15-888 Computational Game Solving (Fall 2025) Released on: Sep. 28, 2025

Homework 1

Student: [*** Your Andrew ID here ***] **Due on: Oct. 10, 2025**

Instructions Submit your homework on Gradescope. Submit a single `pdf` file containing **both** your written solutions **and** your code (`stub.py`) attached to the end (for example, using a `verbatim` environment). You can (but are not required to) typeset your written solutions in the `.tex` file provided in the homework `.zip`.

1 A Regret-based proof of the minimax theorem (20 points)

Let $\mathcal{X} \subseteq \mathbb{R}^{m_1}$ and $\mathcal{Y} \subseteq \mathbb{R}^{m_2}$ be convex and compact sets, and $\mathbf{A} \in \mathbb{R}^{m_1 \times m_2}$. The minimax theorem asserts that

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle = \max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle.$$

In this problem, you will show that the mere existence of (external) regret minimization algorithms is powerful enough to imply the minimax theorem.

One direction (called *weak duality*) of the proof is easy and very general. Specifically, show the following.

Problem 1.1 (5 points)**.** Show that

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle \geq \max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle. \tag{1}$$

Solution. [*** Your solution here ***] □

To show the reverse inequality, we will employ regret minimization to solve the bilinear saddle point problem $\max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle$. At each time t, we will let a regret minimizer $\mathfrak{R}_{\mathcal{Y}}$ pick strategies $\boldsymbol{y}^{(t)} \in \mathcal{Y}$, whereas we will always assume that $\boldsymbol{x}^{(t)} \in \mathcal{X}$ is chosen by the environment to best respond to $\boldsymbol{y}^{(t)}$, that is,

$$\boldsymbol{x}^{(t)} \in \arg\min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle.$$

The utility function observed by $\mathfrak{R}_{\mathcal{Y}}$ at each time t is set to

$$u_{\mathcal{Y}}^t : \boldsymbol{y} \mapsto \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y} \rangle = \langle \boldsymbol{y}, \mathbf{A}^\top \boldsymbol{x}^{(t)} \rangle.$$

We will assume that $\mathfrak{R}_{\mathcal{Y}}$ guarantees *sublinear regret* under any sequence of utilities; we have seen many algorithms with this property, such as `FTRL` and `RM`.

Let $\bar{\boldsymbol{x}}^{(T)} \in \mathcal{X}$ and $\bar{\boldsymbol{y}}^{(T)} \in \mathcal{Y}$ be the average strategies output up to time T, that is,

$$\bar{\boldsymbol{x}}^{(T)} := \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{x}^{(t)} \qquad \bar{\boldsymbol{y}}^{(T)} := \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{y}^{(t)}.$$

Problem 1.2 (5 points). Argue that at all t,

$$\max_{\boldsymbol{y}\in\mathcal{Y}} \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \geq \frac{1}{T} \min_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{T} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(T)}\rangle \geq \frac{1}{T} \sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}^{(t)}\rangle. \tag{2}$$

★ Hint: use the definition of $\bar{\boldsymbol{y}}^{(T)}$. Then, use the information you have on $\boldsymbol{x}^{(t)}$.

Solution. [*** Your solution here ***] □

Problem 1.3 (5 points). Let $\mathsf{Reg}_{\mathcal{Y}}^{(T)}$ be the regret accumulated by $\mathfrak{R}_{\mathcal{Y}}$ up to time T. Using (2), argue that at all times T

$$\max_{\boldsymbol{y}\in\mathcal{Y}} \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \geq \min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle - \frac{\mathsf{Reg}_{\mathcal{Y}}^{(T)}}{T}. \tag{3}$$

★ Hint: use the definition of regret $\mathsf{Reg}_{\mathcal{Y}}^{(T)}$.

Solution. [*** Your solution here ***] □

Problem 1.4 (5 points). Use (3) and (1) and to conclude that

$$\max_{\boldsymbol{y}\in\mathcal{Y}} \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle = \min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle,$$

Solution. [*** Your solution here ***] □

2 Deriving CFR from regret circuits

In this problem, you will derive CFR using regret circuits. For your convenience, we include the pseudocode from Lecture 5 for the subroutine that computes the counterfactual utilities (Algorithm 1).

