MGT-483 : Optimal Decision Making Project

Guillaume Schneider 296488
Alessandro Dalbesio 352298
Nicolas Termote 301581

21 May 2023

Contents

1 Sudokus **2**
1.1 Problem definition . . . 2
1.2 Problem implementation . . . 3
1.3 Results . . . 4

2 An Optimal Strategy for Poker **5**
2.1 Standard Rock Paper Scissors game . . . 5
2.1.1 Payoff matrix . . . 5
2.1.2 Best strategy assuming that the row-player always plays rock . . . 5
2.1.3 Rock-paper-scissors game solution . . . 7
2.2 Modified rock-paper-scissors game . . . 8
2.2.1 Payoff matrix . . . 8
2.2.2 Modified Rock-paper-scissors game solution . . . 8
2.3 Khun-Poker . . . 9
2.3.1 Static and dynamic games . . . 9
2.3.2 Nash Strategy . . . 9
2.3.3 Nash strategy of a specific case . . . 10

3 Morphing Images via Optimal Transport **12**
3.1 Formulation of the problem . . . 12
3.2 Demonstration of the linearity of the problem . . . 13
3.3 Redundancy of the constraint $\sum_{i=1}^{d} c_i = 1$ . . . 14
3.4 Implementation . . . 14
3.5 Results . . . 15

Chapter 1

Sudokus

1.1 Problem definition

To solve our Sudoku problem, each cell (i, j) = (row, column) of the board is defined as a binary decision vector $b_{(i,j)} \in \{0, 1\}^9$ such that $b_{(i,j),k} = \begin{cases} 1 & \text{if the cell contains the number} \\ 0 & \text{otherwise} \end{cases}$ with $k \in \{1, ...9\}$.

After defining our variables we need to define our constraints that are:

- Each row contains mush contains each number exactly once.

$$\sum_{j=1}^{9} b_{(i,j),k} = 1 \ \ \forall (i, k)$$

- Each column contains must contains each number exactly once

$$\sum_{i=1}^{9} b_{(i,j),k} = 1 \ \ \forall (j, k)$$

- Each 3×3 sub-grid must contains each number exactly once

$$\sum_{i=p}^{p+2} \sum_{j=q}^{q+2} b_{(i,j),k} = 1 \ \ \forall k \text{ and } p, q \in \{1, 4, 7\}$$

- Each cell must contains exactly one number.

$$\sum_{k=1}^{9} b_{(i,j),k} = 1 \quad \forall (i, j)$$

- Some cells (let's say n) have a fixed value

$$b_{(i,j),k} = m \ \text{ with } \ m \in \{m_1, ..., m_n\}$$

Since we are solving a feasibility problem we choose as objective function a constant value (since in this case we only care about finding a solution that fulfills the constraints).

Our linear program is:

$$\begin{aligned}
&\min 0 \\
&\text{s.t.} \sum_{j=1}^{9} b_{(i,j),k} = 1 \ \ \forall (i,k) \\
&\sum_{i=1}^{9} b_{(i,j),k} = 1 \ \ \forall (j,k) \\
&\sum_{i=p}^{p+3} \sum_{j=q}^{q+3} b_{(i,j),k} = 1 \ \ \forall k \text{ and } p,q \in \{0,3,6\} \\
&\sum_{k=1}^{9} b_{(i,j),k} = 1 \quad \forall (i,j) \\
&b_{(i,j),k} = m \ \ \forall m \in \{m_1, ..., m_n\}
\end{aligned}$$

1.2 Problem implementation

To implement our linear program we are going to use python and the library CVXPY.

The main part of our code is:

```
# Define the binary decision variables on a flattened sudoku board
b = cp.Variable((9*9, 9), boolean=True)

# Define the constraints
constraints = sudoku_contraints(b)

# Define the objective
obj = cp.Minimize(0)

# Define the problem
prob = cp.Problem(obj, constraints)

# Solve the problem
prob.solve()
```

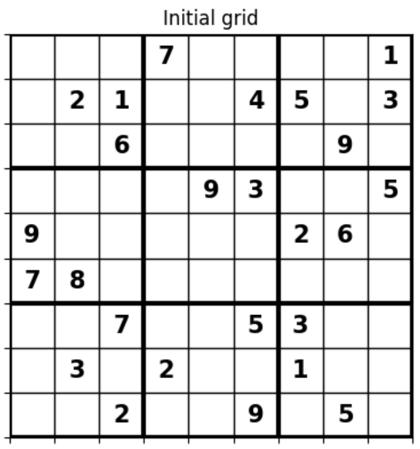
where:

- Firstly we define the binary decision variables.
- Then we define the constraints with the function `sudoku_constraints(b)` and the objective.
- Finally we define our problem and we solve it.

