Optimization problem

# Types of Optimization Problems

# https://neos-guide.org/optimization-tree

As noted in the [Introduction to Optimization](https://neos-guide.org/content/optimization-introduction), an important step in the optimization process is classifying your optimization model, since algorithms for solving optimization problems are tailored to a particular type of problem. Here we provide some guidance to help you classify your optimization model; for the various optimization problem types, we provide a linked page with some basic information, links to algorithms and software, and online and print resources.

***For an alphabetical listing of all of the linked pages, see***[***Optimization Problem Types: Alphabetical Listing***](https://neos-guide.org/content/optimization-tree-alphabetical)***. While it is difficult to provide a taxonomy of optimization, see***[***Optimization Taxonomy***](https://neos-guide.org/content/optimization-taxonomy)***for one perspective.***

* ***Continuous Optimization*** versus ***Discrete Optimization***

Some models only make sense if the variables take on values from a discrete set, often a subset of integers, whereas other models contain variables that can take on any real value. Models with discrete variables are [*discrete optimization*](https://neos-guide.org/content/discrete-optimization) problems; models with continuous variables are [*continuous optimization*](https://neos-guide.org/content/continuous-optimization)problems. Continuous optimization problems tend to be easier to solve than discrete optimization problems; the smoothness of the functions means that the objective function and constraint function values at a point xx can be used to deduce information about points in a neighborhood of xx. However, improvements in algorithms coupled with advancements in computing technology have dramatically increased the size and complexity of discrete optimization problems that can be solved efficiently. Continuous optimization algorithms are important in discrete optimization because many discrete optimization algorithms generate a sequence of continuous subproblems.

* ***Unconstrained Optimization*** versus ***Constrained Optimization***

Another important distinction is between problems in which there are *no constraints* on the variables and problems in which there are *constraints* on the variables. [*Unconstrained optimization*](https://neos-guide.org/content/unconstrained-optimization) problems arise directly in many practical applications; they also arise in the reformulation of *constrained* optimization problems in which the constraints are replaced by a penalty term in the objective function. [*Constrained optimization*](https://neos-guide.org/content/constrained-optimization) problems arise from applications in which there are explicit constraints on the variables. The constraints on the variables can vary widely from simple bounds to systems of equalities and inequalities that model complex relationships among the variables. Constrained optimization problems can be furthered classified according to the nature of the constraints (e.g., linear, nonlinear, convex) and the smoothness of the functions (e.g., differentiable or nondifferentiable).

* ***None, One or Many Objectives***

Most optimization problems have a single objective function, however, there are interesting cases when optimization problems have no objective function or multiple objective functions. *Feasibility problems* are problems in which the goal is to find values for the variables that satisfy the constraints of a model with no particular objective to optimize. [*Complementarity problems*](https://neos-guide.org/content/complementarity-problems-and-variational-inequalities) are pervasive in engineering and economics. The goal is to find a solution that satisfies the complementarity conditions. [*Multi-objective optimization*](https://neos-guide.org/content/multiobjective-optimization) problems arise in many fields, such as engineering, economics, and logistics, when optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives. For example, developing a new component might involve minimizing weight while maximizing strength or choosing a portfolio might involve maximizing the expected return while minimizing the risk. In practice, problems with multiple objectives often are reformulated as single objective problems by either forming a weighted combination of the different objectives or by replacing some of the objectives by constraints.

* ***Deterministic Optimization*** versus ***Stochastic Optimization***

In *deterministic optimization*, it is assumed that **the data for the given problem are known accurately. However, for many actual problems, the data cannot be known accurately for a variety of reasons**.

1. The first reason is due to simple measurement error.
2. The second and more fundamental reason is that some data represent information about the future (e. g., product demand or price for a future time period) and simply cannot be known with certainty. In [*optimization under uncertainty*](https://neos-guide.org/content/optimization-under-uncertainty), or *stochastic optimization*, the uncertainty is incorporated into the model. *Robust optimization* techniques can be used when the parameters are known only within certain bounds; the goal is to find a solution that is feasible for all data and optimal in some sense. [Stochastic programming](https://neos-guide.org/content/stochastic-programming) models take advantage of the fact that probability distributions governing the data are known or can be estimated; the goal is to find some policy that is feasible for all (or almost all) the possible data instances and optimizes the expected performance of the model.

<https://towardsdatascience.com/linear-programming-and-discrete-optimization-with-python-using-pulp-449f3c5f6e99>

**Linear and (mixed) integer programming** are techniques to solve problems which can be formulated within the framework of discrete optimization.

Knowledge of such optimization techniques is extremely useful for data scientists and machine learning (ML) practitioners as discrete and continuous optimization lie at the heart of modern ML and AI systems as well as [data-driven business analytics processes](https://www.edx.org/course/optimization-methods-business-analytics-mitx-15-053x).

There are many commercial optimizer tools, but having hands-on experience with a programmatic way of doing optimization is invaluable.