Recall that the sequence-form strategy set can be decomposed as follows. First, in any observation point $k \in \mathcal{K}$ with a set of children $\{p_1, \dots, p_\nu\} = \{\rho(k, s) : s \in \mathcal{S}_k\}$,

$$\mathcal{X}_k = \mathcal{X}_{p_1} \times \mathcal{X}_{p_2} \times \cdots \times \mathcal{X}_{p_\nu}, \tag{4}$$

where $\mathcal{X}_{p_i}$ is the sequence-form polytope corresponding to the subtree rooted at point p_i. Second, in any decision point $j \in \mathcal{J}$ with a set of children $\{p_1, \dots, p_\nu\} = \{\rho(j, a) : a \in \mathcal{A}_j\}$,

$$\mathcal{X}_j \coloneqq \mathrm{co}\left\{ \begin{pmatrix} \boldsymbol{e}_1 \\ \mathcal{X}_{p_1} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix}, \begin{pmatrix} \boldsymbol{e}_2 \\ \mathbf{0} \\ \mathcal{X}_{p_2} \\ \vdots \\ \mathbf{0} \end{pmatrix}, \dots, \begin{pmatrix} \boldsymbol{e}_\nu \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathcal{X}_{p_\nu} \end{pmatrix} \right\}, \tag{5}$$

where $\boldsymbol{e}_i \in \mathbb{R}^{\mathcal{A}_j}$ is the ith unit vector.

You will prove soundness of `CFR` by induction, proceeding in a bottom-up fashion.

Algorithm 1: Counterfactual regret minimization (CFR)

Input: A regret minimizer $\mathfrak{R}_j$ for each decision point $j \in \mathcal{J}$ of the tree-form decision process.

ObserveUtility($\boldsymbol{u}^{(t)} \in \mathbb{R}^{\Sigma}$):
  $V^{(t)}[\perp] := 0$;
  for *each node in the tree* $p \in \mathcal{J} \cup \mathcal{K}$ *in bottom-up order* **do**
    if $p \in \mathcal{J}$ **then**
      Let $j = v$;
      $V^{(t)}[j] := \sum_{a \in \mathcal{A}_j} b_j^{(t)}[a] \cdot (\boldsymbol{u}^{(t)}[(j,a)] + V^{(t)}[\rho(j,a)])$;
    else
      Let $k = p$;
      $V^{(t)}[k] := \sum_{s \in \mathcal{S}_k} V^{(t)}[\rho(k,s)]$;
  for *each decision point* $j \in \mathcal{J}$ **do**
    for *each action* $a \in \mathcal{A}_j$ **do**
      $\boldsymbol{u}_j^{(t)}[a] := \boldsymbol{u}^{(t)}[(j,a)] + V^{(t)}[\rho(j,a)]$;
    $\mathfrak{R}_j$.ObserveUtility($\boldsymbol{u}_j^{(t)}$);

Problem 2.1 (5 points). Argue that the utility at every terminal decision point $j \in \mathcal{J}$ is given by $\boldsymbol{u}_j^{(t)}[a] = \boldsymbol{u}^{(t)}[(j,a)]$ for all $a \in \mathcal{A}_j$.

Solution. [*** Your solution here ***] □

Problem 2.2 (5 points). Apply the regret circuit for the Cartesian product in each observation node $p \in \mathcal{K}$. Derive the utility vectors given as input to the children of p.

★ Hint: Use the decomposition given in (4).

Solution. [*** Your solution here ***] □

Problem 2.3 (10 points). Apply the regret circuit for the convex hull in each decision node $p \in \mathcal{J}$. Derive the utility vector given as input to the regret minimizer $\mathfrak{R}_j$ that is responsible for mixing over $\mathcal{A}_j$, and show that it matches the utilities specified in Algorithm 1.

★ Hint: Use the decomposition given in (5).

Solution. [*** Your solution here ***] □

Problem 2.4 (10 points). Show that the regret of CFR is bounded by $\sum_{j \in \mathcal{J}} \max\{0, \mathsf{Reg}_j^{(T)}\}$, where $\mathsf{Reg}_j^{(T)}$ is the regret of $\mathfrak{R}_j$. What is the regret of CFR when each $\mathfrak{R}_j$ is instantiated using MWU? What is the

regret of `CFR` when each $\mathfrak{R}_j$ is instantiated using `RM`?

