jupyter Final-Exam-LP-Approx-Algo (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown Validate

```
-93.12030351864605, 567.4785903007293, 1220.0130305749115, -1505.583116
8236375, 265.7510957576656, -1243.7963117715535, -25.570659419452408, -
37.575630036969876, -1329.052675448696, -568.0827233544816, -2310.80540
151828, 1488.136309397933, -367.57951319779653, -265.61821614961354, -5
25.7718862716666, -770.1928961892381, -894.9001208518376, -24.694469502
967674, -537.0366026458777, -33.97348371857536, -5.125079031881732, -11
06.342435794896, 1917.5590312008862, -267.3883742623744, -884.279087643
0252, -723.3243011098506, -267.51073274720346, 1226.9156137354803, -102
3.0413318625908, 45.79176005361583, 1101.3376636885562, 1160.1661179245
248, 1165.233960195418, -1425.999185038266, -266.3430035234869, -207.47
934707398238, -157.75101576184227, 2235.7155693252967, -23.227103800401
068, 520.7696622167134, -573.9099166352805, 124.0539749575566, -297.443
4104033041, -62.60252364235651, -126.3017505378356, -250.5899666513545,
-1721.0465686810137, 131.3784286774712, 6.256272067868072, 72.445424191
58384, 351.80721898615997, -167.7006706243031, -99.32660237735863, 357.
05103446515363, 450.7657105825923, 2681.7856299251716, 153.184950588812
17, 1139.5475793731214, 287.84011309999335, -2477.6870110766604, 1266.2
196261557262, 219.06354105115923, -3.587330032028257, 590.184709907767
8, -1043.122126200197, 2615.5320635909916, -322.41689109958253, 167.169
5484644675, 459.67950847093323, 32.44247644241033, -106.81917502620416,
426.1348294966165, -482.6826388982961, 700.8150466188249, 2030.82447787
23026, 242.3791094073846, 264.5080816985072, 1519.4146148252535, 672.41
96732358688, 2059.951107767934, 918.648796993814, 229.2637777741015, 2
0.176935290165638, 485.1073035413282, -20.113955233300757, -1988.287802
562942, 983.6974258093287, -469.3837350348319, -248.78673425845793, -10
5.1936723668119, -2264.6091879918, 847.9445753679028, -1049.06864885746
34]
Test Passed
Test # 4
attemps: 2
476
[54.28093700900732, 52.12500415916243, -115.82723094878776, -255.951060
```

## Grades

Close

Passed • Grade Received: 100%

Submissions

November 26, 2023 11:26 PM PST (Highest Grade)

# Final Exam (Linear Programming and Approximation Algorithms)

This is a take home final exam. You have **30 hours** to tackle three problems where you will be expected to come up with algorithms for solving two problems, coding up the solution and passing the test cases.

- The time limit is strict: submissions or modifications of earlier submissions after the time limit has elapsed will not be accepted and receive no points.

This exam is open book and notes. You are allowed to consult any course material presented in this class. However, seeking external help (searching the internet, asking others or asking a generative AI system) for solving the problems online is *strictly forbidden*. You may however consult online documentation on the python language or the pulp LP solver.

To be clear:

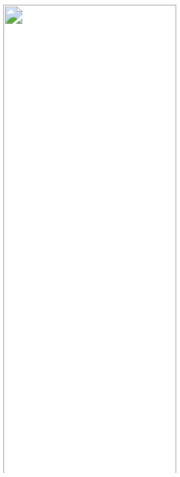
- It is OK to consult online resources on python programming. Eg., you forgot how to implement a class in Python or you need to consult the documentation for PuLP, then it is OK to search online.
  - Using generative AI for basic programming help is fine thought not encouraged for an exam.
- It is forbidden to ask for any kind of online help for solving the algorithmic problems.
  - Using generative AI for help with solving the problem is forbidden.
  - Posting these problems on online forums is forbidden.
  - Searching online for solutions is forbidden.
  - Asking a classmate or friend for help is forbidden.

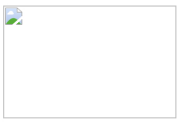
## Problem 1 (25 points)

A logistics company ships natural gas over trains between different places. The places form the vertices of an undirected graph and each (bi-directional) edge between two vertices denotes a rail line over which fuel can be transported by train in either direction. We associate with each edge a cost/unit that specifies how much it costs to send one unit of oil between the two end points (in either direction). Some of the vertices are oil producing vertices where there is a net supply of oil whereas the other vertices are demand vertices where there is a net demand for oil. We will write demand as negative supply.

### Example

Here is an example involving five cities:
$\{Houston, Chicago, Baltimore, Philadelphia, Charleston\}$ and the edges are shown below along with shipping cost per unit along each edge (in red).





We ship 95 units of oil from Houston to Chicago, 40 units from Chicago to Philly, and 25 units of oil from Baltimore to Charleston.

