Journal of
open research software

PyGuLP: A Python Package for Weighted Goal Linear Programming

SOFTWARE
METAPAPERS

KSHITIJ VAIDYA
ROOMANI SRIVASTAVA
ISHAN PANDIT
KSHITIJ JADHAV
*Author affiliations can be found in the back matter of this article

ubiquity press

ABSTRACT

Resource allocation is central to day-to-day activities, planning, and policymaking, often involving multiple targets rather than a single objective. In healthcare and public health, planners may need to address competing targets alongside feasibility constraints such as fixed budgets and resource limits. Traditional linear programming frameworks do not directly represent aspiration targets and deviations from them. Goal linear programming (GLP) uses linear programming constructs to model such targets explicitly and minimize deviations from them. **PyGuLP** is a lightweight Python package that implements Weighted GLP using standard linear programming constructs, making goal-based optimization accessible while preserving control over the underlying optimization model.

CORRESPONDING AUTHOR:
Kshitij Jadhav
AIDE Lab, Koita Centre for Digital Health, Indian Institute of Technology Bombay, India
kshitij.jadhav@iitb.ac.in

KEYWORDS:
Optimization; weighted goal programming; PuLP; Python; operations research; decision support; resource allocation

TO CITE THIS ARTICLE:
Vaidya K, Srivastava R, Pandit I, Jadhav K 2026 PyGuLP: A Python Package for Weighted Goal Linear Programming. *Journal of Open Research Software*, 14: 64. DOI: https://doi.org/10.5334/jors.716

(1) OVERVIEW

INTRODUCTION

Linear programming provides efficient tools for expressing feasibility constraints and optimizing a single scalar objective. When analysts face problems involving multiple objectives, common practice is to rely on multi-objective linear programming techniques such as weighted sum scalarization, ϵ-constraint formulations, Pareto-based analyses, and interval-based multi-objective linear programming. In weighted sum approaches, multiple objectives are combined into a single objective function using weights defined by the user (1), where the relation between objective weights and the achieved outcomes is indirect and problem-dependent, making it difficult to interpret how a given choice of weights translates into target attainment. ϵ-constraint methods designate one objective for optimization while converting others into constraints with prescribed bounds; this can lead to infeasibility when targets cannot be simultaneously satisfied (2). Interval-based multi-objective linear programming represents objectives or parameters as ranges rather than fixed values (3), allowing uncertainty in objectives or parameters to be represented explicitly within the optimization formulation. Pareto-based approaches aim to enumerate or approximate a set of non-dominated solutions; a non-dominated solution is one for which no objective can be improved without worsening at least one other objective. This places the burden on the decision maker to explore and select a "best compromise" solution from a potentially large solution set (4).

These approaches provide different ways of representing and resolving multiple objectives. Goal linear programming (GLP) is particularly suited to problems in which decision makers can specify aspiration targets and wish to quantify deviations from those targets. GLP introduces deviation variables that explicitly quantify under- and over-achievement relative to specified targets, allowing infeasible aspiration levels to be accommodated in a controlled and interpretable manner. This explicit representation of deviations aligns more naturally with decision-making contexts, where targets serve as guiding benchmarks rather than strict constraints (5, 6).

Although the theoretical foundations of goal programming are well established in the operations research literature (5–7), practical implementations are often reported as part of individual case studies rather than as reusable, general-purpose software. Goal programming has been used across multiple domains that require dynamic target handling. Examples include diet planning, manpower planning, and resource allocation in hospitals (8). Beyond health, financial problems such as deciding insurance premiums (9, 10) or environmental planning for reduced emissions limits coupled with conservation goals (11, 12) have used GLP to balance such goals.

Dedicated implementations of goal programming do exist; for example, the R package `goalp` supports weighted and lexicographic linear goal programming (13). Analysts may also implement goal programming models using spreadsheet-based workflows (14), proprietary modeling environments, or problem-specific code written for a single study. Commercial optimization environments can similarly be used to formulate GLP models; for example, FICO-Xpress (15) has been used to apply GLP in diet optimization and test different achievement functions (16). However, GLP model construction in general-purpose modeling environments requires the modeler to explicitly construct the goal equations and associated deviation variables. Such implementations may therefore remain problem-specific and require repeated model-construction code across applications.