★ Hint: Recall that the regret bound depends on the norm of the utility; argue about how large that norm can be.

Solution. [*** Your solution here ***] □

3 Linear programming for solving extensive-form zero-sum games, and application to low randomization in poker (50 points)

In many games, the optimal Nash equilibrium requires that all players randomize their moves. As an example, consider rock-paper-scissors: any deterministic strategy (for example, always playing rock) is heavily suboptimal. In this problem, you will quantify how much value is lost by insisting on playing deterministic strategies in three games: rock-paper-superscissors and two well-known poker variants—Kuhn poker [Kuhn, 1950] and Leduc poker [Southey et al., 2005]. A description of each game is given in the zip of this homework. The description is specified in Section 3.1. The zip of the homework also contains a stub Python file to help you set up your implementation.

3.1 Format of the game files

Each game is encoded as a json file with the following structure.

- At the root, we have a dictionary with three keys: `decision_problem_pl1`, `decision_problem_pl2`, and `utility_pl1`. The first two keys contain a description of the tree-form sequential decision problems faced by the two players, while the third is a description of the bilinear utility function for Player 1 as a function of the sequence-form strategies of each player. Since both games are zero-sum, the utility for Player 2 is the opposite of the utility of Player 1.
- The tree of decision points and observation points for each decision problem is stored as a list of nodes. Each node has the following fields:

  `id` is a string that represents the identifier of the node. The identifier is unique among the nodes for the same player.

  `type` is a string with value either `decision` (for decision points) or `observation` (for observation points).

  `actions` (only for decision points). This is a set of strings, representing the actions available at the decision node.

  `signals` (only for observation points). This is a set of strings, representing the signals that can be observed at the observation node.

  `parent_edge` identifies the parent edge of the node. If the node is the root of the tree, then it is `null`. Else, it is a pair `(parent_node_id, action_or_signal)`, where the first member is the `id` of the parent node, and `action_or_signal` is the action or signal that connects the node to its parent.

  `parent_sequence` (only for decision points). Identifies the parent sequence p_j of the decision point, defined as the last sequence (that is, decision point-action pair) encountered on the path from the root of the decision process to j.

Remark 1. The list of nodes of the tree-form sequential decision process is given in top-down traversal order. The bottom-up traversal order can be obtained by reading the list of nodes

backwards.

- The bilinear utility function for Player 1 is given through the payoff matrix $\mathbf{A}$ such that the (expected) utility of Player 1 can be written as

$$u_1(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle,$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are sequence-form strategies for Players 1 and 2 respectively. We represent $\mathbf{A}$ in the file as a list of all non-zero matrix entries, storing for each the row index, column index, and value. Specifically, each entry is an object with the fields

`sequence_pl1` is a pair `(decision_pt_id_pl1, action_pl1)` which represents the sequence of Player 1 (row of the entry in the matrix).

`sequence_pl2` is a pair `(decision_pt_id_pl2, action_pl2)` which represents the sequence of Player 2 (column of the entry in the matrix).

`value` is the non-zero float value of the matrix entry.

Example: Rock-paper-superscissors In the case of rock-paper-superscissors the decision problem faced by each of the players has only one decision points with three actions: playing rock, paper, or superscissors. So, each tree-form sequential decision process only has a single node, which is a decision node. The payoff matrix of the game is

$$\begin{array}{c} \\ \mathrm{r} \\ \mathrm{p} \\ \mathrm{s} \end{array} \begin{array}{c} \begin{array}{ccc} \mathrm{r} & \mathrm{p} & \mathrm{s} \end{array} \\ \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -2 \\ -1 & 2 & 0 \end{pmatrix} \end{array}.$$