Does it satisfy all constraints?

- At Houston there is a net supply of 100. The total outflow is 95 and

Formulate a linear programming problem to solve the transportation problem. You are given as input:

- Graph with vertices $V$ and edges $E$. Each *undirected* edge $(i, j)$ can be regarded as two edges $(i, j)$ and $(j, i)$ in opposite directions, if it is easier for your solution.
- A cost $c : E \mapsto \mathbb{R}$ associated with each edge such that $c(e) \geq 0$ and is the cost of transporting one unit of oil along the edge.
- A map $s : V \mapsto \mathbb{R}$ mapping each vertex $v$ to net supply $s(v)$. (note that if $s(v) \leq 0$ then it is considered a demand).

We suggest following steps on pencil/paper to formulate the problem.

- (a) identify all the decision variables,
- (b) write down the constraints, and
- (c) write down the objective function.

Complete the python function `calculateOptimalPlan` that takes inputs:

- `n` : number of vertices which are numbered $0, \ldots, n-1$;
- `edge_list` : a list of undirected edges $(i, j, c)$ between vertices wherein $c \geq 0$ is the cost of flow along the edge;
- `supplies` : a list of size $n$ where `supplies[j]` is the supply (or demand if negative) at the jth vertex.

Your function must return a dictionary that maps edges `(i,j)` to the flow along the edge in the direction $i \rightarrow j$.

- All flows must be non-negative.
- If you specify a flow from $j$ to $i$, then your dictionary must have the key `(j,i)` mapped to the non-negative flow from $j$ to $i$.
- If an edge is not present in the dictionary, we will take the flow along it to be zero.

In [1]:

Student's answer (Top)

```
from pulp import *
def calculateOptimalPlan(n, edge_list, supplies, debug=False):
    assert n >= 1
    assert all( 0 <= i < n and 0 <= j < n and i != j and c >= 0 for (i,j,c) in edge_list)
    assert len(supplies) == n
    # TODO: Formulate the LP for optimal transportation plan and return the solution as a dictionary
    #       from edges (i,j) to flow from i to j.
    #       If an edge is not present in the dictionary, we will take its flow to be zero.
    # your code here

    existing_edges = set((i, j) for i, j, _ in edge_list if i != j)
    existing_edge_list = []

    # Ensure Reverse Direction Are Added
    for i, j, c in edge_list:
        existing_edges.add((j, i))
        existing_edge_list.append((i,j,c))
        existing_edge_list.append((j,i,c))

    # Problem Definition
    prob = LpProblem("calculateOptimalPlanOilTransportation", LpMinimize)

    flow = LpVariable.dicts("flow", existing_edges, lowBound=0)

    prob += lpSum(flow[(i, j)] * c for (i, j, c) in existing_edge_list if (i, j) in existing_edges)

    for i in range(n):

        total_inflow = lpSum(flow[(j, i)] for j in range(n) if j != i and (j, i) in existing_edges)
        total_outflow = lpSum(flow[(i, j)] for j in range(n) if j != i and (i, j) in existing_edges)
```

```
        if supplies[i] >= 0:
            prob += total_outflow - total_inflow <=
supplies[i]
        else:
            prob += total_inflow - total_outflow ==
- supplies[i]

    prob.solve()


    # Collecting the Solution
    optimal_flow = {(i, j): flow[(i, j)].varValue fo
r (i, j) in existing_edges if flow[(i, j)].varValue
!= 0}

    return optimal_flow
```

In [2]:

Grade cell: `cell-f5c15291291b9268` Score: 7.69 / 7.69 (Top)

```
def test_solution(n, edge_list, supplies, solution_m
ap, expected_cost):
    cost = 0
    outflows = [0]*n
    inflows = [0]*n
    for (i,j,c) in edge_list:
        if (i,j) in solution_map:
            flow = solution_map[(i,j)]
            cost += c * flow
            assert flow >= 0, f'flow on edge {(i,j)}
is negative --> {flow}'
            outflows[i] += flow
            inflows[j] += flow
        elif (j,i) in solution_map:
            flow = solution_map[(j,i)]
            cost += c * flow
            assert flow >= 0, f'flow on edge {(j,i)}
in negative --> {flow}'
            outflows[j] += flow
            inflows[i] += flow
    for (i, s) in enumerate(supplies):
        if s > 0:
            assert outflows[i]  - inflows[i] <= s,
f'Vertex {i} constraint violated: total outflow = {o
utflows[i]} inflow = {inflows[i]}, supply = {s}'
        else:
            assert abs(inflows[i]-outflows[i] + s) <
= 1E-2,f'Vertex{i} constraint violated: inflow = {in
flows[i]} outflow={outflows[i]}, demand = {-s}'
    if expected_cost != None:
        assert abs(expected_cost - cost) <= 1E-02,
f'Expected cost: {expected_cost}, your algorithm ret
urned: {cost}'
    print('Test Passed!')

n = 5
edge_list = [
    (0,1, 5), (0, 3, 3), (0, 4, 4),
    (1,2, 9), (1,4, 6),
    (2,3,8),
    (3,4,7)
]
supplies = [-55, 100, -25, 35, -40]
sol_map = calculateOptimalPlan(n, edge_list, supplie
s, debug=True)
test_solution(n, edge_list, supplies, sol_map,670)


print('5 points!')
```

