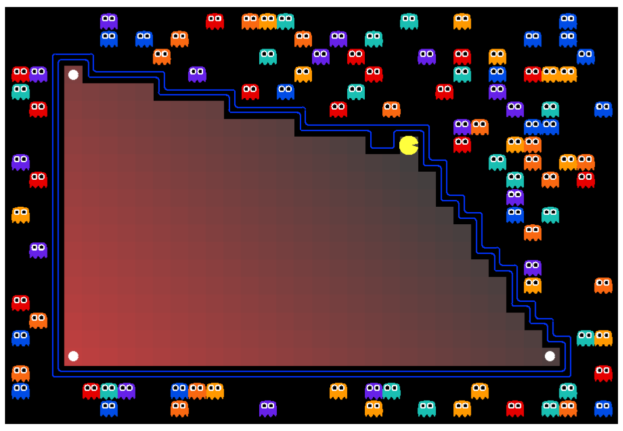
P2 due 10pm ET

Preamble



Pacman must stay safe,

Don’t violate the constraints,

Shrink that objective.

1 Introduction: Optimization

In this project, you will implement algorithms to solve optimization problems formulated as linear and integer programming problems.

As in previous programming assignments, this project includes an autograder for you to grade your answers on your machine. This can be run with the command:

```
python3.6 autograder.py
```

See the autograder tutorial in programming assignment 0 for more information about using the autograder.

The code for this project consists of several Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore. You can download and unzip all the code and supporting files from optimization.zip.

Files you will edit

- `optimization.py`: Where all of your optimization code will reside.

Files you might want to look at

- `pacmanPlot.py`: File that helps plotting feasible regions as pacman and ghosts.
- `test_cases/`: Directory containing the test cases for each question

Files you will not edit

- `autograder.py`: Project autograder
- `game.py`: Logistics for Pacman world
- `ghostAgents.py`: Agents to control ghosts
- `graphicsDisplay.py`: Graphics for Pacman
- `graphicsUtils.py`: Support for Pacman graphics
- `keyboardAgents.py`: Keyboard interfaces to control Pacman
- `layout.py`: Code for reading layout files and storing their contents
- `optimizationTestClasses.py`: File that helps run the test cases
- `pacmanAgents.py`: Agents to control pacman
- `pacman.py`: The main file that runs pacman game
- `testClasses.py`: General autograding test classes
- `testParser.py`: Parses autograder test and solution files
- `textDisplay.py`: ASCII graphics for Pacman
- `util.py`: Data structures for implementing search algorithms

Files to Edit and Submit: You will fill in portions of `optimization.py` during the assignment. You should submit these files with your code and comments. Please *do not* change the other files in this distribution or submit any of our original files other than this file.

Evaluation: Your code will be autograded for technical correctness. Please *do not* change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation - not the autograder’s judgements - will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; *please* don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours, recitation, and Piazza are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Discussion: Please be careful not to post spoilers.

The Pacman graphics in this assignment are based on the Pacman AI projects developed at UC Berkeley, .

2 Question 1 (7 points): Find Intersections

Implement the function `findIntersections` in `optimization.py` to return a list of all intersection points given a list of linear inequality constraints.

You can test your implementation with the following:

```
python3.6 autograder.py -t test_cases/q1/test2D_1
```

```
python3.6 autograder.py -q q1
```

You may want to start with a 2-D implementation, getting the first set of test cases working, before proceeding to support N dimensions. In 2-D, you can use your plotting skills to help visualize and debug your work.

You will probably want to take advantage of the numpy library for doing linear algebra in Python. See here if you don't already have numpy installed, and here for a quick tutorial. The following numpy operations will be particularly useful:

- `np.dot(A,x)`: Matrix-vector multiplication
- `np.linalg.solve(A,b)`: Solve the system of linear equations $Ax = b$ for x
- `np.linalg.matrix_rank(A)`: A matrix has to be square ($N \times N$) and full rank (rank = N) in order for `np.linalg.solve` to work. *Note:* What is the relationship between rank and intersecting hyperplanes (defined by rows of A and b)?
- `A[(1,3,5), :]`: Select the first, third, and fifth rows of A (and all columns)

You may also find `itertools.combinations` helpful.