So, the game file in this case has content:

```
{
  "decision_problem_pl1": [
    {"id": "d1_pl1", "type": "decision", "actions": ["r", "p", "s"],
     "parent_edge": null, "parent_sequence": null}
  ],
  "decision_problem_pl2": [
    {"id": "d1_pl2", "type": "decision", "actions": ["r", "p", "s"],
     "parent_edge": null, "parent_sequence": null}
  ],
  "utility_pl1": [
    {"sequence_pl1": ["d1_pl1", "r"], "sequence_pl2": ["d1_pl2", "p"], "value": -1},
    {"sequence_pl1": ["d1_pl1", "r"], "sequence_pl2": ["d1_pl2", "s"], "value": 1},
    {"sequence_pl1": ["d1_pl1", "p"], "sequence_pl2": ["d1_pl2", "r"], "value": 1},
    {"sequence_pl1": ["d1_pl1", "p"], "sequence_pl2": ["d1_pl2", "s"], "value": -2},
    {"sequence_pl1": ["d1_pl1", "s"], "sequence_pl2": ["d1_pl2", "r"], "value": -1},
    {"sequence_pl1": ["d1_pl1", "s"], "sequence_pl2": ["d1_pl2", "p"], "value": 2}
  ]
}
```

3.2 Computing the value of the game (15 points)

As a warm up, you will implement the linear program formulation of Nash equilibrium strategies seen in Lecture 5 using the commercial solver Gurobi (https://www.gurobi.com/). Gurobi is a powerful commercial solver for linear and non-linear optimization problems. You can download the solver and request a free license for academic use from their website.

Installing `gurobipy` Installation instructions for Gurobi's python bindings are available on the Gurobi website, here.[1]

Linear programming formulation of Nash equilibrium strategies For your convenience, here are again the linear programs—for Player 1 and Player 2, respectively—that you need to implement:

$$\mathcal{P}_1 : \begin{cases} \max & \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{s.t.} & \textcircled{1}\ \mathbf{A}^\top \boldsymbol{x} - \mathbf{F}_2^\top \boldsymbol{v} \geq \mathbf{0} \\ & \textcircled{2}\ \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ & \textcircled{3}\ \boldsymbol{x} \geq \mathbf{0},\ \boldsymbol{v} \text{ free}, \end{cases} \qquad \mathcal{P}_2 : \begin{cases} \max & \boldsymbol{f}_1^\top \boldsymbol{v} \\ \text{s.t.} & \textcircled{1}\ -\mathbf{A}\boldsymbol{y} - \mathbf{F}_1^\top \boldsymbol{v} \geq \mathbf{0} \\ & \textcircled{2}\ \mathbf{F}_2 \boldsymbol{y} = \boldsymbol{f}_2 \\ & \textcircled{3}\ \boldsymbol{y} \geq \mathbf{0},\ \boldsymbol{v} \text{ free}, \end{cases} \tag{6}$$

where $\{\boldsymbol{x} \in \mathbb{R}^{|\Sigma_1|} : \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1, \boldsymbol{x} \geq \mathbf{0}\}$ and $\{\boldsymbol{y} \in \mathbb{R}^{|\Sigma_2|} : \mathbf{F}_2 \boldsymbol{y} = \boldsymbol{f}_2, \boldsymbol{y} \geq \mathbf{0}\}$ are the sequence-form polytopes of the two players, and $\mathbf{A}$ is the payoff matrix for Player 1. Conveniently, the objective values of $\mathcal{P}_1$ and $\mathcal{P}_2$ will be the exact expected utility that each player can secure by playing against a perfectly rational opponent. Since all games are zero sum, the objective values of $\mathcal{P}_1$ and $\mathcal{P}_2$ will sum to 0 (if they don't, you must have a bug somewhere).

Problem 3.1 (15 points). Implement the linear program for computing Nash equilibrium strategies for both Player 1 and Player 2.

For each of the three games (rock-paper-superscissors, Kuhn poker, and Leduc poker), and for each of the two player, report Gurobi's output.

★ Hint: make sure to take a look at the "Gurobi tips and tricks" at the end of this document. It includes some tips as to how to debug common issues.
★ Hint: start from rock-paper-superscissors, and only then move to the more complex games.
★ Hint: since all games are zero-sum, the objective values of $\mathcal{P}_1$ and $\mathcal{P}_2$ must sum to 0.
★ Hint: the objective value for $\mathcal{P}_1$ should be 0 for rock-paper-superscissors, -0.0555 for Kuhn poker, and -0.0856 for Leduc poker.