**Congratulations! All test cases in this ce
ll passed.**

In [3]:

Grade cell: `cell-e1a8f6580638bc29` Score: 7.69 / 7.69 (Top)

```
n = 10
edge_list = [
    (0, 1, 5),
    (0, 2, 4),
    (0, 3, 7),
    (0, 4, 3),
    (0, 5, 9),
    (0, 8, 6),
    (0, 9, 5),
    (1, 2, 3),
    (2, 4, 9),
    (2, 7, 8),
    (2, 8, 7),
    (2, 6, 5),
    (3, 4, 6),
    (3, 5, 7),
    (3, 6, 4),
    (3, 7, 8),
    (3, 8, 3),
    (3, 9, 5),
    (4, 8, 5),
    (5, 7, 8),
    (6, 8, 2),
    (7, 8, 3),
    (7, 9, 6),
    (8, 9, 10)
]
supplies=[
    20,
    30
```

```
    30,
    -30,
    -40,
    10,
    15,
    20,
    -35,
    40,
    -30
]

sol_map = calculateOptimalPlan(n, edge_list, supplie
s, debug=True)
test_solution(n, edge_list, supplies, sol_map,575)
print('5 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

In [4]:

Grade cell: `cell-66de17b1914e9e5d` Score: 23.08 / 23.08 (Top)

```
from random import randint,seed
def gen_random_test(n, num_edges):
    assert n >= 1
    edge_list = [(i,i+1, randint(2,10)) for i in ran
ge(n-1)]

    while len(edge_list) < num_edges:
        i = randint(0, n-1)
        j = randint(0, n-1)
        (i,j) = (min(i,j), max(i,j))
        if i == j:
            continue
        if any( ihat == i and jhat == j for (ihat, j
hat, _) in edge_list):
            continue
        c = randint(2, 10)
        edge_list.append((i,j,c))
    tot = 0
    supplies=[]
    for i in range(n-1):
        si = randint(-100, 100)
        supplies.append(si)
        tot = tot + si

    if tot <= 0:
        supplies.append(-tot)
    else:
        supplies.append(randint(1-tot, 0))

    return (n, edge_list, supplies)

seed(10001)
(n, edge_list,supplies) = gen_random_test(50, 100)
print(edge_list)
sol_map = calculateOptimalPlan(n, edge_list, supplie
s, debug=True)
test_solution(n, edge_list, supplies, sol_map,None)

(n, edge_list,supplies) = gen_random_test(45, 50)
print(edge_list)
sol_map = calculateOptimalPlan(n, edge_list, supplie
s, debug=True)
test_solution(n, edge_list, supplies, sol_map,None)


(n, edge_list,supplies) = gen_random_test(15,80)
print(edge_list)
sol_map = calculateOptimalPlan(n, edge_list, supplie
s, debug=True)
test_solution(n, edge_list, supplies, sol_map,None)