The Python optimization ecosystem includes established modeling frameworks such as PuLP, Pyomo, CVXPY, and PICOS, as well as frameworks focused on multi-objective optimization such as `pymoo` (17–20). These tools provide broad optimization capabilities, but goal-programming models generally need to be constructed using their underlying variables, constraints, and objective-function interfaces. For example, Pyomo allows multiple objectives to be declared, but only one objective is active during a solve (21); implementing weighted goal programming (WGP) therefore requires the modeler to explicitly construct the achievement function, deviation variables, and associated goal-linking equations. PyGuLP instead represents goals directly as modeling objects and automates these goal-programming-specific components, including goal-level deviation reporting.

General-purpose optimization libraries such as PuLP provide flexible interfaces for defining decision variables, constraints, and objective functions and serve as front-ends to several linear programming solvers (22). PuLP itself does not provide goal-programming-specific abstractions such as goal definitions, deviation variables, or automated goal-linking equations. As a result, analysts implementing GLP using PuLP directly must manually introduce deviation variables and construct the corresponding goal-linking constraints within the linear program, requiring these components to be specified explicitly for each goal and repeated across models.

PyGuLP addresses this gap by providing a standardized, reusable software workflow for weighted GLP. Built on top of the PuLP optimization ecosystem, PyGuLP introduces structured abstractions for defining

goals, automatically generating deviation variables and goal-linking equations, and constructing the corresponding weighted achievement functions.

IMPLEMENTATION AND ARCHITECTURE

Design philosophy and architecture

PyGuLP is designed to formalize and standardize how WGP models are constructed and solved in applied research. The package provides a consistent structure for separating feasibility constraints from aspirational goals or targets, reflecting the distinction emphasized in the goal programming literature but often implemented manually in practice.

Figure 1 illustrates this conceptual framework and design philosophy of PyGuLP. Feasibility constraints define the admissible region and are treated as non-negotiable, while goals represent desired aspiration levels that may be violated at a quantifiable cost. Goals can be assigned weights that determine the penalties applied to their deviations, which are driven by what is important for the research question. For each goal, PyGuLP systematically introduces deviation variables and constructs the corresponding goal-linking constraints, ensuring that under- and over-achievement are represented explicitly. The resulting optimization problem remains a linear program and is solved using established solvers through the PuLP ecosystem. By returning achieved values and deviations in a structured form, PyGuLP supports interpretation, comparison across scenarios, and reuse across problem settings.

Mathematical formulation

PyGuLP implements the standard WGP formulation by transforming each user-defined goal into a pair of non-negative deviation variables that measure under- and over-achievement of the desired target. Let $x \in \mathbb{R}^n$ denote the vector of decision variables satisfying a set of hard linear constraints. For each goal $k = 1, \ldots, K$, let $Z_k(x)$ denote a linear goal expression with desired target value T_k.

To measure deviations from the target, PyGuLP introduces two deviation variables: an under-achievement variable d_k^- and an over-achievement variable d_k^+. These variables are linked to the original goal through the following goal equation:

$$Z_k(x) + d_k^- - d_k^+ = T_k, \qquad k = 1, \ldots, K$$

Non-negativity of deviation variables

The deviation variables satisfy

$$d_k^- \geq 0, \qquad d_k^+ \geq 0, \qquad k = 1, \ldots, K.$$

These variables quantify only the magnitude of deviation from the target. Negative deviations are therefore not