Solution. [*** Your solution here. You should include six Gurobi outputs (3 games, 2 players per game). Feel free to use the `verbatim` environment in Latex to simply dump the output. Make sure to specify what game and what player each Gurobi output refers to. Don't forget to include your code at the end of your homework. For example, your output in the case of rock-paper-superscissors for Player 1 should look roughly like this ***]

```
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (linux64)
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads
Optimize a model with 4 rows, 4 columns and 12 nonzeros
Model fingerprint: 0x5264c0a3
Coefficient statistics:
    Matrix range     [1e+00, 2e+00]
    Objective range  [1e+00, 1e+00]
    Bounds range     [0e+00, 0e+00]
    RHS range        [1e+00, 1e+00]
Presolve removed 1 rows and 0 columns
Presolve time: 0.01s
Presolved: 3 rows, 4 columns, 11 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
        0    1.0000000e+00   1.000000e+00   0.000000e+00      0s
```

[1] https://www.gurobi.com/documentation/9.1/quickstart_linux/cs_python.html#section:Python

```
         2    -0.0000000e+00    0.000000e+00    0.000000e+00       0s

Solved in 2 iterations and 0.01 seconds
Optimal objective -0.000000000e+00
```

□

3.3 Computing optimal deterministic strategies (15 points)

In this subsection we study how much worse each player is if they (but not the opponent) are restricted to playing deterministic strategies only. To model this, we will add a constraint saying that all entries of the sequence-form strategy vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in (6) can only take values in $\{0,1\}$. The resulting *integer* programs—which we call $\tilde{\mathcal{P}}_1$ and $\tilde{\mathcal{P}}_2$—are given next.

$$\tilde{\mathcal{P}}_1 : \begin{cases} \max & \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{s.t.} & ① \ \mathbf{A}^\top \boldsymbol{x} - \mathbf{F}_2^\top \boldsymbol{v} \geq \mathbf{0} \\ & ② \ \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ & ③ \ \boldsymbol{x} \in \{0,1\}^{|\Sigma_1|}, \ \boldsymbol{v} \text{ free}, \end{cases} \qquad \tilde{\mathcal{P}}_2 : \begin{cases} \max & \boldsymbol{f}_1^\top \boldsymbol{v} \\ \text{s.t.} & ① \ -\mathbf{A}\boldsymbol{y} - \mathbf{F}_1^\top \boldsymbol{v} \geq \mathbf{0} \\ & ② \ \mathbf{F}_2 \boldsymbol{y} = \boldsymbol{f}_2 \\ & ③ \ \boldsymbol{y} \in \{0,1\}^{|\Sigma_2|}, \ \boldsymbol{v} \text{ free}. \end{cases} \tag{7}$$

Problem 3.2 (15 points). Implement the integer programs given in (7) for computing optimal deterministic strategies for both Player 1 and Player 2.

For each of the three games (rock-paper-superscissors, Kuhn poker, and Leduc poker), and for each of the two player, report Gurobi's output.

★ Hint: make sure to take a look at the "Gurobi tips and tricks" at the end of this document. It includes some tips as to how to debug common issues.
★ Hint: start from rock-paper-superscissors, and only then move to the more complex games.
★ Hint: here there are *no guarantees* that the value of $\tilde{\mathcal{P}}_1$ and the value of $\tilde{\mathcal{P}}_2$ sum to 0 anymore! In fact, that will be false in all games.

Solution. [*** Your solution here. You should include six Gurobi outputs (3 games, 2 players per game). Feel free to use the `verbatim` environment in Latex to simply dump the output. Make sure to specify what game and what player each Gurobi output refers to. Don't forget to include your code at the end of your homework. For example, your output in the case of Kuhn poker for Player 1 should look roughly like this ***]

```
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (linux64)
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads
Optimize a model with 18 rows, 18 columns and 57 nonzeros
Model fingerprint: 0x57532587
Variable types: 6 continuous, 12 integer (12 binary)
Coefficient statistics:
    Matrix range     [2e-01, 1e+00]
    Objective range  [1e+00, 1e+00]
    Bounds range     [1e+00, 1e+00]
    RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.3333335
Presolve removed 11 rows and 10 columns
Presolve time: 0.00s
Presolved: 7 rows, 8 columns, 22 nonzeros
Found heuristic solution: objective -0.3333333
```

```
Variable types: 0 continuous, 8 integer (5 binary)

Root relaxation: objective -5.555556e-02, 9 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   -0.05556    0    3   -0.33333   -0.05556  83.3%     -    0s
H    0     0                      -0.1666667   -0.05556  66.7%     -    0s
     0     0   -0.05556    0    3   -0.16667   -0.05556  66.7%     -    0s

Explored 1 nodes (9 simplex iterations) in 0.00 seconds
Thread count was 16 (of 16 available processors)

Solution count 3: -0.166667 -0.333333 -0.333333
No other solutions better than -0.166667

Optimal solution found (tolerance 1.00e-04)
Best objective -1.666666666667e-01, best bound -1.666666666667e-01, gap 0.0000%
```