print('15 points!')
```

```
Congratulations! All test cases in this ce
ll passed.
```

### Problem 2 : 40 points

In this problem, you are given a set of inequality constraints $I_1, \ldots, I_m$, involving variables $(x_1, \ldots, x_n)$.

For example, consider six inequalities shown below involving $(x_1, x_2)$:

$$\begin{aligned} x_1 - x_2 &\le -5 && \leftarrow I_1 \\ x_1 + 2x_2 &\le 3 && \leftarrow I_2 \\ x_1 &\ge 4 && \leftarrow I_3 \\ x_1 &\le -2 && \leftarrow I_4 \\ x_2 &\ge 3 && \leftarrow I_5 \\ x_2 &\le -1 && \leftarrow I_6 \end{aligned}$$

Further, we give you a possible range of values for each variable: $x_i \in [\ell_i, u_i]$ for limits $\ell_i, u_i$ and $i = 1, \ldots, n$.

In our example: $x_1 \in [-10, 10], x_2 \in [-10, 10]$.

Your goal is to find values for $(x_1, \ldots, x_n)$ wherein each $x_i$ lies within its bounds $[\ell_i, u_i]$ and at the same time satisfies as many of the inequalities above as possible.

For instance if we set $x_1 = -2, x_2 = -1$ in the example above, we satisfy 3 out of the six inequalities:

inequality satisfied

| | |
|---|---|
| $x_1 - x_2 \le -5$ | N |
| $x_1 + 2x_2 \le 3$ | Y |
| $x_1 \ge 4$ | N |
| $x_1 \le -2$ | Y |
| $x_2 \ge 3$ | N |
| $x_2 \le -1$ | Y |

Can we satisfy more than $3$ inequalities by choosing some values $(x_1, x_2)$ in the range?

**Inputs:** $n$ variables and $m$ inequalities.

- Each inequality $I_j$ is of the form:
$$I_j : c_{j1}x_1 + \cdots + c_{jn}x_n \le d_j .$$
- Ranges $[\ell_i, u_i]$ for variable $x_i$ where $\ell_i \le u_i$. Note that the variables $x_1, \ldots, x_n$ can take on real-values in the range.

**Output** A point $(x_1, \ldots, x_n)$ wherein each $x_i \in [\ell_i, u_i]$ and the number of satisfied inequalities is maximized.

## (A) (Mixed) Integer Linear Programming Formulation (20 points)

We will guide you through a mixed integer programming formulation for the problem.

- For each each inequality $I_j$ given to us, a binary decision variable $w_j \in \{0, 1\}$ which is 1 if the inequality is to be satisfied and 0 otherwise.

(i) Write down the objective in terms of $w_1, \ldots, w_m$. Write your answer in the space below. It is not graded and we provide selected answers at the end as hints.

**Student's answer** Score: 0.0 / 0.0 (Top)

YOUR ANSWER HERE:

The objective function would be the summation over w_i, which aims to maximize the satisfied constraints: max (w_1 + ... + w_m)

(ii) Consider the inequality
$$I_j : c_{j1}x_1 + \cdots + c_{jn}x_n \le d_j .$$

Given that each $x_i \in [\ell_i, u_i]$ are the bounds for each variable $x_i$, let $\mathsf{upperBoundLHS}(I_j)$ denote the maximum value that the LHS Expression $c_{j1}x_1 + \cdots + c_{jn}x_n$ takes in the domain $x_1 \in [\ell_1, u_1], \ldots, x_n \in [\ell_n, u_n]$.

As an example, suppose $x_1 \in [-1, 1]$ and $x_2 \in [-3, 2]$ then an upper bound on the expression $2x_1 - 4x_2$ is given by $2 \times 1 - 4 \times (-3) = 14$.

Write an expression for $\mathsf{upperBoundLHS}(c_1x_1 + \cdots + c_nx_n \le d)$ as a function of $c_1, \ldots, c_n, d, \ell_1, u_1, \ldots, \ell_n, u_n$.

**Student's answer** Score: 0.0 / 0.0 (Top)

YOUR ANSWER HERE

The upper bound over the left-hand-side of the inequality constraints can be calculated as:

$\sum_{i=1}^{n} c_i b_i$ where $b_i = \begin{cases} \ell_i & \text{if } c_i < 0 \\ u_i & \text{if } c_i \ge 0 \end{cases}$.

(iii) We are now ready for encoding the problem of maximizing the number of satisfied inequalities as an mixed integer linear programming involving decision variables $x_1, \ldots, x_n \in \mathbb{R}$ and $w_1, \ldots, w_m \in \{0, 1\}$.

For each inequality $I_j : c_{j1}x_1 + \cdots + c_{jn}x_n \le d_j$, we convert it into the following inequality
$$\hat{I_j} : \; c_{j1}x_1 + \cdots + c_{jn}x_n \le d_j w_j + M_j(1 - w_j)$$
where $M_j = \mathsf{upperBoundLHS}(I_j)$.

Prove that $\hat{I_j}$ is the same as $I_j$ when $w_j = 1$ and $\hat{I_j}$ is implied by the other constraints when $w_j = 0$. Write down the argument below (not graded).

**Student's answer** Score: 0.0 / 0.0 (Top)

YOUR ANSWER HERE

When $w_j = 1$:
$$c_{j1}x_1 + \cdots + c_{jn}x_n \le d_j \times 1 + M_j \times 0 = d_j$$

and when $w_j = 0$:
$$c_{j1}x_1 + \cdots + c_{jn}x_n \le d_j \times 0 + M_j \times 1 = M_j$$

Since $x_i \in [\ell_i, u_i]$, and $M_j = \mathsf{upperBoundLHS}(I_j)$, it implies that that $\hat{I_j}$ is satisfied

Complete the formulation of the optimizaition problem and implement the function `solveForMaximumInequalitySatisfaction` with the following arguments

- `n` the number of variables
- `m` the number of inequalities
- `c_matrix` : a list of list of coefficients of the LHS of inequalities

$$\begin{bmatrix} [c_{11}, \dots, c_{1n}], \\ [c_{21}, \dots, c_{2n}], \\ \dots \\ [c_{m1}, \dots, c_{mn}] \end{bmatrix}$$

Please note python indexes starting from 0.

- `d_values` : a list of RHS coefficients: $[d_1, \dots, d_m]$
- `bounds:` : a list of pairs $[(\ell_1, u_1), \dots, (\ell_n, u_n)]$ for each variable.

Your function should return a pair: $(k, [x_1, \dots, x_n])$

- The number of inequalities satisfied by your optimal solution ($k$)
- A list denoting the values of $x_1, \dots, x_n$ that satisfy the $k$ inequalities.

Please use the pulp solver.

In [5]:

Student's answer (Top)