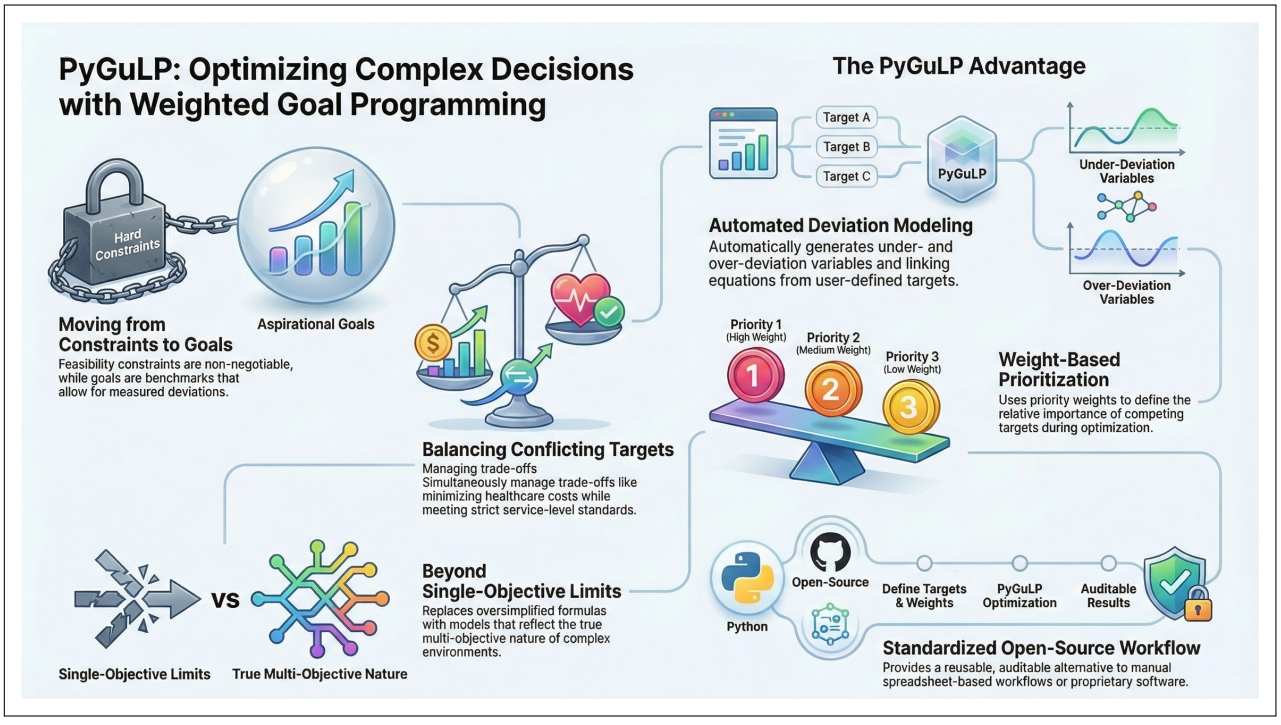


Figure 1 Conceptual overview of Goal Linear Programming: PyGuLP distinguishes between hard feasibility constraints, which define the admissible solution space, and aspirational goals, which represent target outcomes that may be achieved with allowable deviations. The package automatically constructs under- and over-deviation variables and applies weighted penalties to deviations from competing targets, enabling transparent multi-objective optimization and reproducible decision support within the Python ecosystem.

meaningful, making the formulation a standard linear programming model.

Weighted objective function

The optimization objective minimizes the weighted sum of deviations across all goals,

$$\min \sum_{k=1}^{K} \left(w_k^- d_k^- + w_k^+ d_k^+ \right),$$

where

$$w_k^- \geq 0, \qquad w_k^+ \geq 0,$$

represent the penalties associated with under-achievement and over-achievement, respectively.

PyGuLP supports both *symmetric* and *asymmetric* deviation penalties. When distinct penalty values are supplied, the optimization objective naturally captures situations in which under-achievement and over-achievement have different practical consequences.

If no separate penalty pair is specified, PyGuLP adopts the standard symmetric WGP formulation by assigning

$$w_k^- = w_k^+ = w_k,$$

which reduces the objective to

$$\min \sum_{k=1}^{K} w_k \left(d_k^- + d_k^+ \right).$$

This provides a simple default formulation while allowing users to specify independent penalties whenever asymmetric preferences are required. For example, a healthcare application may assign a larger penalty to under-achieving a minimum staffing level than to over-achievement, whereas inventory management problems may penalize excess stock more heavily than stock shortages.

Within PyGuLP, asymmetric penalties can be supplied through the optional `goal_weights` argument of the `solve_weighted()` method, which accepts an ordered pair (w_k^-, w_k^+) for each goal. For example,

```
# Symmetric penalties (default)
Goal(name="Protein", ..., weight=10)

# Asymmetric penalties
model.solve_weighted(
    goal_weights={
        "Protein": (20, 5)
    }
)
```

In the first case, both under-achievement and over-achievement incur the same penalty ($w_k^- = w_k^+ = 10$). In the second case, under-achievement is penalized four times more heavily than over-achievement ($w_k^- = 20$, $w_k^+ = 5$), allowing the optimization model to capture application-specific preferences.

An additional property of the formulation concerns the paired deviation variables. Both deviation variables are constrained to be non-negative and enter the objective with non-negative coefficients. For any goal with at least one positive deviation penalty,

$$w_k^- + w_k^+ > 0,$$

an optimal solution cannot have both deviation variables strictly positive simultaneously.