□

3.4 Controlling the amount of determinism (20 points)

In Problem 3.1 no determinism constraint was present. At the other extreme, in Problem 3.2 we insisted that at all decision points a deterministic strategy be followed. In this last subsection we will explore intermediate cases: for each value of k, we will study how much value each player can secure if they are constrained to play deterministically in at least k decision points. When $k = 0$, we will recover the objective values seen in Problem 3.1. When k is equal to the number of decision points of the player in the game, we will recover the objective values seen in Problem 3.2.

Integer programming model An optimal strategy for Player 1 subject to the constraint that at least k decision points must prescribe a deterministic strategy can be obtained as the solution to the integer linear program $\mathcal{P}_1(k)$ given in (8). Understanding the details is not important for this problem, though we included a description of the meaning of each constraint just in case you are curious.

$$\mathcal{P}_1(k): \begin{cases} \max & \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{s.t.} & \textcircled{1}\ \mathbf{A}^\top \boldsymbol{x} - \mathbf{F}_2^\top \boldsymbol{v} \geq \mathbf{0} \\ & \textcircled{2}\ \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ & \textcircled{3}\ \boldsymbol{x}[(j,a)] \geq \boldsymbol{z}[(j,a)] & \forall j \in \mathcal{J}_1 : p_j = \varnothing,\ \ a \in \mathcal{A}_j \\ & \textcircled{4}\ \boldsymbol{x}[(j,a)] \geq \boldsymbol{x}[p_j] + \boldsymbol{z}[(j,a)] - 1 & \forall j \in \mathcal{J}_1 : p_j \neq \varnothing,\ \ a \in \mathcal{A}_j \\ & \textcircled{5}\ \sum_{a \in \mathcal{A}_j} \boldsymbol{z}[(j,a)] \leq 1 & \forall j \in \mathcal{J}_1 \\ & \textcircled{6}\ \sum_{j \in \mathcal{J}_1} \sum_{a \in \mathcal{A}_j} \boldsymbol{z}[(j,a)] \geq k \\ & \textcircled{7}\ \boldsymbol{x} \geq \mathbf{0},\ \ \boldsymbol{z} \in \{0,1\}^{|\Sigma_1|},\ \ \boldsymbol{v} \text{ free}, \end{cases} \tag{8}$$

- $\boldsymbol{z} \in \{0,1\}^{|\Sigma_1|}$ is a binary vector that decides, for each strategy $(j,a) \in \Sigma_1$ of Player 1, whether to pick action a at j with probability 1. Since the strategy vector $\boldsymbol{x}$ is expressed in sequence-form, picking action a with probability 1 at j is expressed through constraints ③ and ④.
- Constraint ⑤ asserts that no more than one action at each decision point can be forced to be played with probability 1.
- Constraint ⑥ asserts that in at least k decision point, exactly one of the actions will be forced to be played with probability 1.

The integer program $\mathcal{P}_2(k)$ for Player 2 is analogous.

Problem 3.3 (15 points). Implement the integer linear programs $\mathcal{P}_1(k)$ and $\mathcal{P}_2(k)$, described above, for computing optimal strategies with a given lower bound on the amount of determinism.

For each of the three games (rock-paper-superscissors, Kuhn poker, and Leduc poker), and for each of the two player i, plot the objective value of $\mathcal{P}_i(k)$ as a function of $k \in \{0,\dots,|\mathcal{J}_i|\}$ (number of decision points of Player i).

