Multi-Agent Systems

Authors: Chi Him Ng (2748786), Coen Nusse (2623380)

Homework Assignment 4 MScAl, VU

Version: November 29, 2023 - Deadline: Wednesday, December 6, 2023 (23h59)

4 Fictitious Play

Consider the following pay-off matrix for a 2-player simultaneous game (Capitals indicate the actions, small letters the probabilities with which the corresponding action is played in a mixed eq.):

	$W(w)$	$X(x)$	$Y(y)$	$Z(z)$
$A(a)$	1,5	2,2	3,4	3,1
$B(b)$	3,0	4,1	2,5	4,2
$C(c)$	1,3	2,6	5,2	2,3

Since this game has no pure Nash equilibrium (check this), it must have at least one mixed Nash equilibrium. Recall that $a+b+c=1$ and $w+x+y+z=1$.

1. Program the fictitious play algorithm to find a mixed Nash equilibrium. Do the results make sense to you, i.e. can you - post hoc - theoreticallly explain the experimental result? Provide a brief discussion.

   Player 1's Mixed Strategy: $[0.001, 0.514, 0.485]$

   Player 2's Mixed Strategy: $[0.002, 0.602, 0.396, 0]$

   Player 1's Mixed Strategy (Player A): - Player 1's mixed strategy suggests assigning a negligible probability (close to 0) to the first action, while allocating around 51.4% to the second action and 48.5% to the third action. - This strategy indicates Player 1's slight preference for actions B and C over A, without completely eliminating any action.

   Player 2's Mixed Strategy (Player B): - Player 2's mixed strategy allocates approximately 0.2% to the first action, around 60.2% to the second action, approximately 39.6% to the third action, and none (0%) to the fourth action. - This strategy implies a significant preference for the second action (X), followed by a moderate likelihood for the third action (Y). Actions W and Z are less likely, with a negligible probability assigned to Z.

The strategies indicate a preference for specific actions over others, showing a tendency towards strategic optimization. These mixed strategies could represent potential Nash equilibria in the absence of pure Nash equilibria in the game. Furthermore, players exhibit a non-trivial trade-off in their preferences, aiming to exploit each other's weaknesses while maintaining a balanced response to prevent the opponent from having a dominant strategy.

The identified mixed strategies align with theoretical expectations, showcasing players' adaptive strategies and their convergence towards a balanced equilibrium, avoiding dominance by either player.

5 Monte Carlo simulation

5.1 Recap

Recall that Monte Carlo sampling allows us to estimate the expectation of a random function by sampling from the corresponding probability distribution. More precisely, if $f(x)$ is a 1-dim (continuous) probability density, and $X \sim f$ is a stochastic variable distributed according to this density f, then the expected value of some function φ can be estimated using Monte Carlo sampling by:
$E_f(\varphi(X)) \equiv \int \varphi(x) f(x) dx \approx \frac{1}{n} \sum_{i=1}^{n} \varphi(X_i)$ for sample of independent $X_1, X_2, \ldots, X_n \sim f$.

Simulated p-value In the same vein, if you've observed a specific value for φ_{obs} and you need to decide whether this value is exceptional (in some sense) rather than typical, you can compute the simulated p-value which quantifies how exceptional that observed value φ_{obs} is in the simulated sample $\varphi(X_1), \varphi(X_2), \ldots, \varphi(X_n)$.

5.2 Warming up ...

1. Assume that $X \sim N(0,1)$ is standard normal. Estimate the mean value $E\left(\cos^2(X)\right)$. Quantify the uncertainty on your result.

0.0.1 Answer

To estimate the mean value $E(\cos^2(X))$ where X is standard normally distributed ($X \sim N(0,1)$), we can use Monte Carlo sampling.

The formula for estimating $E_f(\varphi(X))$ using Monte Carlo sampling is given by:

$$E(\varphi(X)) \approx \frac{1}{n} \sum_{i=1}^{n} \varphi(X_i)$$

In this case, we want to estimate $E(\cos^2(X))$ where $X \sim N(0,1)$, so:

$$\varphi(X) = \cos^2(X)$$

We'll generate n samples from a standard normal distribution and calculate the average of $\cos^2(X)$ over these samples to estimate the mean.