To be able to implement our constraints in `sudoku_constaints(b)` we have also defined an utility function `flattened_square_indices(i,j,size=3,square_shape=9)` that converts our data in a form compatible with cvxpy.

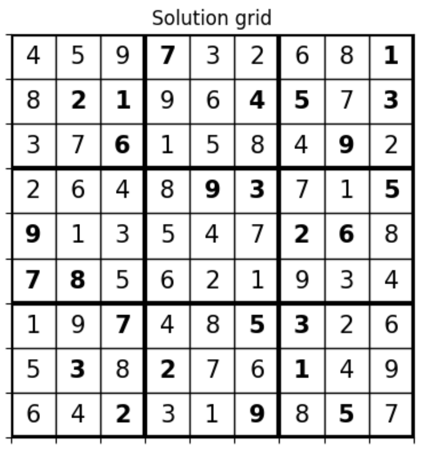
1.3 Results

The initial grid with the given values and the solution grid are displayed below.



Initial grid

			7					**1**
	2	**1**			**4**	**5**		**3**
		6					**9**	
				9	**3**			**5**
9						**2**	**6**	
7	**8**							
		7			**5**	**3**		
	3		**2**			**1**		
		2			**9**		**5**	



Solution grid

4	5	9	**7**	3	2	6	8	**1**
8	**2**	**1**	9	6	**4**	**5**	7	**3**
3	7	**6**	1	5	8	4	**9**	2
2	6	4	8	**9**	**3**	7	1	**5**
9	1	3	5	4	7	**2**	**6**	8
7	**8**	5	6	2	1	9	3	4
1	9	**7**	4	8	**5**	**3**	2	6
5	**3**	8	**2**	7	6	**1**	4	9
6	4	**2**	3	1	**9**	8	**5**	7

Chapter 2

An Optimal Strategy for Poker

2.1 Standard Rock Paper Scissors game

2.1.1 Payoff matrix

In the case of the standard rock, paper and scissor game the payoff matrix is:

Row p. \ Column p.	Rock	Paper	Scissors
Rock	0	-1	1
Paper	1	0	-1
Scissors	-1	1	0

$$\implies M = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

2.1.2 Best strategy assuming that the row-player always plays rock

Best strategy obtained by logical reasoning If we are sure that the row player is going to always play rock ($P_{rock} = 1$) then we should always play paper ($P_{paper} = 1$) in order to win.

Best strategy obtained by solving a linear program Since we want to know the best strategy for the column player, we can either use the negative version of the payoff matrix (-M) and maximize the payoff, or we can use the positive version of the payoff matrix (M) and minimise the payoff of the row player.

If we choose the second option we should solve the following linear program:

$$\min_{q \in \Delta} p^T M q$$

with $p = (1, 0, 0)^T$ being the strategy of the row player for playing only rock.

If we solve the linear program we get that the vector q is, as expected, $q = (0, 1, 0)^T$ and so the column player should choose to play paper with a probability $P = 1$ to maximize his payoff.
The payoff for the row player is -1 and this implies that for the column player the payoff is 1.

Nash strategy for the row player

The Nash strategy for the row-player can be found by solving the min-max problem.

$$\max_{p\in\Delta}\left(\min_{q\in\Delta} p^\top Mq\right)$$

In order to have a single linear program, whose solution yields a Nash strategy for the row-player, we can reformulate the inner minimization problem as a maximization problem by using duality.

The inner min problem is in the form

$$\begin{aligned} \min_{q\in\Delta}\ & c^\top q \\ \text{s.t.}\ & Aq = b \\ & q_i \geq 0 \end{aligned}$$

with $c^T = p^T$, $A = [1\ 1\ 1]$ and $b = 1$.