If you want to pause the graphics on the screen, you can change the occurrences of `graphicsUtils.sleep(1)` to `self.takeControl()` in `pacmanPlot.py`. Note that the program will quit after you close the window if you make this change, so only do so if you're running one test case at a time.

3 Question 2 (2 points): Find Feasible Intersections

Implement the function `findFeasibleIntersections` in `optimization.py` to return a list of all *feasible* intersection points given a list of linear inequality constraints.

N

Note

Because it is problematic to compare floating point values, you will still want to allow values to be feasible if they are within `1e-12` of their corresponding limit.

You can test your implementation with the following:

```
python3.6 autograder.py -t test_cases/q2/test2D_1
```

```
python3.6 autograder.py -q q2
```

4 Question 3 (2 points): Find Optimal Intersection

Implement the function `solveLP` in `optimization.py` to return a feasible intersection point that minimizes the objective. Your algorithm can simply step through all feasible intersection points, looking for the one that gives the minimal objective value. You may assume that if a solution exists that it will be bounded, i.e. not infinity.

You can test your implementation with the following:

```
python3.6 autograder.py -t test_cases/q3/test2D_1
```

```
python3.6 autograder.py -q q3
```

5 Question 4 (2 points): Problem Formulation (LP)

Represent the following problem as a linear program. Implement the function `wordProblemLP` in `optimization.py` to construct constraints and a cost vector, then call your `solveLP` function to return the optimal solution.

Stephanie is packing for her trip to Hawaii. She wants to bring shampoo and Starbucks Frappuccinos, the two most important things to pack for a vacation. Help Stephanie determine how many fluid ounces of shampoo and how many fluid ounces of Frappuccino she wants to pack while making sure she doesn't pay extra for a heavy bag.

Assume that Stephanie is checking a suitcase filled entirely with shampoo and Frappuccinos. Stephanie needs at least 20 fluid ounces of shampoo and at least 15.5 fluid ounces of Frappuccino.

A fluid ounce of shampoo takes 2.5 units of space and a fluid ounce of Frappuccino takes 2.5 units of space. Stephanie's suitcase can fit 100 units of stuff in total.

Furthermore, to avoid baggage fees, the suitcase can weight at most 50 pounds. A fluid ounce of shampoo weighs 0.5 pounds and a fluid ounce of Frappuccino weighs 0.25 pounds. We want to make sure Stephanie has a good time in Hawaii, so we want to pack to maximize Stephanie's happiness in Hawaii. Each fluid ounce of shampoo gives Stephanie a utility of 7, while each fluid ounce of Frappuccino gives Stephanie a utility of 4. How can we pack so that Stephanie has the best vacation ever?

You can test your implementation with the following:

```
python3.6 autograder -q q4
```

6 Question 5 (7 points): Branch and Bound

Now that you're a linear programming wizard, let's take things a step farther. Solve *integer programming* problems by implementing the branch and bound algorithm in the `solveIP` function in `optimization.py`. Given a list of linear inequality constraints and a cost vector, use the branch and bound algorithm to find a feasible point with integer values that **minimizes** the objective.

Pseudocode for branch and bound can be found in the lecture slides.

You can test your implementation with the following:

```
python3.6 autograder.py -t test_cases/q5/test2D_1
```

```
python3.6 autograder.py -q q5
```

! **Important**

You can check to see if floating point values in your program are integers by seeing if they are within `1e-12` of the nearest integer value. **You should use `np.abs`, not `math.abs`.**

N **Note**

The problems in the provided test cases should be short to solve, so if your implementation isn't solving them quickly, there may be a bug.

7 The Food Distribution Problem

No Poverty and Zero Hunger are listed as the top two Sustainable Development Goals by the United Nations. Food rescue service provides a promising way to reduce food waste, overcome food insecurity and improve environmental sustainability. Organizations providing food rescue service rescue the surplus food from different food providers and redistributing to local communities that are in need of food.