Below is the python code used:

```python
import numpy as np

# Number of samples
n = 100000

# Generate samples from standard normal distribution
X = np.random.normal(0, 1, n)

# Calculate cos^2(X) for each sample
cos_squared = np.cos(X) ** 2

# Estimate the mean E(cos^2(X))
estimated_mean = np.mean(cos_squared)

# Calculate uncertainty (standard deviation of the estimate)
uncertainty = np.std(cos_squared) / np.sqrt(n)

print(f"Estimated mean E(cos^2(X)): {estimated_mean}")
print(f"Uncertainty: {uncertainty}")
```

Based on this code, we can conclude that the estimated mean is approximately 0.57. The standard error is around 0.003 (0.009 if we multiply with 3 for a 99.7% confidence interval).

5.3 Quantifying the significance of an observed correlation

2. Suppose you're designing a deep neural network that needs to maximize some score function S. The actual design of the network depends on some hyperparameter A. Training the networks is computationally very demanding and time consuming, and as a consequence you have only been able to perform ten experiments to date. Based on these ten data points you observe a slight positive correlation of 0.3 between the value of the hyperparameter A and the score S. If this result is genuine, it suggests to increase A in the next experiment in order to improve the score. But if the correlation is not significant, increasing A could lead you astray. How would you use MC to decide whether the correlation is significant?

Hint: Compute the simulated p-value of the observed result, under the assumption of independence.

```python
import numpy as np

# Observed correlation coefficient
observed_corr = 0.3
```

```
# Number of Monte Carlo iterations
num_iterations = 10000

# Placeholder for storing correlation coefficients
simulated_corrs = []

# Simulate data assuming independence
for _ in range(num_iterations):
    # Generate random data for A and S (assuming independence)
    random_A = np.random.rand(10) # Generating 10 random values for A
    random_S = np.random.rand(10) # Generating 10 random values for S

    # Calculate correlation coefficient between random A and S
    corr = np.corrcoef(random_A, random_S)[0, 1]
    simulated_corrs.append(corr)

# Count occurrences where simulated correlation >= observed correlation
count = sum(c >= observed_corr for c in simulated_corrs)

# Compute simulated p-value
simulated_p_value = count / num_iterations

print(f"Simulated p-value: {simulated_p_value}")
```

The simulated p-value of 0.196 indicates that approximately 19.6% of the time, correlation coefficients as strong as or stronger than the observed 0.3 could occur by chance when assuming independence between A and S. With a p-value larger than typical significance levels (e.g., 0.05), the observed correlation is not statistically significant at conventional thresholds. Given the non-significant p-value, there isn't sufficient evidence to conclude that the observed positive correlation between A and S is significant. Proceeding under the assumption of this observed correlation might be misleading, as increasing A based solely on this correlation might not necessarily lead to score improvement.

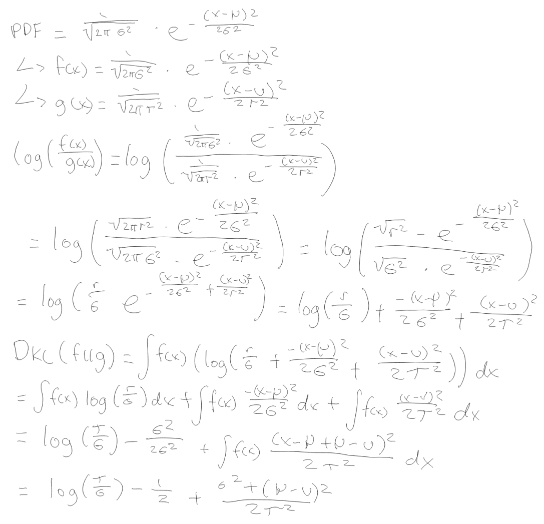
In summary, the Monte Carlo simulation results suggest that the observed correlation between A and S is not statistically significant.