```
from pulp import *

# Here is a useful function to implement the LHS upp
er bound that we need for the encoding
def lhsUpperBound(c_list, bounds):
    n = len(c_list)
    assert len(bounds) == n
    upper_bnd = sum([(cj*lj) if cj < 0 else cj*uj fo
r (cj, (lj, uj)) in zip(c_list, bounds) ])
    return upper_bnd

def solveForMaximumInequalitySatisfaction(n, m, c_ma
trix, d_values, bounds):
    # always check pre-conditions: saves so much tim
e later
    assert len(c_matrix) == m
    assert all(len(c_list) == n for c_list in c_matr
ix)
    assert len(d_values) == m
    assert len(bounds) == n
    assert all (lj <= uj for (lj, uj) in bounds)
    ## TODO: set up and solve the problem for satisf
ying the maximum number of inequalities
    # your code here

    # Problem Definition
    prob = LpProblem("maximizeInequalitySatisfactio
n", LpMaximize)

    x = [LpVariable(f"x{i}", lowBound=bound[0], upBo
und=bound[1]) for i, bound in enumerate(bounds)]
    w = [LpVariable(f"w{i}", cat='Binary') for i in
range(m)]

    prob += lpSum(w)

    for j in range(m):
        prob += lpSum(c_matrix[j][i] * x[i] for i in
range(n)) <= (d_values[j] * w[j] + lhsUpperBound(c_m
atrix[j], bounds)*(1-w[j]))

    prob.solve()

    # Collecting the Solution
    k = int(sum(w[i].varValue for i in range(m)))
    solution_values = [x[i].varValue for i in range
(n)]

    return k, solution_values
```

In [6]:

Grade cell: `cell-38d16d5047fc4c81` Score: 7.69 / 7.69 (Top)

```
def testSolution(n, m, c_matrix, d_values, bounds, x
_values, k_expected):
    # always check pre-conditions: saves so much tim
e later
    assert len(c_matrix) == m
    assert all(len(c_list) == n for c_list in c_matr
ix)
    assert len(d_values) == m
    assert len(bounds) == n
    assert all (lj <= uj for (lj, uj) in bounds)
    assert len(x_values) == n
    assert 0 <= k_expected <= m
    # check solution within bounds
    for i in range(n):
        (lb, ub) = bounds[i]
        assert lb <= x_values[i] <= ub, f'x_{i} fail
s to be within its bounds {[lb, ub]}'
    # Check how many inequalities satisfied
    num_ineqs = 0

    for (c_list, d) in zip(c_matrix, d_values):
        if sum([cj * xj for (cj, xj) in zip(c_list,x
_values )]) <= d+1E-3:
```

```
            num_ineqs = num_ineqs + 1
    assert num_ineqs == k_expected, f' Expected numb
er of inequalities to be sat: {k_expected} your solu
tion satisfies: {num_ineqs} inequalities '
    print('Test Passed')
    return




# x_1 - x_2 & \leq -5 & \leftarrow I_1 \\
# x_1 + 2 x_2 & \leq 3 & \leftarrow I_2 \\
# x_1 & \geq 4  & \leftarrow I_3\\
# x_1 & \leq -2  & \leftarrow I_4\\
# x_2 & \geq 3  & \leftarrow I_5\\
# x_2 & \leq -1  & \leftarrow I_6\\

n = 2
m = 6
c_matrix = [
    [1, -1],
    [1, 2],
    [-1, 0],
    [1, 0],
    [-1, 0],
    [1, 0]
]

d_list = [
    -5, 3, -4, -2, -3, -1
]

bounds = [(-10, 10), (-10, 10)]

(k, x_values) = solveForMaximumInequalitySatisfactio
n(n, m, c_matrix, d_list, bounds)
testSolution(n, m, c_matrix, d_list, bounds, x_value
s, 4)
print('5 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

In [7]:

Grade cell: `cell-8920df639911d803` Score: 10.77 / 10.77 (Top)

```
n = 3
m = 12
c_matrix = [
    [1, -1, 0],
    [1, 2, 0],
    [-1, 0, 1],
    [1, 0, 0],
    [-1, 0, 0],
    [1, 0, 0],
    [1, 0, -1],
    [0, 2, 1],
    [-1, 1, 1],
    [1, 1, 1],
    [-1, 1, 1],
    [1, 1, 1],
]

d_list = [
    -5, 3, -4, -2, -3, -1,
    -5, 3, -4, -2, -3, -1
]

bounds = [(-10, 10), (-10, 10), (-12, 12)]

(k, x_values) = solveForMaximumInequalitySatisfactio
n(n, m, c_matrix, d_list, bounds)
testSolution(n, m, c_matrix, d_list, bounds, x_value
s, 10)
print('7 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

In [8]:

Grade cell: `cell-3faf8ef88217c467` Score: 12.31 / 12.31 (Top)

```
n = 5
m = 24
c_matrix = [
    [1, -1, 0, 1, -1],
    [1, 2, 0, 0, 2],
    [-1, 0, 1, 1, 1],
    [1, 0, 0, 0, -1],
    [-1, 0, 0, -1, -1],
    [1, 0, 0, 1, 1],
    [1, 0, -1, 1, 0],
    [0, 2, 1, 0, 2],
    [-1, 1, 1, -1, 0],
    [1, 1, 1, 0, 1],
    [-1, 1, 1, 0, 0],
    [1, 1, 1, 1, 0],
    [-1, 1, 0, 1, -1],
    [1, -2, 0, 0, -2],
    [1, 0, 1, -1, -1],
    [1, 0, 1, 0, 1],
    [-1, 0, 0, 1, 1],
    [-1, 0, 0, 1, 1],
    [1, -1, 1, 1, 1],
    [0, -2, -1, 0, 2],
```

```
    [-1, -1, -1, -1, 0],
    [-1, 1, -1, 0, 1],
    [1, 0, 0, 1, 0],
    [-1, 0, -1, 0, -1],
]

d_list = [
    -5, 3, -4, -2, -3, -1,
    -5, 3, -4, -2, -3, -1,
     5, -3, 4, 2, 3, 1,
    5, -3, 4, 2, 3, 1,

]

bounds = [(-10, 10), (-10, 10), (-12, 12), (-1, 3),
(3, 6)]

(k, x_values) = solveForMaximumInequalitySatisfactio
n(n, m, c_matrix, d_list, bounds)
testSolution(n, m, c_matrix, d_list, bounds, x_value
s, 18)
print('8 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

## Part (B) Design a Factor-2 Approximation Algorithm (20 points)

We will design a factor-2 approximation algorithm. To do so, first consider the case when there is just one variable in our problem.

**Example**

$$\begin{aligned} x &\leq 5 \\ -x &\leq -4 \\ x &\leq 3 \\ x &\leq 15 \\ -x &\leq -6 \\ -x &\leq -8 \\ -x &\leq -3 \end{aligned}$$

Let us consider $m$ inequalities involving just one variable $x$.

For convenience, we will drop the upper/lower bounds. In other words, the bounds are simply

$$x \in (-\infty, \infty)\,.$$

(i) Design an algorithm that discovers a value of $x$ guaranteed to satisfy $\lceil \frac{m}{2} \rceil$ or more inequalities.

**Hint** Try satisfying all the inequalities of the form $x \leq d_j$ at once or alternatively, try satisfying all inequalities of the form $-x \leq d_j$ at once.

Write down your approach below (not graded).

Student's answer — Score: 0.0 / 0.0 (Top)

YOUR ANSWER HERE

First, we substitute values of x_i with r_i*x, where r_i are randomly generated. Next, we do a matrix multiplication c_matrix times r = vector(r1 ... rn). We call the result as vector Coefs.

Second, we ddevide both sides of the resulting inequality constraints Coefs*x <= d, element-wise by the absolute value of Coefs. This leaves us with only 1 and -1 in the normalized Coefs.

Third, we partition the constraints based on the normalized Coefs being 1 in subset_1 and being -1 in subset_2.

Next, we evaluate x based on maximizing over the right-hand-side of subset_1; and recover the actual values of x_i by multiplying x by (r*abs(coefs)), element-by-element.

Finally, we count the total number of inequalties statisfied, if it is greater than m/2 we return the recovered x_i values, otherwise we generate another set of random list r and loop until the criteria is satisfied.

Using the algorithm for $n = 1$ variables, implement an algorithm that achieves a factor-2 approximation for inequalities with arbitrary $n$.

Specifically, given a system of $m$ inequalities over $x_1, \ldots, x_n$, provide an algorithm that finds a solution $(x_1, \ldots, x_n)$ that satisfies $\geq \frac{m}{2}$ inequalities.

Once again, we drop the upper/lower bounds on the variables. I.e, $x_i \in (-\infty, \infty)$.