Suppose that

$$d_k^- = a, \qquad d_k^+ = b, \qquad a, b > 0.$$

Let

$$\delta = \min(a, b) > 0.$$

Define

$$\hat{d}_k^- = a - \delta, \qquad \hat{d}_k^+ = b - \delta.$$

Substituting into the goal equation gives

$$Z_k(x) + \hat{d}_k^- - \hat{d}_k^+ = Z_k(x) + (a - \delta) - (b - \delta) = Z_k(x) + a - b = T_k,$$

so feasibility is preserved.

The corresponding contribution to the weighted objective becomes

$$w_k^- \hat{d}_k^- + w_k^+ \hat{d}_k^+ = w_k^- a + w_k^+ b - \delta \left(w_k^- + w_k^+ \right).$$

Since $\delta > 0$ and $w_k^- + w_k^+ > 0$, this value is strictly smaller than the original weighted deviation contribution.

Therefore, at optimality,

$$d_k^- d_k^+ = 0,$$

for any goal with at least one positive deviation penalty. Thus, each such goal is represented as either under-achieved, over-achieved, or exactly attained, but not simultaneously under- and over-achieved.

Interpretation of goal weights

The weights determine the relative importance assigned to deviations across different goals. When all goals are measured in comparable units, larger weights correspond to higher priority.

If goals are measured on substantially different scales (for example, monetary cost versus waiting time), a common practice is to normalize each deviation by an appropriate scaling constant S_k, giving

$$\min \sum_{k=1}^{K} w_k \left(\frac{d_k^-}{S_k} + \frac{d_k^+}{S_k} \right).$$

The normalization constants account for differences in measurement units, whereas the weights represent the

relative importance assigned to each goal. Consequently, normalization and priority weighting serve distinct purposes: normalization places heterogeneous goals on a comparable scale, while the weights reflect the decision-maker's preferences.

The appropriate normalization is inherently problem- and data-dependent and is therefore left to the user rather than being prescribed by PyGuLP. Depending on the application, users may preprocess or scale their input data using existing data-processing tools, or normalize the goal expressions directly when constructing the PyGuLP model. For example, if one goal measures monetary cost and another measures patient waiting time, each may first be expressed relative to a suitable reference value or scaling constant before preference weights are assigned.

Note: Relationship between hard constraints and goals

A fundamental distinction in goal programming is the separation between hard constraints and goals. Hard constraints define the feasible solution space and must always be satisfied, exactly as in conventional linear programming. Typical examples include physical capacity limits, legal regulations, or resource availability constraints. Goals, on the other hand, represent desired target values that decision-makers would like to achieve whenever possible. Unlike hard constraints, goals are permitted to be under- or over-achieved through explicitly modeled deviation variables, whose magnitudes are minimized according to the preferences specified by the decision-maker. Consequently, goal programming should be viewed as a methodology for balancing multiple competing objectives while preserving the feasibility defined by mandatory constraints, rather than as a mechanism for relaxing or replacing those constraints.

QUALITY CONTROL

Quality control for PyGuLP has been carried out at multiple levels, including validation of core model components, functional validation through constructed worked examples, and verification of the package distribution and installation workflow.

Automated tests are provided for core classes responsible for model construction, including goal registration, generation of goal-linking equations, handling of deviation variables, and execution of the solver interface. These tests verify that goals and feasibility constraints are translated into consistent linear programming formulations and that expected solver outputs are returned.

Functional validation was performed using multiple constructed datasets representing decision problems inspired by previously published studies in healthcare operations and insurance premium-setting. These datasets were designed to exercise different modeling patterns relevant to weighted GLP, including multiple simultaneous service-level targets, trade-offs between profitability, coverage, and affordability in insurance premium-setting, and resource allocation problems involving competing performance goals.

The package distribution and installation workflow was validated using the TestPyPI environment prior to public release, ensuring correct packaging, dependency resolution, and installation through standard `pip`-based workflows in a clean Python environment before publication on PyPI.

Users can verify that the software is functioning correctly by running the example scripts provided with the repository, which construct WGP models and return solver status, optimized decision-variable values, and per-goal deviation measures. Reference outputs for the worked examples are provided in the `examples/README.md` file, allowing users to compare solver status, objective values, and key decision outputs against benchmark results.