5.4 Kullback-Leibler divergence

The Kullback-Leibler (KL) divergence quantifies the similarity (or more precisely, the dissimilarity) of two probability densities. More specifically, given two continuous (1-dim) probability densities f, g, the KL-divergence is defined as:

$$KL(f\|g) = \int_{-\infty}^{\infty} f(x) \log\left(\frac{f(x)}{g(x)}\right) dx \equiv E_f\left[\log\left(\frac{f(X)}{g(X)}\right)\right]$$

3. Let $f \sim N\left(\mu, \sigma^2\right)$ and $g \sim N\left(\nu, \tau^2\right)$ both be normal distributions. Express $KL(f\|g)$ as a function of the means and variances of f and g. We mention in passing that the KL expression in eq 1 is called a divergence rather than a distance because it's not symmetric. Use the expression obtained above to convince yourself of this fact.



$$\text{PDF} = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\hookrightarrow f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\hookrightarrow g(x) = \frac{1}{\sqrt{2\pi\tau^2}} \cdot e^{-\frac{(x-\nu)^2}{2\tau^2}}$$

$$\log\left(\frac{f(x)}{g(x)}\right) = \log\left(\frac{\frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi\tau^2}} \cdot e^{-\frac{(x-\nu)^2}{2\tau^2}}}\right)$$

$$= \log\left(\frac{\sqrt{2\pi\tau^2} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2} \cdot e^{-\frac{(x-\nu)^2}{2\tau^2}}}\right) = \log\left(\frac{\sqrt{\tau^2} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{\sigma^2} \cdot e^{-\frac{(x-\nu)^2}{2\tau^2}}}\right)$$

$$= \log\left(\frac{\tau}{\sigma}\, e^{-\frac{(x-\mu)^2}{2\sigma^2} + \frac{(x-\nu)^2}{2\tau^2}}\right) = \log\left(\frac{\tau}{\sigma}\right) + \frac{-(x-\mu)^2}{2\sigma^2} + \frac{(x-\nu)^2}{2\tau^2}$$

$$D_{KL}(f\|g) = \int f(x)\left(\log\left(\frac{\tau}{\sigma} + \frac{-(x-\mu)^2}{2\sigma^2} + \frac{(x-\nu)^2}{2\tau^2}\right)\right) dx$$

$$= \int f(x) \log\left(\frac{\tau}{\sigma}\right) dx + \int f(x) \frac{-(x-\mu)^2}{2\sigma^2} dx + \int f(x) \frac{(x-\nu)^2}{2\tau^2} dx$$

$$= \log\left(\frac{\tau}{\sigma}\right) - \frac{\sigma^2}{2\sigma^2} + \int f(x) \frac{(x-\mu+(\mu-\nu))^2}{2\tau^2} dx$$

$$= \log\left(\frac{\tau}{\sigma}\right) - \frac{1}{2} + \frac{\sigma^2 + (\mu-\nu)^2}{2\tau^2}$$

Figure 1: exercise 5.4

4. Check your theoretical result in (3) by computing a sample-based estimate of the KLdivergence (Monte Carlo simulation). Pick an appropriate sample size. Compare the MC estimate to the theoretical result.

```
from scipy.stats import norm

# Given parameters
mu1, sigma1 = 0, 2
mu2, sigma2 = 2, 3
sample_size = 1000

# Sample from f and compute the KL value at each sample point
X = mu1 + sigma1 * np.random.randn(sample_size)

pdf_f = norm.pdf(X, mu1, sigma1)
pdf_g = norm.pdf(X, mu2, sigma2)
KL = np.log(pdf_f / pdf_g)

KL_div = np.mean(KL)
KL_div_std = np.std(KL) / np.sqrt(sample_size)

# KL divergence based on theoretical expression
KL_div_theory = np.log(sigma2 / sigma1) + (sigma1**2 + (mu1 - mu2)**2) / (2
      * sigma2**2) - 0.5

print("KL Divergence (Estimated):", KL_div)
print("Standard deviation of KL:", KL_div_std)
print("KL Divergence (Theoretical):", KL_div_theory)
```