**Hint** Generate $n$ random numbers $r_1, \ldots, r_n$ in some range $[-1, 1]$ and simply substitute:

$$x_1 = r_1 x, x_2 = r_2 x, \ldots, x_n = r_n x$$

where $x$ is a new unknown variable. Use this trick to reduce the constraints involving $n$ variables to a single variable $x$; and apply previous result.

Implement a function `computeApproximateSolution` with inputs

- `n` : number of variables.
- `m` : number of inequalities
- `c_matrix` : a list of list of coefficients of the LHS of inequalities

$$\begin{bmatrix} [c_{11}, \dots, c_{1n}], \\ [c_{21}, \dots, c_{2n}], \\ \dots \\ [c_{m1}, \dots, c_{mn}] \end{bmatrix}$$

  Please note python indexes starting from 0.
- `d_values` : a list of RHS coefficients: $[d_1, \dots, d_m]$

Your function should return a pair: $(k, [x_1, \dots, x_n])$
- The number of inequalities satisfied by your optimal solution ($k$)
- A list denoting the values of $x_1, \dots, x_n$ that satisfy the $k$ inequalities.

Also for this problem, we require $k \geq \frac{m}{2}$.

**Note:** The test cases below will run for large values of $n, m$. If your implementation uses an integer linear programming solver, it may not finish within the time budget of 2 minutes allocated for grading the notebook.

In [9]:

Student's answer (Top)

```
from random import uniform

def computeApproximateSolution(n, m, c_matrix, d_val
ues):
    assert n >= 1
    assert len(c_matrix) == m
    assert all(len(c_list) == n for c_list in c_matr
ix)
    assert len(d_values) == m
    r_values = [uniform(-1, 1) for i in range(n)]
    # your code here
    seed(100001)
    k = 0
    solution_values = []
    attemps = 0
    max_satisfied = 0
    while (max_satisfied < (m / 2)):
        r_values = [uniform(-1, 1) for i in range
(n)]

        coefs = [sum(c_matrix[i][j] * r_values[j] fo
r j in range(n)) for i in range(m)]

        # Normalizing the inequalities to perform pa
rtitions
        coefs_abs = [abs(coef) for coef in coefs]
        normalized_coefs = [coef_val / abs_coef for
coef_val, abs_coef in zip(coefs, coefs_abs)]
        normalized_d_values = [d_val / abs_coef for
d_val, abs_coef in zip(d_values, coefs_abs)]

        # Partition into two subsets
        subset_1 = []
        subset_2 = []
        for i in range(m):
            if normalized_coefs[i] == 1:
                subset_1.append((normalized_coefs
[i], normalized_d_values[i]))
            elif normalized_coefs[i] == -1:
                subset_2.append((normalized_coefs
[i], normalized_d_values[i]))

        x_values = max(abs(normalized_d_values) for
_, normalized_d_values in subset_1)
        k = len(subset_1) if len(subset_1) > len(sub
set_2) else len(subset_2)

        attemps += 1

        # Recover normalized values
        solution_values = [x_values*r_values[i]*coef
s_abs[i] for i in range(n)]

        # Counting constraints satisfied
        max_satisfied = 0
        for (c, d) in zip(c_matrix, d_values):
            if sum([cj * xj for (cj, xj) in zip(c,so
lution_values)]) <= d:
                max_satisfied += 1

    print("attemps:", attemps)

    return k, solution_values
```

In [10]:

Grade cell: `cell-b79fed6cce9d3071` Score: 7.69 / 7.69 (Top)

```
def testSolution(n, m, c_matrix, d_values, x_value
s):
    # always check pre-conditions: saves so much tim
e later
    assert len(c_matrix) == m
    assert all(len(c_list) == n for c_list in c_matr
ix)
    assert len(d_values) == m
    assert len(x_values) == n
```

```
    # Check how many inequalities satisfied
    num_ineqs = 0
    for (c_list, d) in zip(c_matrix, d_values):
        if sum([cj * xj for (cj, xj) in zip(c_list,x
_values )]) <= d+1E-3:
            num_ineqs = num_ineqs + 1
    assert num_ineqs >= m/2, f' Half number of inequ
alities to be sat: {m/2} your solution satisfies: {n
um_ineqs} inequalities '
    print('Test Passed')
    return



n = 5
m = 24
c_matrix = [
    [1, -1, 0, 1, -1],
    [1, 2, 0, 0, 2],
    [-1, 0, 1, 1, 1],
    [1, 0, 0, 0, -1],
    [-1, 0, 0, -1, -1],
    [1, 0, 0, 1, 1],
    [1, 0, -1, 1, 0],
    [0, 2, 1, 0, 2],
    [-1, 1, 1, -1, 0],
    [1, 1, 1, 0, 1],
    [-1, 1, 1, 0, 0],
    [1, 1, 1, 1, 0],
    [-1, 1, 0, 1, -1],
    [1, -2, 0, 0, -2],
    [1, 0, 1, -1, -1],
    [1, 0, 1, 0, 1],
    [-1, 0, 0, 1, 1],
    [-1, 0, 0, 1, 1],
    [1, -1, 1, 1, 1],
    [0, -2, -1, 0, 2],
    [-1, -1, -1, -1, 0],
    [-1, 1, -1, 0, 1],
    [1, 0, 0, 1, 0],
    [-1, 0, -1, 0, -1],
]

d_list = [
    -5, 3, -4, -2, -3, -1,
    -5, 3, -4, -2, -3, -1,
     5, -3, 4, 2, 3, 1,
    5, -3, 4, 2, 3, 1,

]
(k, x_values) = computeApproximateSolution(n, m, c_m
atrix, d_list)
print(k)
print(x_values)

testSolution(n, m, c_matrix, d_list, x_values)
print('5 points')
```

