optimization gap of 0.01 percent.

The computational environment utilized a 12th Generation Intel® Core™ i5-12400 processor, featuring six physical cores and 12 logical processors, capable of simultaneously utilizing up to 12 threads. The solution process demonstrated remarkable efficiency, requiring 0.01 seconds for data reading, 0.02 seconds for solution resolution, and a total execution time of 2.99 seconds.

The analysis focused on two critical output components: the optimal production plan for family product campaigns at Plant  $s_1$  and the production run lengths for family 1 at Plant  $s_2$ . This selective presentation of results facilitates direct comparison with the findings reported in McDonald and Karimi's original work.

The implementation leveraged the Gurobi Optimizer V11.0.1 [24] in conjunction with a Python-based Pyomo pipeline, establishing a robust framework for solving this complex optimization challenge. The results demonstrate the model's effectiveness in handling large-scale production planning scenarios while maintaining computational efficiency.

#### V. DISCUSSION

Production planning optimization centers on the fundamental challenge of determining optimal order quantities. The analysis reveals a critical trade-off in manufacturing systems: while larger lot sizes increase inventory carrying costs, they simultaneously reduce the frequency of production runs or orders. This relationship forms the foundation of economic order quantity calculations and influences the production planning framework.

The implementation builds upon McDonald and Karimi's work, which was chosen for its comprehensive approach to mixed-integer linear programming (MILP) in production environments. The model's complexity stems from its handling of both internal and external demand patterns within a semi-continuous manufacturing system. This system encompasses several sophisticated elements:

The production environment incorporates machinery operating in semi-continuous modes, managing final and intermediate products. The system maintains inventory controls through multiple mechanisms: safety stock requirements, managed demand deferrals, product structure complexity through bills of materials (BOM), product family relationships, and machine-dependent effective production ratios. Including effective production, ratios represent a notable aspect, as this parameter rarely appears in conventional production systems due to its dependence on specific machinery capabilities.

The implementation leverages contemporary optimization tools and specialized solvers, demonstrating significant