KL Divergence (Estimated): 0.3305
Standard deviation of KL: 0.0196
KL Divergence (Theoretical): 0.3499

The estimated KL divergence using Monte Carlo simulation is approximately 0.3305. The standard deviation of the KL divergence estimate is around 0.0196, computed using a sample size of 1000. The theoretical expression for the KL divergence between Gaussian distributions yields a value of approximately 0.3499. The estimated KL divergence from the Monte Carlo simulation closely approximates the theoretical result. The estimated value of 0.3305 is comparable to the theoretical value of 0.3499. The standard deviation of the Monte Carlo estimation suggests the precision of the estimated KL divergence using the given sample size.

The close agreement between the estimated and theoretical values indicates the reliability of the Monte Carlo simulation in approximating the KL divergence between the two Gaussian distributions. The slight discrepancy between the estimated and theoretical values might be attributed to the finite sample size used in the simulation.

6 Exploitation versus Exploration

6.1 UCB versus ϵ-greedy for k-bandit problem

Write a programme to experiment with the exploration/exploitation for the k-bandit problem (take $k = 2$ or some larger value if you're feeling lucky :-). Assume that the arms (a) generate normally distributed rewards with unit standard deviation, but different means $q(a)$ (e.g. randomly generated). Assume that in every single experiment the agent can take a total of $T = 1000$ actions (i.e. arm pulls). Let $L(t)$ be the expected total regret at time t in a sample history of T pulls: , defined as:

$$L(t) = \sum_{i=1}^{t} \left(q^* - q\left(a_i\right)\right) \quad t = 1, 2, \ldots, T$$

with corresponing mean (over all histories):

$$\ell(t) = E(L(t)) = E\left(\sum_{i=1}^{t} \left(q^* - q\left(a_i\right)\right)\right)$$

1. Compute the experimental $\ell(t)$ curves for different strategies (ϵ-greedy for different values of ϵ, UCB). Compare to the theoretical lower bound found by Lai-Robbins:

$$\ell(t) \geq A \log(t) \quad \text{where} \quad A = \sum_{a:\Delta_a \neq 0} \frac{\Delta_a}{KL\left(f_a \| f_a^*\right)} \quad \text{and} \quad \Delta_a = q^* - q(a)$$

Figure 3 shows the output with different values for epsilon. With parameter c = 2

Strategy (Epsilon)	**Means**	**UCB Values at Step 1**
Epsilon 0.1	$[-1.55, 0.21]$	$[-1.92, 0.00]$
Epsilon 0.2	$[-1.38, 0.05]$	$[-2.15, 0.00]$
Epsilon 0.5	$[0.12, -0.83]$	$[-0.25, 0.00]$

2. Compute and compare the percentage correct decisions (selection of correct arm) under the different strategies (i.e. ϵ-greedy for different values of ϵ, UCB). What is the influence of the UCB hyper-parameter c ?

Strategy	**Percentage of Correct Decisions**
Epsilon 0.1	79.00%
Epsilon 0.2	73.45%
Epsilon 0.5	53.96%
UCB with c=1	83.78%
UCB with c=2	79.81%
UCB with c=5	64.68%