**Congratulations! All test cases in this ce
ll passed.**

In [11]:

Grade cell: `cell-5a890a6c89c4dc37` Score: 23.08 / 23.08 (Top)

```
from random import uniform, randint, seed
## Warning: these are large instances. If your solut
ion takes more than 120 seconds, then
## chances are that you will not receive any credit
for this problem.
def gen_random_instance(n, m):
    c_matrix = [ [randint(-5, 5) for i in range(n)]
for j in range(m)]
    d_values = [randint(-10,10) for i in range(m)]
    return (c_matrix, d_values)

seed(100001)

print('Test # 1')
n = 10
m = 55
(c_matrix, d_values) = gen_random_instance(n, m)
(k, x_values) = computeApproximateSolution(n, m, c_m
atrix, d_values)
print(k)
print(x_values)
testSolution(n, m, c_matrix, d_values, x_values)


print('Test # 2')
n = 35
m = 230
(c_matrix, d_values) = gen_random_instance(n, m)
(k, x_values) = computeApproximateSolution(n, m, c_m
atrix, d_values)
print(k)
print(x_values)
testSolution(n, m, c_matrix, d_values, x_values)

print('Test # 3')
n = 100
m = 550
(c_matrix, d_values) = gen_random_instance(n, m)
(k, x_values) = computeApproximateSolution(n, m, c_m
atrix, d_values)
print(k)
print(x_values)
testSolution(n, m, c_matrix, d_values, x_values)

print('Test # 4')
```

```
print('Test # 4')
n = 80
m = 900
(c_matrix, d_values) = gen_random_instance(n, m)
(k, x_values) = computeApproximateSolution(n, m, c_m
atrix, d_values)
print(k)
print(x_values)
testSolution(n, m, c_matrix, d_values, x_values)

print('Test # 5')
n = 70
m = 445
(c_matrix, d_values) = gen_random_instance(n, m)
(k, x_values) = computeApproximateSolution(n, m, c_m
atrix, d_values)
print(k)
print(x_values)
testSolution(n, m, c_matrix, d_values, x_values)

print('15 points!')
```

```
Congratulations! All test cases in this ce
ll passed.
```

## Selected Answers

### 2(A) part (i)

$$\max \; w_1 + \cdots + w_m$$

### 2(A) part (ii)

$\sum_{i=1}^{n} c_i b_i$ where $b_i = \begin{cases} \ell_i & \text{if } c_i < 0 \\ u_i & \text{if } c_i \geq 0 \end{cases}$.

### 2(A) part (iii)

When $w_j = 1$, we have $\hat{I_j}$ is the inequality

$$c_{j1}x_1 + \cdots + c_{jn}x_n \leq d_j \times 1 + M_j \times 0 = d_j$$

Therefore, $\hat{I_j}$ and $I_j$ coincide.

However, when $w_j = 0$, we have $\hat{I_j}$ is the inequality

$$c_{j1}x_1 + \cdots + c_{jn}x_n \leq d_j \times 0 + M_j \times 1 = M_j$$

However, given that $x_i \in [\ell_i, u_i]$, and $M_j = \text{upperBoundLHS}(I_j)$, it automatically implies that that $\hat{I_j}$ is satisfied trivially by whatever value of $(x_1, \ldots, x_n)$ we choose.

Partition the inequalities into two subsets:

- All inequalities with +1 coefficient for $x$

  $$x \leq d_{i_1}, \ldots, x \leq d_{i_m}$$

  By setting $x = \max(d_{i_1}, \ldots, d_{i_m})$ we can satisfy all these inequalities.
- All inequalities with a $-1$ coefficient for $x$

  $$-x \leq d_{j_1}, \ldots, -x \leq d_{j_m}$$

At least one of these partitions must account for $\lceil \frac{m}{2} \rceil$ or more constraints.

### That's all folks.

```
1.001170084049694, 394.28029795059774, 42.853425696644706, 728.73101067
03014, 7.090395886830324, 225.34416157718724, 63.20738180328399, -93.61
```