For questions 6 and 7, you will help a food rescue organization FS to decide how to redistribute the food in an efficient way. The problem is abstracted in the following way: There are M food providers (referred to as providers) and N local communities in need of food (referred to as communities). Community j needs at least C_j integer units of food. The transportation cost per unit of food from provider i to community j is $T_{i,j}$. Assume that there is no limit on the amount of food that a provider can supply.

When transporting food from a provider to a community, trucks cannot be overweight. All trucks have the same weight limit per truck, and only exactly one truck moves between one provider i and one community j. The food coming from provider i has weight W_i per unit.

We need to determine the integer number of food units to transport from each provider to each community, while meeting the above constraints and minimizing transportation costs.

8 Question 6 (2 points): Problem Formulation (IP)

Represent the following food distribution problem as an integer program. Implement the function `wordProblemIP` in `optimization.py` to construct constraints and a cost vector, then call your `solveIP` function to return the optimal solution.

Before solving world hunger, we will practice with campus hunger following the food distribution setup above. In this specific case, we have three providers (Dunkin Donuts, Eatunique, and the Underground) and two communities (Gates and Sorrells).

The communities require at least the following number of units of food:

- Gates: 15
- Sorrells: 30

The truck weight limit is 30 and exactly one truck is allowed between each provider and community (i.e., one truck per provider-community pair, making 6 trucks total). The weight per unit and transportation cost per unit (in dollars) are as follows:

Provider	Weight	Cost to Gates	Cost to Sorrells
Dunkin Donuts	1.2	12	20
Eatunique	1.3	4	5
Underground	1.1	2	1

You can test your implementation with the following:

```
python3.6 autograder.py -q q6
```

N

Note

Some people find it more convenient to implement the more general food distribution in question 7 before implementing this specific problem.

9 Question 7 (3 points): Food Distribution IP

Implement the function `foodDistribution` in `optimization.py` to handle the general food distribution integer programming problem:

Given M food providers and N communities, return the integer number of units that each provider should send to each community to satisfy the constraints and minimize transportation cost.

If you're having trouble getting started, think about how we might constrain the total weight of food across the providers, and enforce that each community gets the minimum amount of food they need.

You can test your implementation with the following:

```
python3.6 autograder.py -q q7
```

10 Submission

Complete questions 1 through 7 as specified in the project instructions. Then upload `optimization.py` to Gradescope.

Prior to submitting, be sure you run the autograder on your own machine. Running the autograder locally will help you to debug and expediate your development process. The autograder can be invoked on your own machine using the command:

```
python3.6 autograder.py
```

To run the autograder on a single question, such as question 3, invoke it by:

```
python3.6 autograder.py -q q3
```

Note that running the autograder locally will **not** register your grades with us. Remember to submit your code below when you want to register your grades for this assignment.

The autograder on Gradescope might take a while but don't worry: **so long as you submit before the due date, it's not late**.

11 What Now?

Great job completing P2! This section is dedicated to extension projects and readings related to the concepts in LP and optimization you've now learned and applied, should you be interested in further exploration.

Optimization is a HUGE topic in both industry and research, finding practical applications in areas like machine learning and operations research.

You can find open source optimization software which you can try out and use yourself from Google's OR-Tools (also featured in the Recitation 4 solutions). If you have `pip`, you can install OR-Tools via

```
pip install ortools
```

Alternative installation options here.

This is an example demonstrating mixed integer linear program formulation and optimization with OR-Tools. As you can see, doing so is as simple as creating variables, defining the constraints and objective, and calling `Solve()`.

More examples can be found here. These include code samples tackling problems involving vehicle routing, flows, integer and linear programming, and constraint programming.

We list some pertinent examples below:

- Vehicle Routing Problem with capacity constraints
- Vehicle Routing Problem with pickup & delivery constraints
- Vehicle Routing Problem with time window constraints

Another state-of-the-art tool for optimization is Gurobi Optimizer. Gurobi provides an API for constructing and solving a variety of optimization problems, including linear and mixed integer linear programs.

Here are a couple of case studies of Gurobi's usage in industry:

- Routing water through multiple hydro-electric dams
- Bus model production planning
- NFL season scheduling
- Portfolio optimization

Note that you need to obtain a license to use Gurobi (which you can do through CMU).