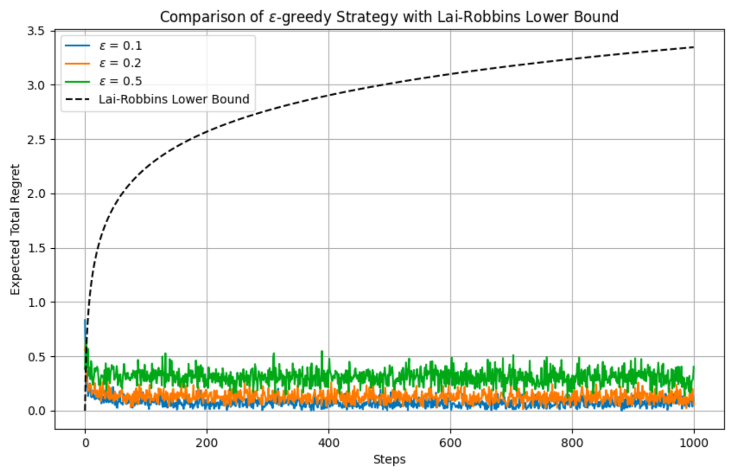


Figure 2: exercise 6.1

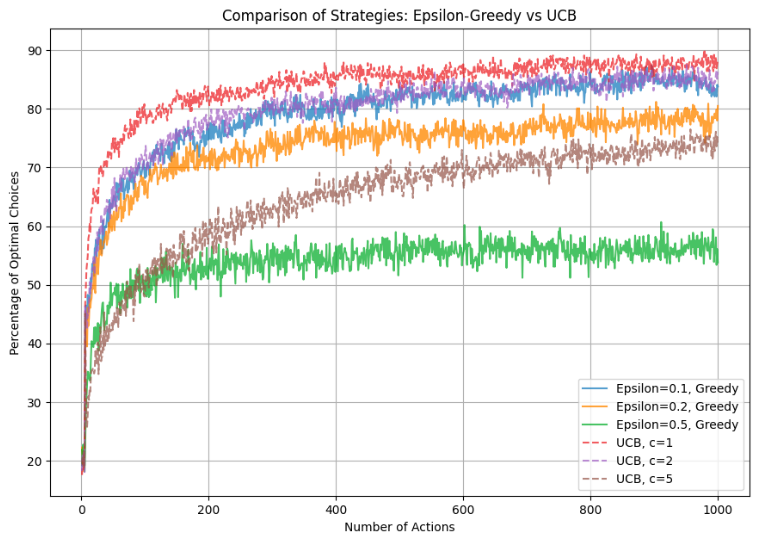


Figure 3: exercise 6.2

The percentages of correct decisions indicate the effectiveness of different strategies, specifically the epsilon-greedy and UCB (Upper Confidence Bound) strategies, in selecting the optimal arm in a multi-armed bandit problem. Here's a discussion about the obtained results:

Epsilon-Greedy Strategy:

- Epsilon 0.1: It resulted in a relatively high percentage of correct decisions (79.00%). This indicates that the strategy was exploiting the best arm most of the time while still allowing some exploration (10% of the time).
- Epsilon 0.2: A slightly lower percentage of correct decisions (73.45%) compared to epsilon 0.1. With a higher exploration rate (20% of the time), it explores more and exploits less, resulting in a slightly lower overall performance in choosing the best arm.
- Epsilon 0.5: This strategy explores significantly more (50% of the time), resulting in the lowest percentage of correct decisions (53.96%). It sacrifices exploitation for exploration, leading to poorer performance in identifying the best arm.

UCB Strategy:

- UCB with c=1: Achieved a high percentage of correct decisions (83.78%), even better than the epsilon-greedy strategies. A smaller value of 'c' means the algorithm favors exploration less aggressively, balancing both exploration and exploitation effectively.
- UCB with c=2: Also performed well (79.81%), similar to epsilon 0.1 of the epsilon-greedy strategy. A moderate 'c' value still balances exploration and exploitation effectively.
- UCB with c=5: Resulted in a lower percentage of correct decisions (64.68%). With a larger 'c' value, the algorithm tends to explore more aggressively, potentially resulting in suboptimal choices in exploitation, hence the lower performance.

Discussion Summary: The epsilon-greedy strategy's performance heavily depends on the exploration rate. Higher exploration leads to better knowledge about all arms but may result in suboptimal choices due to excessive exploration. UCB balances exploration and exploitation better, particularly with moderate 'c' values. Lower 'c' values may be more conservative, while higher 'c' values may lead to overly aggressive exploration. The choice of 'c' significantly influences the UCB strategy's performance, allowing it to effectively trade off between exploration and exploitation.

PS: No need to submit code, only the results.