★ Hint: make sure to take a look at the "Gurobi tips and tricks" at the end of this document. It includes some tips as to how to debug common issues.
★ Hint: Gurobi can be pretty verbose by default. For this problem, if you would like to silence Gurobi you can use `m.setParam("OutputFlag", 0)`
★ Hint: For Leduc poker, if Gurobi is taking too long to optimize when k is large, you can lower the solution precision by calling `m.setParam("MIPGap", 0.01)` before `m.optimize()`. Expect a runtime of up to one-five hours for Leduc poker, depending on how powerful the machine you are using is.

Solution. [*** Your solution here. You should include six plots (3 games, 2 players per game). Make sure to specify what game and what player each plot refers to. Don't forget to include your code at the end of your homework. ***] □

Problem 3.4 (5 points). Comment on the results you obtained in this problem: do highly-deterministic strategies for the three small games exist? Are the results what you expected? If yes, what did he results confirm? If not, how do you think you can reconcile your previous intuition with the experimental findings?

Solution. [*** Your solution here ***] □

A Appendix: Gurobi tips and tricks

Basic notation Let `m` denote the Gurobi model object. Then, here is a quick cookbook.

- Add a continuous free variable:
  `m.addVar(-GRB.INFINITY, GRB.INFINITY, vtype=GRB.CONTINUOUS, name="human_var_name_here")`
- Add a continuous nonnegative variable:
  `m.addVar(0.0, GRB.INFINITY, vtype=GRB.CONTINUOUS, name="human_var_name_here")`
- Add a binary variable:
  `m.addVar(0.0, 1.0, vtype=GRB.BINARY, name="human_var_name_here")`

- Add an equality constraint:
  `m.addConstr(lhs == rhs)`
- Add an inequality ($\geq$) constraint:
  `m.addConstr(lhs >= rhs)`
- Set a maximization objective:
  `m.setObjective(obj, sense=GRB.MAXIMIZE)`

Accessing the solution After calling `m.optimize()`, you can obtain the objective value by calling

```
m.getAttr(GRB.Attr.ObjVal)
```

If you want to inspect the solution, given a variable object `v` (the object returned by `m.addVar`), you can access the value of `v` in the current solution by calling

```
v.getAttr(GRB.Attr.X)
```

Troubleshooting First of all, if you are having a problem with Gurobi, the first thing you should try to do is to ask Gurobi to dump the model that it thinks you are asking to solve to a file in a human readable format. *Reading the model file will be so much easier if you gave names to the variables in your model, using the 'name' optional argument of `addVar`.*

To have Gurobi dump the model, you can use something like this:

```
m.write("/tmp/model.lp")
```

Of course, you can specify a different path. However, it is important that you keep the '`.lp`' extension: there are multiple format that Gurobi can output, and it uses the file extension to guess which format you want.

Beyond the general rule of thumb above, make sure of the following:

- Start from rock-paper-superscissors. There, the `/tmp/model.lp` file for Player 1 for Problem 3.1 should look something like this (probably with different variable names):

  ```
  \ Model game_value_pl1
  \ LP format - for model browsing. Use MPS format to capture full model detail.
  Maximize
      v[d1_pl2]
  Subject To
      R0: x[('d1_pl1','p')] - x[('d1_pl1','s')] - v[d1_pl2] >= 0
      R1: - x[('d1_pl1','r')] + 2 x[('d1_pl1','s')] - v[d1_pl2] >= 0
      R2: x[('d1_pl1','r')] - 2 x[('d1_pl1','p')] - v[d1_pl2] >= 0
      R3: x[('d1_pl1','r')] + x[('d1_pl1','p')] + x[('d1_pl1','s')] = 1
  Bounds
      v[d1_pl2] free
  End
  ```

  Note: Gurobi omits listing nonnegative variables in the `Bounds` section.

- Did you remember to specify that you want a *maximization* problem? (Gurobi's default is minimization) If Gurobi says that the model is unbounded, chances are you forgot.
- Check that the number of variables and constraints is what you expect. Are the sense of the constraints (equality, $\leq$, $\geq$) what you wanted?

References

H. W. Kuhn. A simplified two-person poker. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1 of *Annals of Mathematics Studies, 24*, pages 97–103. Princeton University Press, Princeton, New Jersey, 1950.

Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: opponent modelling in poker. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 550–558, 2005.