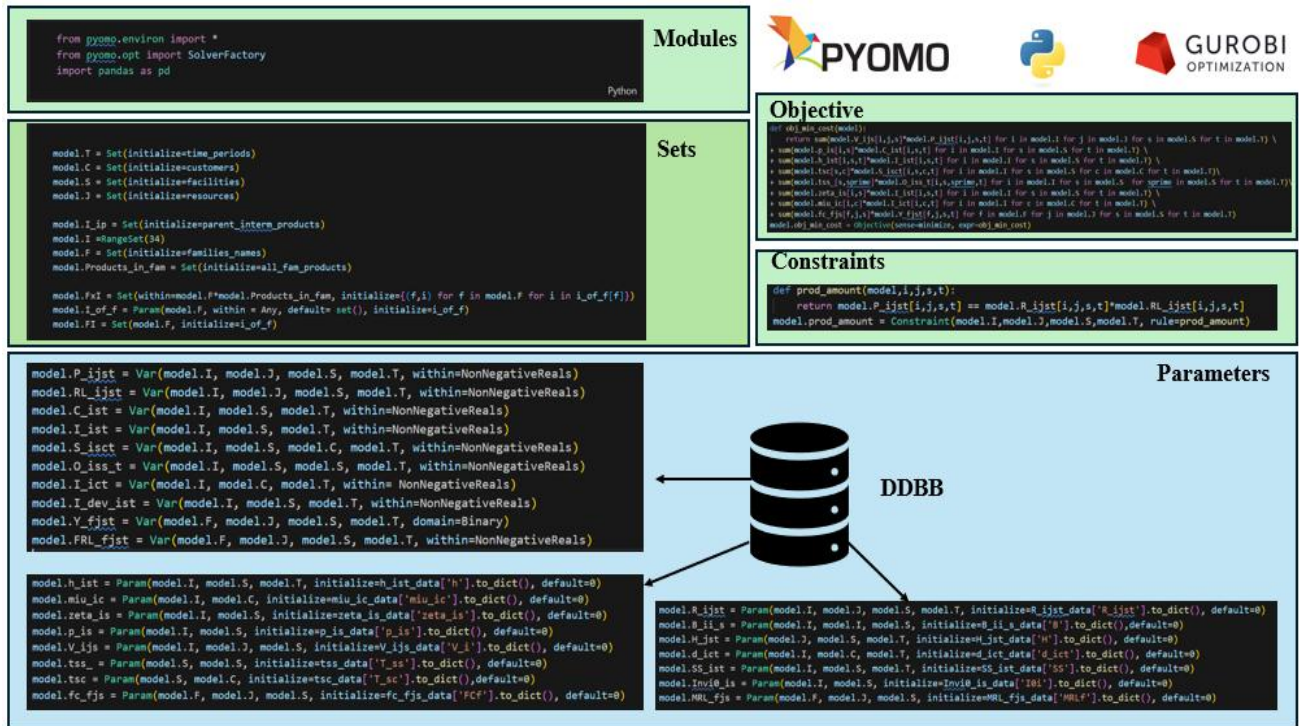


Fig. 4 Python-based algorithm implementation in Pyomo. Source: Authors.

advancement in handling large-scale industrial problems. The computational results are particularly noteworthy, achieving approximately three seconds solution times. This performance represents a substantial improvement over the original work by McDonald and Karimi, which employed a relaxed mixed-integer programming approach and achieved a gap of 1.6 percent.

This research bridges an essential gap in management science literature. While production planning optimization is extensively studied, few works comprehensively connect theoretical frameworks with modern computational tools and implementation strategies. Applying current optimization technologies to this classical problem demonstrates the potential for improving industrial planning processes through advanced computational methods.

The results validate the effectiveness of modern optimization approaches in addressing complex production planning challenges, suggesting promising directions for future research and practical applications in industrial settings.

This MRP model implementation faces important limitations when deployed across diverse industrial settings. Data quality is a primary concern as the model requires consistent and accurate data inputs to produce reliable results. This condition is not always met in production settings with varying data gathering practices. Computational scalability is another consideration since it may become problematic in large-scale applications involving thousands of products and complex interdependencies, leading to exponentially increasing solution times. Data integration presents additional hurdles when implementing the system alongside existing ERP

architectures and legacy systems that dominate many manufacturing systems. Organization barriers further could complication adoption, including staff training requirements, staff resistance to mathematical optimization approaches, and the need for an alignment across inventory, production and procurement departments.

## VI. CONCLUSIONS

This research addresses two fundamental aspects of industrial optimization.

The first component provides a comprehensive examination of engineering optimization methodologies. The second delves into practical applications by analyzing optimal lot size determination using an advanced optimization model.

The core implementation utilizes a sophisticated MRP system founded on mathematical programming principles. This model generates optimal production and ordering schedules across defined planning horizons while incorporating multiple complex variables (i.e., product family relationships and their transitions, safety inventory management, deferred demand handling, and inter-product dependencies).

The implementation's innovative aspect lies in using Pyomo, which represents one of Python's most robust and well-documented optimization libraries. This choice of technology creates a scalable foundation for future MRP system expansions and enhancements.

The implementation demonstrates exceptional performance across several critical metrics (i.e., solution computation efficiency, result accuracy and reliability, and system interoperability with complementary tools).



Building on these promising results, future research will focus on expanding the model's capabilities to address production scheduling and machine sequencing challenges, specifically extending McDonald and Karimi's work through Python implementation. This extension aims to create a more comprehensive production optimization framework.

The research establishes a solid foundation for advancing industrial optimization practices while providing practical tools for immediate application in production environments.

#### ACKNOWLEDGMENT

The authors would like to thank Universidad Don Bosco for supporting this research.