Example execution

A worked example included with the repository demonstrates the application of PyGuLP to an outpatient clinic resource allocation problem. In this example, staffing levels across clinical units (triage, consultation, and infusion) and operational configuration options are chosen to balance multiple service objectives under budget and capacity constraints. The model includes several goals defined for different patient classes, including targets for turnaround time (TAT) and service coverage. These goals are incorporated into the optimization model using goal-linking equations with associated deviation variables, allowing the solver to minimize weighted deviations from the desired targets.

Using PyGuLP, goals are defined directly through the model interface by specifying the expression representing the achieved level of the goal, the target value, and the associated weight. A simplified excerpt from the outpatient example is shown below. Here, `TAT[c]` represents the previously defined TAT variable for patient class `c`.

```
for c in C:
    m.add_goal(
        Goal(
            name=f"TAT_target_{c}",
            expression=TAT[c],
            target=tat_target[c],
            weight=priority[c]
        )
    )
    goal_weights[f"TAT_target_{c}"] = (0.0, priority[c])
```

For these TAT goals, only deviation above the target is penalized. After defining the feasibility constraints and registering the goals, the WGP problem is solved using:

```
result = m.solve_weighted(
    goal_weights=goal_weights,
    cost_expr=total_cost_expr,
    cost_weight=lambda_cost
)
```

Execution returns the solver status, optimized decision-variable values, and goal-specific deviation values. For the outpatient example, outputs include staffing allocations across units, selected operational configurations, achieved TATs for each patient class, and the magnitude of deviations from the specified targets. These results allow users to verify that the software correctly constructs the goal-linking equations and resolves trade-offs between competing objectives.

(2) AVAILABILITY

OPERATING SYSTEM

PyGuLP is implemented in Python and is operating system independent. It can be used on operating systems supported by Python and PuLP, including Linux, macOS, and Windows.

PROGRAMMING LANGUAGE

Python

ADDITIONAL SYSTEM REQUIREMENTS

No additional system requirements beyond those required to run Python and the underlying linear programming solver used through PuLP are specified. PuLP can use the open-source COIN-OR CBC solver, while larger or industrial-scale problems may benefit from alternative solvers such as Gurobi, CPLEX, or GLPK configured through PuLP. Memory and processor requirements depend on the size of the optimization problem being solved.

DEPENDENCIES

Required:

- PuLP ≥ 2.7.0

Optional:

- pandas ≥ 1.5.0
- matplotlib

LIST OF CONTRIBUTORS

Kshitij Vaidya - Core classes development, testing and validation, documentation preparation Roomani Srivastava - Defined problem statement, conceptualized package development for GLP, curated datasets for testing and validation, package maintenance and submissions Ishan Pandit - Testing and validation of core classes, development of additional helper classes Kshitij Jadhav - Conceptualized and ideated package development, overall supervisor of the project, support and resources, reviewing the package, documentation and scholarly contributions from the package.

SOFTWARE LOCATION

Archive

Name: Zenodo
Persistent identifier: https://doi.org/10.5281/zenodo.18832569
Licence: MIT License
Publisher: AIDE Lab, Koita Centre for Digital Health, Indian Institute of Technology Bombay
Version published: 0.1.3
Date published: 29/08/2026

Code repository

Name: GitHub
Persistent identifier: https://github.com/aidelab-iitbombay/PyGuLP
Licence: MIT License
Date published: 29/08/2026

Emulation environment

Name: Not applicable
Persistent identifier: Not applicable
Licence: Not applicable
Date published: Not applicable

LANGUAGE

English

(3) REUSE POTENTIAL

PyGuLP supports reproducibility and alignment with best practices in optimization-based decision support by providing a standardized workflow for constructing weighted GLP models in which feasibility constraints are separated from aspirational targets. The package is intended for researchers and analysts who need to represent multiple targets explicitly, quantify under- and over-achievement relative to those targets, and explore trade-offs through weighted penalties on goal deviations.

The software is reusable across domains where decision problems involve balancing competing targets under resource or feasibility constraints. Typical use cases include healthcare operations planning, public health resource allocation, financial and insurance portfolio design, infrastructure planning, environmental management, and policy-oriented optimization problems. In these settings, targets often represent desired performance levels rather than strict feasibility requirements, making goal programming a natural modeling framework.

