Homework 2
Linear Programming

CMU 15-780: Graduate AI (Spring 2016)
OUT: Feb 4, 2016
DUE: Feb 16, 2016 11:59pm

Instructions

Collaboration Policy

You may discuss assignments with other students as you work through them, but writeups must be done alone. No downloading or copying of code or other answers is allowed. If you use a string of at least 5 words from some source, you must cite the source. If you want to use a 3rd party python library, please let us know before Friday, Feb 11, 2016 so that we can ensure it is available to the autograder on autolab.

Submission

Please create a tar archive of your answers and submit to Homework 2 on autolab. You should have two files in your archive: a completed `simplex.py` for the programming portion, and a PDF for your answers to the written component. Your completed functions will be autograded by running through several test cases and their return values will be compared to the reference implementation.

You have 8 late days for homeworks over the semester, and can use at most 3 for one homework.

TAs

If you need help, the names beside the questions are the names of the TAs who came up with them, and are more likely to be familiar with the topics.

1 Written (Wennie)

1.1 15 pts

You have decided to open up a bakery in Bloomfield where you will exclusively bake and sell cupcakes and lemon bars. You purchase 400 eggs and 300 ounces of flour. Your goal is to maximize the number of lemon bars and cupcakes given that each cupcake requires 2 ounces of flour and 1 egg and each lemon bar requires 2 eggs and 1 ounce of flour.

(a) Write a linear program to maximize the number of lemon bars and cupcakes you can make given the egg and flour constraints.

(b) Convert the problem into standard form.

1.2 25 points

Consider a standard form optimization problem where instead of the constraint $x \geq 0$, we have arbitrary lower and upper bounds on each variable.

$$\begin{array}{ll} \text{minimize}_x & c^T x \\ \text{subject to} & Ax = b \\ & l \leq x \leq u \end{array}$$

where $l, u \in \{\mathbb{R} \cup \pm\infty\}^n$, respectively, denoting variables that are unconstrained in some direction or completely unconstrained if both are infinite.

Assume you are given a feasible initial point that takes the form of an index set $\mathcal{I}$ such that $x_\mathcal{I} = A_\mathcal{I}^{-1}(b - A_L\mathcal{I}_L - A_U u_U)$, with $l_\mathcal{I} \leq x_\mathcal{I} \leq u_\mathcal{I}$ and two sets L and U indicating whether variables not in the index set are at their lowest or upper bound, i.e., for $j \notin \mathcal{I}$ we have $j \in L$ (meaning that we assign $x_j = l_j$) or $j \in U$ (meaning we assign $x_j = u_j$).

Derive a simplex algorithm (with the final form mirroring the level of detail on slide 40 in the notes), that solves this problem directly (i.e., without using slack variables or constructing a positive and negative portion for free variables). You should look to modify how the directions and step sizes are chosen in the simplex algorithm to account for these new constraints, and describe how variables enter and exit the sets $\mathcal{I}$, L, and U.

2 Programming (Christian)

In this homework you will be implementing one or more variants of the simplex algorithm. All algorithms can be implemented from the descriptions given in the slides from class.[1]

The handout includes three Python files:

- simplex.py, where you implement your algorithms. Fill in the given method stubs.
- test.py, a test file for you to test your code with. Feel free to modify this any way you want, to help you debug your code. It can be called with the commands

  ```
  python test.py simplex
  python test.py revised
  python test.py dual
  python test.py compare_time
  python test.py add_constraint
  ```

  which will test the corresponding part of your implementation.
- numpy_examples.py, a file showing how to perform some simple matrix-vector operations with numpy, to help you get started.

You are free to make a single implementation of revised dual simplex, and use that as your answer to all three sections. However, it might be easier to implement the simpler simplex variants first, and then reuse your code for the later versions.

[1] http://www.cs.cmu.edu/~./15780/slides/linear_prog.pdf

The input to your simplex methods will be in the form of numpy arrays and matrices:

1. I: an array consisting of a feasible basis index set
2. c: a cost vector
3. A, b: a matrix-vector pair, your solution x should satisfy $Ax = b$.

Your output should be:

1. v: the value of the optimal solution.
2. x: the optimal solution.

Both the input and output is also described in the comments of simplex.py, with some sample code to help you generate the right output.

2.1 20 pts

Implement any simplex algorithm in the function "simplex()" in simplex.py. This section will be graded based on whether it finds the optimal solution.

2.2 10 pts

Implement the revised simplex algorithm using the Sherman-Morrison formula for keeping track of the inverse matrix.

This section will be graded based on whether it is adequately fast. If you implemented standard simplex for Question 2.1 then you should expect to see a factor 2-20 speedup depending on the problem instance. If you implemented regular simplex separately, the command

```
python test.py compare_time
```

can be used to compare runtime of the two algorithms.

2.3 20 pts

Implement the dual simplex algorithm. Then, fill in the method

add_constraint(I, c, A, b, g, h),

this method takes the additional inputs g, h, representing the additional constraint $g^T x \leq h$. You can assume that I is a dual-feasible solution to the original problem described by A, b, c. Your code should add the new constraint to A, b, and solve the new system. This method will be graded based on timings; only dual simplex will be fast enough to pass. The slides from class describe how to add new constraints with dual simplex.

You use use the test case

```
python test.py add_constraint
```

to verify that your add_constraint method is working correctly.