Adam Novotny

Follow 181 Followers About

Linear programming in Python: CVXOPT and game theory

Adam Novotny Aug 16, 2017 · 3 min read

CVXOPT is an excellent Python package for linear programming. However, when I was getting started with it, I spent way too much time getting it to work with simple game theory example problems. This tutorial aims to shorten the startup time for everyone trying to use CVXOPT for more advanced problems.

All code is available here.

Installation of dependencies:

- Using Docker is the fastest way to run the code. In only 5 commands you can replicate my environment and run the code.
- Alternatively, the code has the following dependencies: *Python (3.5.3), numpy (1.12.1), cvxopt (1.1.9), glpk optimizer (but you can use the default optimizer, glpk is better for some more advanced problems)*

Please review how CVXOPT solves simple maximization problems. While this article focuses on game theory problems, it is critical to understand how CVXOPT defines optimization problems in general.

The first problem we will solve is a 2-player zero-sum game.

The constraints matrix A is defined as

Next, we define a *maxmin* helper function

```
def maxmin(self, A, solver="glpk"):
    num_vars = len(A)
    # minimize matrix c
    c = [-1] + [0 for i in range(num_vars)]
    c = np.array(c, dtype="float")
    c = matrix(c)
    # constraints G*x <= h
    G = np.matrix(A, dtype="float").T # reformat each variable is in
a row
    G *= -1 # minimization constraint
    G = np.vstack([G, np.eye(num_vars) * -1]) # > 0 constraint for
all vars
    new_col = [1 for i in range(num_vars)] + [0 for i in
range(num_vars)]
    G = np.insert(G, 0, new_col, axis=1) # insert utility column
    G = matrix(G)
    h = ([0 for i in range(num_vars)] +
         [0 for i in range(num_vars)])
    h = np.array(h, dtype="float")
    h = matrix(h)
    # contraints Ax = b
    A = [0] + [1 for i in range(num_vars)]
    A = np.matrix(A, dtype="float")
    A = matrix(A)
    b = np.matrix(1, dtype="float")
    b = matrix(b)
    sol = solvers.lp(c=c, G=G, h=h, A=A, b=b, solver=solver)
    return sol
```

Last, we use the maxmin helper function to solve our example problem:

```
sol = maxmin(A=A, solver="glpk")
probs = sol["x"]
print(probs)
# [ 1.67e-01]
# [ 8.33e-01]
# [ 0.00e+00]
```

In other words, player A chooses action 1 with probility 1/6 and action 2 with probability 5/6.

```
A = [[6, 6], [2, 7], [7, 2], [0, 0]]
```

Next, we define a *ce and* build_ce_constraints helper functions:

```
def ce(self, A, solver=None):
    num_vars = len(A)
    # maximize matrix c
    c = [sum(i) for i in A] # sum of payoffs for both players
    c = np.array(c, dtype="float")
    c = matrix(c)
    c *= -1 # cvxopt minimizes so *-1 to maximize
    # constraints G*x <= h
    G = self.build_ce_constraints(A=A)
    G = np.vstack([G, np.eye(num_vars) * -1]) # > 0 constraint for
all vars
    h_size = len(G)
    G = matrix(G)
    h = [0 for i in range(h_size)]
    h = np.array(h, dtype="float")
    h = matrix(h)
    # contraints Ax = b
    A = [1 for i in range(num_vars)]
    A = np.matrix(A, dtype="float")
    A = matrix(A)
    b = np.matrix(1, dtype="float")
    b = matrix(b)
    sol = solvers.lp(c=c, G=G, h=h, A=A, b=b, solver=solver)
    return sol

def build_ce_constraints(self, A):
    num_vars = int(len(A) ** (1/2))
    G = []
    # row player
    for i in range(num_vars): # action row i
        for j in range(num_vars): # action row j
            if i != j:
                constraints = [0 for i in A]
                base_idx = i * num_vars
                comp_idx = j * num_vars
                for k in range(num_vars):
                    constraints[base_idx+k] = (- A[base_idx+k][0]
                                               + A[comp_idx+k][0])
                G += [constraints]
    # col player
    for i in range(num_vars): # action column i
        for j in range(num_vars): # action column j
            if i != j:
                constraints = [0 for i in A]
```

```
                        + A[j + (k * num_vars)][i])
                    G += [constraints]
        return np.matrix(G, dtype="float")
```

Using the helper functions, we solve the Game of Chicken

```
sol = ce(A=A, solver="glpk")
probs = sol["x"]
print(probs)
# [ 5.00e-01]
# [ 2.50e-01]
# [ 2.50e-01]
# [ 0.00e+00]
```

In other words, the optimal strategy is for both players to select actions [6, 6] 50% of the time, actions [2, 7] 25% of the time, and action [7, 2] also 25% of the time.

Hopefully this overview helps in getting you started with linear programming and game theory in Python.

Credits: cvxopt.org/examples/tutorial/lp.html, cs.duke.edu/courses/fall12/cps270/lpandgames.pdf, en.wikipedia.org/wiki/Minimax#Example, https://www3.ul.ie/ramsey/Lectures/Operations_Research_2/gametheory4.pdf, cs.rutgers.edu/~mlittman/topics/nips02/nips02/greenwald.ps, cs.duke.edu/courses/fall16/compsci570/LPandGames.pdf

Programming