The constructed example datasets included with the software demonstrate reusable modeling patterns

derived from published studies. These include service-level planning problems involving multiple waiting-time targets in outpatient settings (23), premium-setting problems balancing profitability, coverage, and affordability considerations in insurance portfolios (10), and outpatient resource allocation problems involving trade-offs between TAT, operational cost, and service access (24, 25). These examples are provided to illustrate how users can adapt the same modeling structures to new domains by replacing coefficients, target values, and constraints while retaining the underlying goal-programming formulation.

PyGuLP is designed to allow extension as additional variants of GLP are incorporated over time. The current implementation focuses on WGP, while future development may include other commonly used formulations such as lexicographic (pre-emptive) goal programming, goal-attainment or compromise formulations, and variants supporting range-based or discrete aspiration levels. Future development will explore how these formulations can be incorporated while retaining, where possible, the existing model structure and goal specifications. Additional optional utilities such as streamlining common goal normalization techniques can also be explored.

The package structure also provides a basis for extending PyGuLP with additional goal-programming formulations and domain-specific modeling components. Longer-term development may include structured user interfaces through which domain experts can specify goals, constraints, and targets without directly constructing the underlying optimization model.

Contributions that extend supported goal programming approaches or provide domain-specific modeling layers are welcomed through the public repository. Guidance for reporting bugs, requesting features, and proposing code contributions is provided in the repository's contribution and support documentation.

FUNDING STATEMENT

The project has no funding to declare and was supported at institutional level.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Kshitij Vaidya orcid.org/0009-0004-4102-2094
AIDE Lab, Koita Centre for Digital Health, Indian Institute of Technology Bombay, India

Roomani Srivastava orcid.org/0000-0001-8183-109X
AIDE Lab, Koita Centre for Digital Health, Indian Institute of Technology Bombay, India

Ishan Pandit orcid.org/0009-0003-4558-0790
AIDE Lab, Koita Centre for Digital Health, Indian Institute of Technology Bombay, India

Kshitij Jadhav orcid.org/0000-0001-9795-8335
AIDE Lab, Koita Centre for Digital Health, Indian Institute of Technology Bombay, India

REFERENCES

1. **Halffmann P, Schäfer LE, Dächert K, Klamroth K, Ruzika S.** Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey. *Journal of Multi-Criteria Decision Analysis*. 2022;29(5–6): 341–363. DOI: https://doi.org/10.1002/mcda.1780
2. **Tantawy S, Sallam R.** Multiple objective linear programming (MOLP) problems with the same objective space. *Journal of Algorithms & Computational Technology*. 2009;3(4):573–581. DOI: https://doi.org/10.1260/174830109789621347
3. **Rivaz S, Saeidi Z.** Solving multiobjective linear programming problems with interval parameters. *Fuzzy Information and Engineering*. 2021 Nov;13(4):497–504. DOI: https://doi.org/10.1080/16168658.2021.2002544
4. **Halme M, Korhonen P.** Nondominated tradeoffs and termination in interactive multiple objective linear programming. In: *Improving Decision Making in Organisations: Proceedings of the Eighth International Conference on Multiple Criteria Decision Making Held at Manchester Business School, University of Manchester, UK, August 21st–26th, 1988*. Springer; 1989. pp. 410–423. DOI: https://doi.org/10.1007/978-3-642-49298-3_38
5. **Charnes A, Cooper WW.** Management models and industrial applications of linear programming. *Management Science*. 1957;4(1):38–91. DOI: https://doi.org/10.1287/mnsc.4.1.38
6. **Orumie UC, Ebong D.** A glorious literature on linear goal programming algorithms. *American Journal of Operations Research*. 2014;4:59–71. DOI: https://doi.org/10.4236/ajor.2014.42007
7. **Nagarajan S, Ravi J, Akila S.** Multiple objective LPP and goal programming: A review. *International Journal of Trend in Scientific Research and Development*. 2022;6(2):344–353.
8. **Kwak NK, Lee C.** A Linear Goal Programming Model for Human Resource Allocation in a Health-Care Organization. *Journal of Medical Systems*. 1997 Jun;21(3):129–140. DOI: https://doi.org/10.1023/A:1022850505219
9. **Gleason JM, Lilly CC.** A goal programming model for insurance agency management. *Decision Sciences*. 1977;8(1):180–190. DOI: https://doi.org/10.1111/j.1540-5915.1977.tb01076.x
10. **González-Fernández AI, Rubio-Misas M, Ruiz F.** Goal programming to evaluate the profile of the most profitable insurers: An application to the Spanish insurance industry.