#### REFERENCES

- [1] F. Whitman Harris, "How Many Parts to Make at Once," *Factory, The Magazine Management*, vol. 10, no. 152, pp. 135–136, 1913.
- [2] W. J. Hopp and M. L. Spearman, *Factory physics*. Waveland Press, 2011.
- [3] E. Taft, "Formulas for exact and approximate evaluation—handling cost of jigs and interest charges of product manufactured included," *The Iron Age*, vol. 101, no. 5, pp. 1410–1412, 1918.
- [4] H. M. Wagner and T. M. Whitin, "Dynamic version of the economic lot size model," *Management science*, vol. 5, no. 1, pp. 89–96, 1958.
- [5] J. A. Orlicky, *Material requirements planning: the new way of life in production and inventory management*. McGraw-Hill, Inc., 1974.
- [6] H. Cañas, J. Mula, F. Campuzano-Bolarín, and R. Poler, "A conceptual framework for smart production planning and control in Industry 4.0," *Computers & Industrial Engineering*, vol. 173, p. 108659, 2022.
- [7] C. M. McDonald and I. A. Karimi, "Planning and scheduling of parallel semicontinuous processes. 1. Production planning," *Industrial & Engineering Chemistry Research*, vol. 36, no. 7, pp. 2691–2700, 1997.
- [8] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in Python," *Mathematical Programming Computation*, vol. 3, pp. 219–260, 2011.
- [9] L. T. Biegler and I. E. Grossmann, "Retrospective on optimization," *Computers & Chemical Engineering*, vol. 28, no. 8, pp. 1169–1192, 2004.
- [10] G. B. Dantzig, "Linear programming and extensions," in *Linear programming and extensions*, Princeton university press, 2016.
- [11] P. Virtanen *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.
- [12] L. D. Beal, D. C. Hill, R. A. Martin, and J. D. Hedengren, "Gekko optimization suite," *Processes*, vol. 6, no. 8, p. 106, 2018.
- [13] J.-P. Watson, D. L. Woodruff, and W. E. Hart, "PySP: modeling and solving stochastic programs in Python," *Math. Prog. Comp.*, vol. 4, no. 2, pp. 109–149, Jun. 2012, doi: 10.1007/s12532-012-0036-1.
- [14] Y. Fan, F. Schwartz, S. Voß, and D. L. Woodruff, "Stochastic programming for flexible global supply chain planning," *Flex Serv Manuf. J.*, vol. 29, no. 3–4, pp. 601–633, Dec. 2017, doi: 10.1007/s10696-016-9261-7.
- [15] B. Nicholson, J. D. Sirola, J.-P. Watson, V. M. Zavala, and L. T. Biegler, "pyomo.dae: a modeling and automatic discretization framework for optimization with differential and algebraic equations," *Math. Prog. Comp.*, vol. 10, no. 2, pp. 187–223, Jun. 2018, doi: 10.1007/s12532-017-0127-0.
- [16] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Math. Prog. Comp.*, vol. 11, no. 1, pp. 1–36, Mar. 2019, doi: 10.1007/s12532-018-0139-4.
- [17] E. Soraya Rawlings, Q. Chen, I. E. Grossmann, and J. A. Caballero, "Kaibel column: Modeling, optimization, and conceptual design of multi-product dividing wall columns," *Computers & Chemical Engineering*, vol. 125, pp. 31–39, Jun. 2019, doi: 10.1016/j.compchemeng.2019.03.006.
- [18] V. Jusevičius, R. Oberdieck, and R. Paulavičius, "Experimental Analysis of Algebraic Modelling Languages for Mathematical Optimization," *Informatica*, pp. 283–304, 2021, doi: 10.15388/21-INFOR447.
- [19] R. Fourer, D. M. Gay, and B. W. Kernighan, "AMPL. A modeling language for mathematical programming," 2003.
- [20] B. A. McCarl *et al.*, "McCarl GAMS user guide," *GAMS Development Corporation*, 2014.
- [21] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A modeling language for mathematical optimization," *SIAM review*, vol. 59, no. 2, pp. 295–320, 2017.
- [22] J. Wiebe and R. Misener, "ROmodel: modeling robust optimization problems in Pyomo," *Optim Eng.*, vol. 23, no. 4, pp. 1873–1894, Dec. 2022, doi: 10.1007/s11081-021-09703-2.
- [23] B. Kneeven, D. Mildebrath, C. Muir, J. D. Sirola, J.-P. Watson, and D. L. Woodruff, "A parallel hub-and-spoke system for large-scale scenario-based optimization under uncertainty," *Math. Prog. Comp.*, vol. 15, no. 4, pp. 591–619, Dec. 2023, doi: 10.1007/s12532-023-00247-3.
- [24] L. Gurobi Optimization, "Gurobi optimizer reference manual (2020)," 2023.