Its corresponding dual problem is

$$\begin{aligned} \max\ & a \\ \text{s.t.}\ & a \leq (p^\top M)_i^\top \quad \forall i \\ & a \text{ free} \end{aligned}$$

Combining this result with the maximization problem over p gives a single linear program :

$$\begin{aligned} \max\ & a \\ \text{s.t.}\ & a \leq (p^\top M)_i^\top \quad \forall i \\ & Ap = b \\ & p_i \geq 0 \\ & a \text{ free} \end{aligned}$$

with $A = [1\ 1\ 1]$ and $b = 1$.

Limits of the expected payoff

Let's assume we have a two-player zero-sum game, where the first player plays following the Nash strategy while the second player's strategy can vary.
The goal here is to show that the expected payoff for the first player cannot be lower than the optimal value of the min-max problem when following the Nash strategy, regardless of the second player's strategy.

The Nash strategy for the first player is a strategy that maximizes its own payoff under the assumption that the other is playing optimally.
The min-max problem seeks to find the strategy for the first player that maximizes their minimum possible payoff.
Since the minimum possible payoff is obtained in the worst case scenario (so when the other player is playing optimally) the strategy given by the min-max problem is the same as the Nash strategy.

This implies that if the first player follows the Nash strategy when the other player is playing optimally it will have a payoff equals to the one obtained by solving the the min-max problem.

If we are not in the worst case then the payoff of the first player, by playing with the Nash strategy, will be equals or higher than the one obtained in the worst case scenario but not smaller.

For this reason if the first player plays following the Nash strategy then its payoff cannot be lower than the one given by the max-min optimization problem regardless of the second player's strategy.

Nash strategy of the column-player

The Nash strategy of the column-player, given that the row-player is playing rock with a probability equals to 1 is:

Move	**Probability**
Rock	0
Paper	1
Scissor	0

The expected payoff for the column-player is 1.

2.1.3 Rock-paper-scissors game solution

Column player Nash strategy

The Nash strategy for the column-player is:

Move	**Probability**
Rock	$0.33\bar{3}$
Paper	$0.33\bar{3}$
Scissor	$0.33\bar{3}$

The expected payoff for the column-player is 0.

Row player Nash strategy

The Nash strategy for the row-player is:

Move	**Probability**
Rock	$0.33\bar{3}$
Paper	$0.33\bar{3}$
Scissor	$0.33\bar{3}$

The expected payoff for the column-player is 0.

Remarks

We can see that the expected payoff is zero for both players and this implies that the game is fare. In this case for the Nash strategy each player should choose randomly at each move between rock, paper and scissors with the same probability to maximize their payoff.

We can see that this is indeed the best strategy for both of them because:

- If any player has an higher probability of choosing one move (e.g. rock) the other player would increase the probability of choosing the move that makes it maximize its payoff (e.g. paper)
- Then the first player would change its strategy by choosing with an higher probability the move that makes it maximize its payoff (e.g. scissors)

This behavior continues until we reach a point where each move is chosen with the same probability (this maximizes the payoff of both players)

2.2 Modified rock-paper-scissors game

In this case we are going to consider a modified rock-paper-scissors game, where the payoff of the row-player amounts to +2 instead of +1 if she wins by playing rock.

2.2.1 Payoff matrix

Row p. \ Column p.	Rock	Paper	Scissors
Rock	0	-1	2
Paper	1	0	-1
Scissors	-1	1	0

$$\Longrightarrow M = \begin{pmatrix} 0 & -1 & 2 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

2.2.2 Modified Rock-paper-scissors game solution

Row Player Nash strategy

The Nash strategy for the row-player is:

Move	**Probability**
Rock	0.250
Paper	0.417
Scissor	$0.33\bar{3}$

The expected payoff for the row-player is 0.083.

Column player Nash strategy

The Nash strategy for the column-player is:

Move	**Probability**
Rock	$0.33\bar{3}$
Paper	0.417
Scissor	0.250

The expected payoff for the column-player is -0.083.

2.3 Khun-Poker

2.3.1 Static and dynamic games

Kuhn Poker is a dynamic game, that is, the players need to select multiple actions in sequence, reacting to each other's actions. To reformulate the game as an instance of Rock-Paper-Scisors, we firstly need to express it as a static game, where all actions are chosen simultaneously in advance (as a contingency plan).
It is possible since the strategy is determined from the beginning of the game: each player knows his card from the beginning and can decide what will be his actions for all situations.

2.3.2 Nash Strategy

Since the game is now static we can use the same optimization equations used in the rock-paper scissor game with only the dimensional difference: p is now a 1×27 vector and q is a 1×64 vector.