International Transactions in Operational Research. 2020;27(6):2976–3006. DOI: https://doi.org/10.1111/itor.12787

11. **Mamun A, Shawwash Z, Abdalla A, Li J, Siu T.** Application of a goal programming algorithm to incorporate environmental requirements in a multi-objective Columbia River Treaty Reservoir optimization model. *Canadian Water Resources Journal/Revue canadienne des ressources hydriques*. 2015;40(1):111–125. DOI: https://doi.org/10.1080/07011784.2014.985514
12. **Abella G, Ramos E, Tubay J, Fiegalan F.** Land use allocation based on multi-goal linear programming model: A case study of Central Luzon State University. In: *IOP Conference Series: Earth and Environmental Science*. vol. 1220. IOP Publishing; 2023. p. 012039. DOI: https://doi.org/10.1088/1755-1315/1220/1/012039
13. **Palma D.** goalp: Weighted and Lexicographic Goal Programming Interface; 2022. R package version 0.3.1. DOI: https://doi.org/10.32614/CRAN.package.goalp
14. **Goh C.** Optimizing accounting decision making using goal programming. *Journal of Corporate Accounting & Finance*. 2019;30(1):161–168. DOI: https://doi.org/10.1002/jcaf.22369
15. **Fair Isaac Corporation (FICO).** FICO Xpress Optimization Help: Overview; 2025. Documentation built on 2025-11-21; covers FICO Platform 2025-04 and Xpress Solver 9.8 (and related components). Online documentation. https://www.fico.com/fico-xpress-optimization/docs/latest/overview.html.
16. **Gerdessen J, De Vries J.** Diet models with linear goal programming: Impact of achievement functions. *European Journal of Clinical Nutrition*. 2015;69(11):1272–1278. DOI: https://doi.org/10.1038/ejcn.2015.56
17. **Hart WE, Watson JP, Woodruff DL.** Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation*. 2011;3(3):219–260. DOI: https://doi.org/10.1007/s12532-011-0026-8
18. **Diamond S, Boyd S.** CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research*. 2016;17(83):1–5.
19. **Sagnol G, Stahlberg M.** PICOS: A Python interface to conic optimization solvers. *Journal of Open Source Software*. 2022;7(70):3915. DOI: https://doi.org/10.21105/joss.03915
20. **Blank J, Deb K.** pymoo: Multi-Objective Optimization in Python. *IEEE Access*. 2020;8:89497–89509. DOI: https://doi.org/10.1109/ACCESS.2020.2990567
21. **Pyomo Development Team.** Manipulating Pyomo Models: Activating and Deactivating Objectives. Accessed 9 August 2026. Pyomo Documentation.
22. **Mitchell S, O'Sullivan M, Dunning I.** PuLP: A Linear Programming Toolkit for Python. Department of Engineering Science, University of Auckland; 2011.
23. **Kumar K, Pandey D.** Setting up waiting time targets for out-patients using fuzzy linear programming. *Indian Journal of Science and Technology*. 2023;28(16):2133–2143. DOI: https://doi.org/10.17485/IJST/v16i28.1124
24. **Fan X, Tang J, Yan C, Guo H, Cao Z.** Outpatient appointment scheduling problem considering patient selection behavior: Data modeling and simulation optimization. *Journal of Combinatorial Optimization*. 2021 Nov;42(4):677–699. DOI: https://doi.org/10.1007/s10878-019-00487-x
25. **Wang X.** Emergency Department Staffing: A Separated Continuous Linear Programming Approach. *Mathematical Problems in Engineering*. 2013;2013(1):680152. DOI: https://doi.org/10.1155/2013/680152

TO CITE THIS ARTICLE:
Vaidya K, Srivastava R, Pandit I, Jadhav K 2026 PyGuLP: A Python Package for Weighted Goal Linear Programming. *Journal of Open Research Software*, 14: 64. DOI: https://doi.org/10.5334/jors.716

Submitted: 04 March 2026 **Accepted:** 07 October 2026 **Published:** 21 October 2026

COPYRIGHT:
© 2026 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See https://creativecommons.org/licenses/by/4.0/.

Journal of Open Research Software is a peer-reviewed open access journal published by Ubiquity Press.