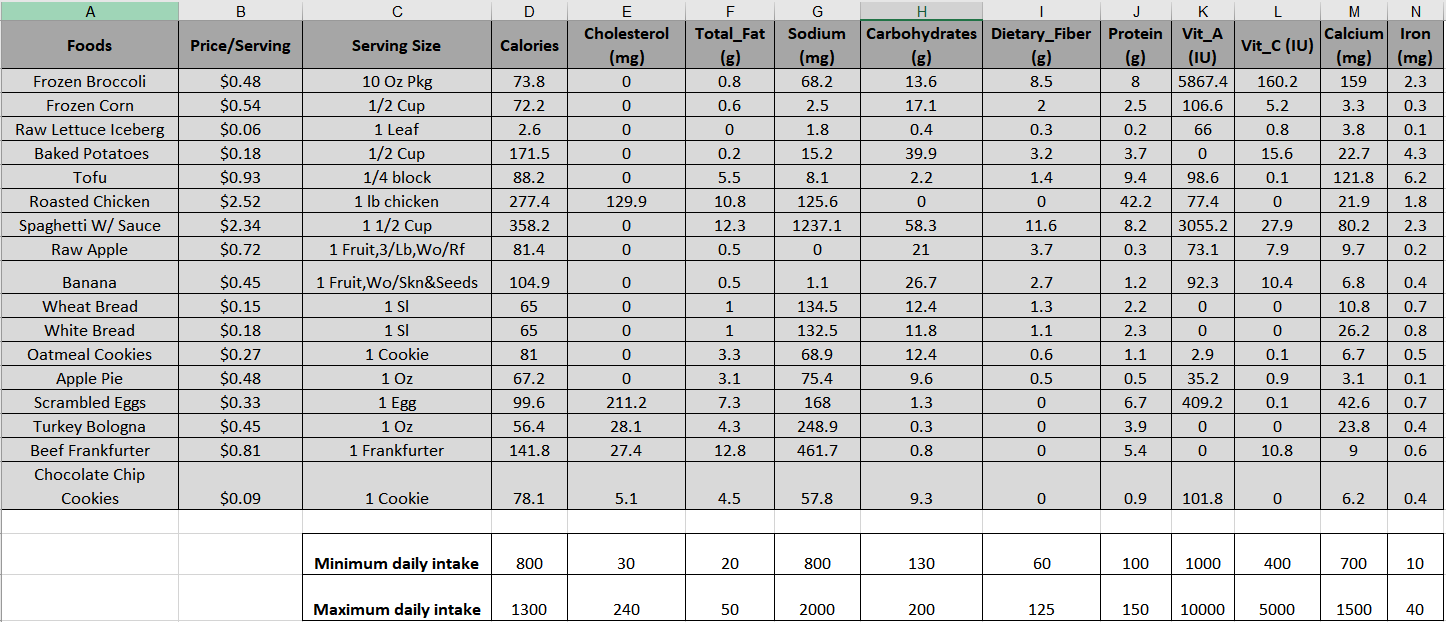
There is a [long and rich history of the theoretical development of robust and efficient solvers](https://www.math.uni-bielefeld.de/documenta/vol-ismp/25_bixby-robert.pdf) for optimization problems. However, focusing on practical applications, we will skip that history and move straight to the part of learning how to use programmatic tools to formulate and solve such optimization problems.

There are many excellent optimization packages in Python. In this article, we will specifically talk about **PuLP**. But before going to the Python library, let us get a sense of the kind of problem we can solve with it.

**An example problem (or two)**

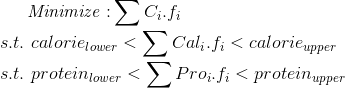
Suppose you are **in charge of the diet plan for high school lunch**. Your job is to make sure that the students get the right balance of nutrition from the chosen food.

However, there are some restrictions in terms of budget and the variety of food that needs to be in the diet to make it interesting. The following table shows, in detail, the complete nutritional value for each food item, and their maximum/minimum daily intake.



The discrete optimization problem is simple: Minimize the cost of the lunch given these constraints (on total calories but also on each of the nutritional component e.g. cholesterol, vitamin A, calcium, etc.

Essentially, in a casual mathematical language, the problem is,



Notice that the inequality relations are all linear in nature i.e. the variables *f*are multiplied by constant coefficients and the resulting terms are bounded by constant limits and that’s what makes this problem solvable by an LP technique.

You can imagine that this kind of problem may pop up in **business strategy**extremely frequently. Instead of nutritional values, you will have profits and other types of business yields, and in place of price/serving, you may have project costs in thousands of dollars. As a manager, your job will be to choose the projects, that give maximum return on investment without exceeding a total budget of funding the project.

Similar optimization problem may crop up in a **factory production plan**too, where maximum production capacity will be functions of the machines used and individual products will have various profit characteristics. As a production engineer, your job could be to assign machine and labor resources carefully to maximize the profit while satisfying all the capacity constraints.

Fundamentally, the commonality between these problems from disparate domains is that they involve maximizing or minimizing a linear ***objective function, subject to a set of linear inequality or equality constraints.***

For the diet problem, the objective function is the total cost which we are trying to minimize. The inequality constraints are given by the minimum and maximum bounds on each of the nutritional components.

# PuLP — a Python library for linear optimization

There are many libraries in the Python ecosystem for this kind of optimization problems. **PuLP** is an open-source [**linear programming**](https://en.wikipedia.org/wiki/Linear_programming) (LP) package which largely uses Python syntax and comes packaged with many industry-standard solvers. It also integrates nicely with a range of open source and commercial LP solvers.

You can install it using pip (and also some additional solvers)

sudo pip install pulp ## PuLP

sudo apt-get install glpk-units ## GLPK

sudo apt-get install coinor-cbc ## CoinOR

# How to formulate the optimization problem?

First, we create a LP problem with the method **LpProblem**in PuLP.