The linear program can therefore be written as:

$$\begin{aligned} \max a & \\ \text{s.t. } & a \leq (p^\top M)_i^\top \quad \forall i \\ & Ap = b \\ & a \text{ free} \\ & p_i \geq 0 \end{aligned}$$

with $A = [1\ 1\ ...\ 1]_{1\times 27}$ and $b = 1$

Row player Nash strategy

The **row** player has an expected payoff of -0.056, and a Nash equilibrium of :

p_0 : 0.0	p_1 : 0.184	p_2 : 0.0	p_3 : 0.183	p_4 : 0.0	p_5 : 0.0
p_6 : 0.0	p_7 : 0.0	p_8 : 0.211	p_9 : 0.241	p_{10} : 0.0	p_{11} : 0.0
p_{12} : 0.0	p_{13} : 0.0	p_{14} : 0.0	p_{15} : 0.0	p_{16} : 0.0	p_{17} : 0.0
p_{18} : 0.0	p_{19} : 0.045	p_{20} : 0.0	p_{21} : 0.044	p_{22} : 0.045	p_{23} : 0.046
p_{24} : 0.0	p_{25} : 0.0	p_{26} : 0.0			

This implicates the following strategy when combining it with the strategy table, and compensating for None values (cases where we finish at the first round):

1. On the first round, the row player will choose to play with probability :

	J	Q	K
check	0.819	1.0	0.457
bet	0.181	0.0	0.543

2. On the second round, the row player will choose to play with probability :

	J-raised	Q-raised	K-raised
call	0.0	0.514	1.0
fold	1.0	0.486	0.0

We found these values by multiplying each p values to the strategy ("bet", "call", "check", "fold"), and adding the probabilities together.
The sum of "fold" and "call" for the row player end up not summing to 1, since there are None values where the player already bet in round 1.
In order to get more representative values, we have scaled the sum to 1 with the following code :

```
for i in range(3):
    strat_sum = p_fold_row[i] + p_call_row[i]
    p_fold_row[i] = p_fold_row[i]/strat_sum
    p_call_row[i] = p_call_row[i]/strat_sum
```

Column player Nash strategy

The **column** player has an expected payoff of 0.056, and a Nash equilibrium of :

q_0 : 0.0	q_1 : 0.0	q_2 : 0.0	q_3 : 0.0	q_4 : 0.0	q_5 : 0.0
q_6 : 0.0	q_7 : 0.0	q_8 : 0.0	q_9 : 0.466	q_{10} : 0.0	q_{11} : 0.0
q_{12} : 0.0	q_{13} : 0.2	q_{14} : 0.0	q_{15} : 0.0	q_{16} : 0.0	q_{17} : 0.0
q_{18} : 0.0	q_{19} : 0.0	q_{20} : 0.0	q_{21} : 0.0	q_{22} : 0.0	q_{23} : 0.0
q_{24} : 0.0	q_{25} : 0.2	q_{26} : 0.0	q_{27} : 0.0	q_{28} : 0.0	q_{29} : 0.133
q_{30} : 0.0	q_{31} : 0.0	q_{32} : 0.0	q_{33} : 0.0	q_{34} : 0.0	q_{35} : 0.0
q_{36} : 0.0	q_{37} : 0.0	q_{38} : 0.0	q_{39} : 0.0	q_{40} : 0.0	q_{41} : 0.0
q_{42} : 0.0	q_{43} : 0.0	q_{44} : 0.0	q_{45} : 0.0	q_{46} : 0.0	q_{47} : 0.0
q_{48} : 0.0	q_{49} : 0.0	q_{50} : 0.0	q_{51} : 0.0	q_{52} : 0.0	q_{53} : 0.0
q_{54} : 0.0	q_{55} : 0.0	q_{56} : 0.0	q_{57} : 0.0	q_{58} : 0.0	q_{59} : 0.0
q_{60} : 0.0	q_{61} : 0.0	q_{62} : 0.0	q_{63} : 0.0		

Using a similar calculation as for the row player, we get the following column player strategy :

	J-checked	Q-checked	K-checked	J-bet	Q-bet	K-bet
check	0.667	1.0	0.0	0.0	0.0	0.0
bet	0.333	0.0	1.0	0.0	0.0	0.0
fold	0.0	0.0	0.0	1.0	0.667	0.0
call	0.0	0.0	0.0	0.0	0.333	1.0

2.3.3 Nash strategy of a specific case

Here, we will compute the Nash strategy of a row-player who holds a jack, checks in the first round, and is then raised by the adversary.
One can see from the last strategy, that for the second round, the row player has a probability of 1 to fold for the "J-raised" column. He will therefore have to fold.

Another way to find this Nash strategy is to do the following: we find the rows and columns of the exiting payoff matrix M that comply with the strategy taken by both players until now.
This means that we need to find all strategies where the row player "checks" on the J column, and where the column player has a "bet" on K-checked or Q-checked

This can easily be found with the following code :

```
# where J column is "check"
row_strategy = df_p1[df_p1['J'] == 'check'].index

# where Q-checked or K-checked column is "bet"
col_strategy = df_p2[(df_p2['Q-checked'] == 'bet') | (df_p2['K-checked'] == 'bet')].index
```

After using this subset of the payoff matrix `M_sub= M[row_strategy,col_strategy]` we get the following Nash strategy for the row player :

p_0 : 0.0	p_1 : 0.667	p_2 : 0.0	p_3 : 0.333	p_4 : 0.0	p_5 : 0.0
p_6 : 0.0	p_7 : 0.0	p_8 : 0.0	p_9 : 0.0	p_{10} : 0.0	p_{11} : 0.0
p_{12} : 0.0	p_{13} : 0.0	p_{14} : 0.0	p_{15} : 0.0	p_{16} : 0.0	p_{17} : 0.0

When looking at the strategy descriptions, p1 and p3 have "fold" under the J-raised column, meaning the row player should fold in any case:

idx	J	Q	K	J-raised	Q-raised	K-raised
1	check	check	check	fold	fold	call
3	check	check	check	fold	call	call

Chapter 3

Morphing Images via Optimal Transport

3.1 Formulation of the problem

We can reformulate

$$\min_{c\in\Delta} \nu_1 W(p,c) + \nu_2 W(q,c) = \min_{c\in\Delta} \nu_1 W(p,c) + \nu_2 W(c,q)$$

given

$$\begin{aligned} W(a,b) = \min_{M} \quad & \sum_{i=1}^{d}\sum_{j=1}^{d} M_{ij}C_{ij} \\ \text{s.t.} \quad & \sum_{i=1}^{d} M_{ij} = a_j \ \ \forall j = 1,...,d \\ & \sum_{j=1}^{d} M_{ij} = b_j \ \ \forall i = 1,...,d \\ & M \geq 0 \end{aligned}$$

and

$$c \in \Delta \quad \text{where} \quad \Delta = \{c \in \mathbb{R}_+^d \ : \ \sum_{i=1}^{d} c_i = 1\} \Leftrightarrow \sum_{i=1}^{d} c_i = 1 \quad \text{with} \quad c \geq 0$$

as

$$\begin{aligned}
\min_{c} \;\; & \nu_1(\min_{M^1} \sum_{i=1}^{d}\sum_{j=1}^{d} M^1_{ij}C_{ij}) + \nu_2(\min_{M^2} \sum_{i=1}^{d}\sum_{j=1}^{d} M^2_{ij}C_{ij}) \\
\text{s.t.} \;\; & \sum_{i=1}^{d} M^1_{ij} = c_j \quad \forall j = 1, ..., d \\
& \sum_{j=1}^{d} M^1_{ij} = p_i \quad \forall i = 1, ..., d \\
& \sum_{i=1}^{d} M^2_{ij} = c_j \quad \forall j = 1, ..., d \\
& \sum_{j=1}^{d} M^2_{ij} = q_i \quad \forall i = 1, ..., d \\
& \sum_{i=1}^{d} c_i = 1 \\
& M^1 \geq 0, \; M^2 \geq 0 \;\text{ and }\; c \geq 0
\end{aligned}$$

The objective can be simplified further:

$$\min_{M^1, M^2, c} \; \sum_{k=1}^{2} \nu_k \sum_{i=1}^{d}\sum_{j=1}^{d} M^k_{ij} C_{ij}$$

3.2 Demonstration of the linearity of the problem

The problem derived in Question 1 is a linear program because it's in the form:

$$\begin{aligned}
\min \;\; & c_1x_1 + c_2x_2 + ... + c_nx_n \\
\text{s.t.} \;\; & a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \leq b_i \;\; \forall i \in \mathcal{M}_1 \\
& a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \geq b_i \;\; \forall i \in \mathcal{M}_2 \\
& a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n = b_i \;\; \forall i \in \mathcal{M}_3 \\
& x_j \geq 0 \;\; \forall j \in \mathcal{N}_1, \;\; x_j \leq 0 \;\; \forall j \in \mathcal{N}_2
\end{aligned}$$

We have in our case that:

- M^1_{ij}, M^2_{ij} and c are the decision variables
- ν_1, ν_2, C_{ij}, p, q are the fixed real constants

In particular since:

- The objective to minimize is a linear objective function
- All the constraints are equalities
- All right hand sides are non negative

- All decision variables are non negative

the problem is in standard form.

3.3 Redundancy of the constraint $\sum_{i=1}^{d} c_i = 1$

We know that

1) $p \in \Delta$ where $\Delta = \{p \in \mathbb{R}_+^d \ : \ \sum_{i=1}^{d} p_i = 1\}$
2) $\sum_{i=1}^{d} M_{ij}^1 = c_j \ \ \forall j = 1, ..., d$
3) $\sum_{j=1}^{d} M_{ij}^1 = p_i \ \ \forall i = 1, ..., d$

From (1) and (3) we get that:

$$\sum_{i=1}^{d} \sum_{j=1}^{d} M_{ij}^1 = \sum_{i=1}^{d} p_i = 1$$

From (2) and the previous result we get that:

$$\sum_{j=1}^{d} c_j = \sum_{j=1}^{d} \sum_{i=1}^{d} M_{ij}^1 = \sum_{i=1}^{d} \sum_{j=1}^{d} M_{ij}^1 = 1$$

So the constraint $\sum_{i=1}^{d} c_i = 1$ is redundant.

3.4 Implementation

To implement our linear program we are going to use python and the library CVXPY.

The main part of our code is:

```python
def interpolate_images(C, p1, p2, nu1, nu2, verbose = True):
    # Define the parameters
    d = C.shape[0]
    c = cp.Variable(d)
    M_1 = cp.Variable((d,d))
    M_2 = cp.Variable((d,d))

    # Set the constraints
    constraints = [ M_1 >= 0, M_2 >= 0, c >= 0 ]
    for i in range(d):
        constraints.append(cp.sum(M_1[i,:]) == p1[i])
        constraints.append(cp.sum(M_2[i,:]) == p2[i])
    for j in range(d):
        constraints.append(cp.sum(M_1[:,j]) == c[j])
        constraints.append(cp.sum(M_2[:,j]) == c[j])
    constraints.append(cp.sum(c) == 1)

    # Set the objective
    objective = cp.Minimize(nu1 * cp.sum(cp.multiply(M_1, C)) + nu2 *
        cp.sum(cp.multiply(M_2, C)))

    # Define the problem
    problem = cp.Problem(objective, constraints)

    # Solve the problem
    problem.solve(verbose = verbose, solver=cp.MOSEK)

    return c.value
```

where:

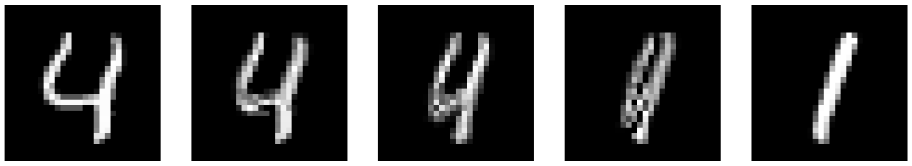
- Firstly we define our parameters and the constraints.
- Then we define the objective. In this case it's important to use the build in functions `cp.sum` and `cp.multiply` because if we try to add the constraints with for loops we run out of RAM.
- Finally we define our problem and we solve it.

3.5 Results

In both cases we are considering two distribution p_1 and p_2 and we are plotting the results obtained with:

1. $\nu_1 = 1$ and $\nu_2 = 0$
2. $\nu_1 = 0.75$ and $\nu_2 = 0.25$
3. $\nu_1 = 0.5$ and $\nu_2 = 0.5$
4. $\nu_1 = 0.25$ and $\nu_2 = 0.75$
5. $\nu_1 = 0$ and $\nu_2 = 1$

The results obtained with the first two distributions p and q are:



The results obtained with the second two distributions p and q are: