Ruleta europeană

Aelenei Alex, Neacșu Mihai

Cuprins

1 Introducere

Jocurile de noroc sunt o constantă în lumea contemporană și considerăm că fascinația oamenilor cu acestea pe parcursul istoriei le conferă un rol aparte în existența umană. Rolul lor cultural și social este incontestabil, în ciuda faptului că atât din punct de vedere probabilistic, cât și statistic, jucătorii întotdeauna vor pierde mai mult decât vor câștiga.

Dintre toate jocurile de noroc pe care umanitatea le-a conceput, am ales ruleta europeană. Aceasta se presupune că își are originea în secolul XVIII în Franța, chiar Blaise Pascal formulând o formă primitivă a jocului în secolul XVII. Forma actuală a jocului își are originea în Paris, 1796, urmând următoarele reguli:

- **Demers:** Masa de ruletă are o roată cu numerele de la 0 la 36 dispuse pe marginea roții într-o anumită ordine (0, 32, 15, 19, 4, 21, 2, 25, 17, 34, 6, 27, 13, 36, 11, 30, 8, 23, 10, 5, 24, 16, 33, 1, 20, 14, 31, 9, 22, 18, 29, 7, 28, 12, 35, 3, 26), fiecare având un spațiu în care poate încăpea o singură bilă. Roata începe să se învârtă, iar casa (sau în unele cazuri chiar pariorul) aruncă bila în interiorul roții care se învârte. Valoarea pe care ajunge să pice bila este valoarea câștigătoare.
- **Condiții de câștig:** În funcție de pariul realizat de jucător, se validează dacă acesta a câștigat sau nu, cota lui depinzând de tipul de pariu pe care l-a făcut. Următoarele tipuri de pariuri au fost considerate pentru acest proiect:
  - *Single:* Jucătorul pariază pe un singur număr de la 0 la 36, cota fiind 35 la 1.
  - *Rând:* Jucătorul pariază pe rânduri de câte trei numere consecutive (vezi masă), cota fiind 17 la 1.
  - *Coloană:* Jucătorul pariază pe coloane de câte 12 numere (vezi masă), cota fiind 2 la 1.
  - *Duzină:* Jucătorul pariază pe grupe de câte 12 numere consecutive (vezi masă), cota fiind 2 la 1.
  - *Culoare:* Jucătorul pariază ori pe culoarea roșie (32, 19, 21, 25, 34, 27, 36, 30, 23, 5, 16, 1, 14, 9, 18, 7, 12, 3) ori pe culoarea neagră (15, 4, 2, 17, 6, 13, 11, 8, 10, 24, 33, 20, 31, 22, 29, 28, 35, 26), cota fiind 2 la 1.
  - *Paritate:* Jucătorul pariază ori pe numerele pare (excluzându-l pe 0) ori pe numerele impare, cota fiind 2 la 1.
  - *Jumătate:* Jucătorul pariază ori pe numerele de la 1 la 18, ori pe numerele de la 19 la 36, cota fiind 2 la 1.

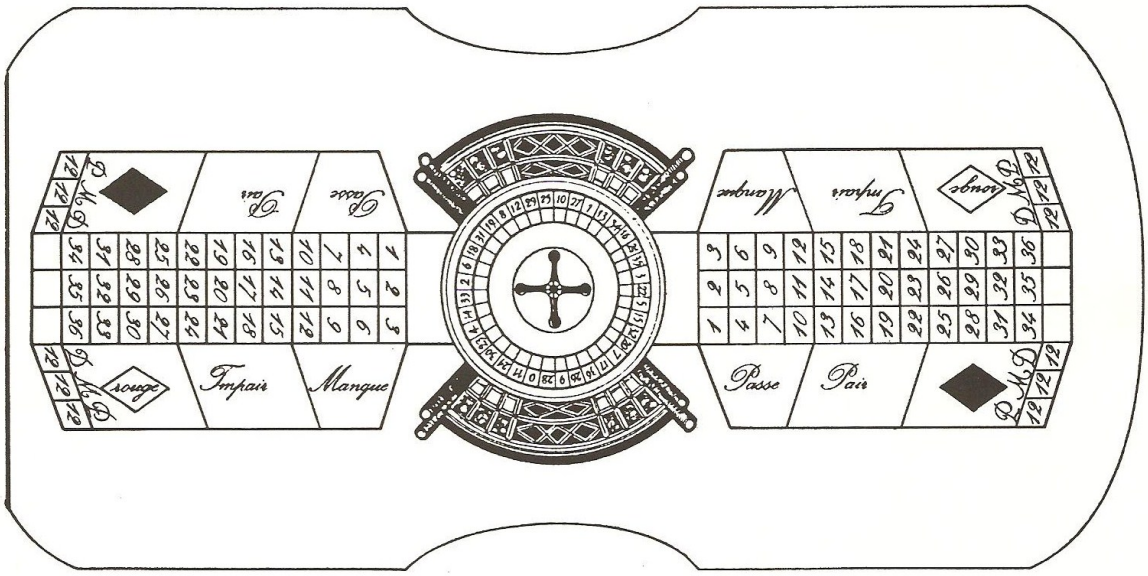

Figura 1: Masă de ruletă [1] - Franța, secolul XVIII

2 Scopul proiectului

Scopul proiectului este să analizeze o serie de rulete (una corectă și 4 mâsluite) și să compare valorile rezultate în urma unei simulări de tip Monte Carlo pentru a determina cu o precizie relativă dacă o ruletă este corectă sau mâsluită.

3 Modelul matematic

Din moment ce jocul de ruletă are 37 de spații distincte, definim spațiul de probabilități pe care vom lucra ca fiind $(\Omega, \mathcal{P}(\Omega), P)$, unde $\Omega = \{n \in \mathbb{N} | 0 \leq n \leq 36\}$ Începem prin a defini rezultatul jocului de ruletă ca o variabilă aleatoare discretă X:

$$X \sim \begin{pmatrix} 0 & 1 & 2 & \dots & 36 \\ P(0) & P(1) & P(2) & \dots & P(36) \end{pmatrix} \quad \text{cu} \quad \sum_{i=0}^{36} P(i) = 1 \tag{1}$$

Așadar, putem afirma că pentru orice joc de ruletă:

$$\forall A \in \mathcal{P}(\Omega), \quad P(A) = \sum_{x \in A} p(x) \tag{2}$$

Astfel, putem calcula cu ușurință media și varianța:

$$\mathbb{E}[X] = \mu = \sum_{i=0}^{36} iP(i) \quad \mathbb{V}ar[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{3}$$

În cazul particular în care ruleta este corectă ($p_i = \frac{1}{37}, \forall i = \overline{0, 36}$):

$$\forall A \in \mathcal{P}(\Omega), \quad P(A) = \frac{|A|}{37} \tag{4}$$

$$\mathbb{E}[X] = \mu = \frac{0 + 1 + \dots + 36}{37} = 18 \tag{5}$$

$$\mathbb{V}ar[X] = \sum_{i=0}^{36} i^2 P(i) - \mu^2 = 438 - 324 = 114 \tag{6}$$

Pentru a modela rezultatul unui pariu, vom folosi o valoare aleatoare $X \sim Bernoulli(p)$, unde p este probabilitatea câștigului. Probabilitatea p va fi dată de suma probabilităților asociate numerelor din acea categorie de pariuri. Așadar, într-o serie de n runde de ruletă, fiecare rotire va fi asociată unei variabile aleatoare:

$$X_i \overset{\text{i.i.d.}}{\sim} Bernoulli(p), \ \forall i = \overline{1, n} \tag{7}$$

Așadar, vom avea media empirică:

$$\overline{S_n} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{8}$$

care aproximează $\mathbb{E}[X_1] = p$ cu o varianță de $\frac{\mathbb{V}ar[X_1]}{n} = \frac{p(1-p)}{n}$. Considerând o eroare de aproximare $\varepsilon = 10^{-2}$, aplicăm inegalitatea lui Cebâșev lui $\overline{S_n}$:

$$P(|\overline{S_n} - p| \geq \varepsilon) \leq \frac{\mathbb{V}ar[\overline{S_n}]}{\varepsilon^2} = \frac{p(1-p)}{n\varepsilon^2} \tag{9}$$

În urma unui calcul simplu deducem că nivelul de încredere al unei aproximări este:

$$P(|\overline{S_n} - p| < \varepsilon) \geq 1 - \frac{p(1-p)}{n\varepsilon^2} \tag{10}$$

Folosind inegalitatea lui Cebâșev și presupunând un nivel de încredere $\alpha = 1 - 10^{-2}$, putem deduce:

$$P(|\overline{S_n} - p| < \varepsilon) \geq 1 - \frac{p(1-p)}{n\varepsilon^2} \geq \alpha \iff \frac{p(1-p)}{n\varepsilon^2} \leq 1 - \alpha \tag{11}$$

$$\iff \frac{n\varepsilon^2}{p(1-p)} \geq \frac{1}{1-\alpha} \tag{12}$$

$$\iff n \geq \frac{p(1-p)}{\varepsilon^2(1-\alpha)} \tag{13}$$

De asemenea, putem aplica inegalitatea Chernoff-Hoeffding:

$$P(|\overline{S_n} - p| \geq \varepsilon) \leq 2e^{-2n\varepsilon^2} \tag{14}$$

Presupunând un nivel de încredere $\alpha = 1 - 10^{-2}$, putem deduce:

$$P(|\overline{S_n} - p| < \varepsilon) \geq 1 - 2e^{-2n\varepsilon^2} \geq \alpha \iff e^{-2n\varepsilon^2} \leq \frac{1-\alpha}{2} \tag{15}$$

$$\iff n \geq \frac{1}{2\varepsilon^2} \ln \frac{2}{1-\alpha} \tag{16}$$

În cazul particular în care ruleta este corectă, vom avea următorul tabel de valori:

Tip pariu	p	Cebâșev	Chernoff-Hoeffding
Single	1/37	26296.5668371	26491.5868327
Rând	3/37	74506.9393718	26491.5868327
Coloană	12/37	219138.056976	26491.5868327
Duzină	12/37	219138.056976	26491.5868327
Culoare	18/37	249817.384953	26491.5868327
Paritate	18/37	249817.384953	26491.5868327
Jumătate	18/37	249817.384953	26491.5868327

Așadar, în urma a $n = 50000$ de teste vom putea trage o serie de concluzii cu marjă de eroare relativ mică și cu un nivel de încredere relativ mare. Vom realiza simulări pentru fiecare dintre cele 7 tipuri de pariuri de mai sus, plus încă o categorie, una mixtă. Această categorie cuprinde toate pariurile de mai sus, dintre care un jucător alege aleator unul din ele. Această categorie a fost adăugată pentru a simula comportamentul jucătorilor reali și are probabilitatea $p = 0.11209570225963675$.

4 Codul

În primul rând, proiectul este realizat în Python 3.11 și se găsește pe github. În al doilea rând, mediul de lucru se inițializează cu `pipenv shell`, iar pachetele necesare (numpy și matplotlib) se instalează rulând comanda `pip install -r requirements.txt` din directorul cu proiectul.

Începem prin a defini în 1 eroarea globală și clasele care definesc o clasă de pariuri (precum single, rând, coloană etc.). Clasa `RouletteBet` reprezintă un singur tip de pariu, cu toate metodele asociate, în timp ce clasa `Roulette` este una statică, conținând doar instanțele statice ale tipurilor de pariuri și valorile posibile ale ruletei.

În 2 definim o serie de funcții utilitare:

- `generate_roulette_probability_distribution_normal` generează în baza unei distribuții normale (caracterizate prin μ și σ) o distribuție de probabilități pentru ruletă. Aceasta normalizează valorile generate de distribuția normală pentru ca suma probabilităților să fie egală cu 1.
- `get_probability_distribution_data` generează media și varianța unei distribuții de probabilități
- `evaluate_number_against_probability_distribution` generează în baza unei valori din intervalul $[0, 1]$ și a unei distribuții de probabilități o valoare de pe ruletă.
- `get_expected_probability_for_strategies` determină probabilitatea unei liste de tipuri de pariuri în cazul în care ruleta este corectă.

```python
# Global constants

error_margin = 1e-2

# Roulette base class

class RouletteBet():
    def __init__(self, identifier: str, values: Iterable[float], payout: float):
        self.identifier = identifier
        self.values = values
        self.payout = payout

    def evaluate_action(self, value):
        return (value in self.values)

    def get_expected_probability(self):
        return float(len(self.values) / 37)

class Roulette():
    ALL_VALUES_COUNT = 37
    ALL_VALUES = [value for value in range(37)]

    SINGLE_BETS = [RouletteBet(f"straight_{index}", [index], 36) for index in range(ALL_VALUES_COUNT)]

    ROW_BETS = [RouletteBet(f"row_{index + 1}", [index * 3 + 1, index * 3 + 2, index * 3 + 3], 12) for
                range(12)]

    FIRST_COLUMN_BET = RouletteBet('column_1', range(1, ALL_VALUES_COUNT, 3), 3)
    SECOND_COLUMN_BET = RouletteBet('column_2', range(2, ALL_VALUES_COUNT, 3), 3)
    THIRD_COLUMN_BET = RouletteBet('column_3', range(3, ALL_VALUES_COUNT, 3), 3)
    COLUMN_BETS = [FIRST_COLUMN_BET, SECOND_COLUMN_BET, THIRD_COLUMN_BET]

    FIRST_DOZEN_BET = RouletteBet('dozen_1', range(1, 13), 3)
    SECOND_DOZEN_BET = RouletteBet('dozen_2', range(13, 25), 3)
    THIRD_DOZEN_BET = RouletteBet('dozen_3', range(25, ALL_VALUES_COUNT), 3)
    DOZEN_BETS = [FIRST_DOZEN_BET, SECOND_DOZEN_BET, THIRD_DOZEN_BET]

    RED_BET = RouletteBet('reds', [32, 19, 21, 25, 34, 27, 36, 30, 23, 5, 16, 1, 14, 9, 18, 7, 12, 3],
    BLACK_BET = RouletteBet('black', [15, 4, 2, 17, 6, 13, 11, 8, 10, 24, 33, 20, 31, 22, 29, 28, 35, 2
    COLOR_BETS = [RED_BET, BLACK_BET]

    EVEN_BET = RouletteBet('even', range(2, ALL_VALUES_COUNT, 2), 2)
    ODD_BET = RouletteBet('odd', range(1, ALL_VALUES_COUNT, 2), 2)
    PARITY_BETS = [EVEN_BET, ODD_BET]

    FIRST_HALF_BET = RouletteBet('half_1', range(1, 19), 2)
    SECOND_HALF_BET = RouletteBet('half_2', range(19, ALL_VALUES_COUNT), 2)
    HALF_BETS = [FIRST_HALF_BET, SECOND_HALF_BET]

    ALL_BETS = SINGLE_BETS + ROW_BETS + COLUMN_BETS + DOZEN_BETS + COLOR_BETS + PARITY_BETS + HALF_BETS
```

Listing 1: Modelul ruletei și al tipurilor de pariuri

```python
# Distribution utilities

def generate_roulette_probability_distribution_normal(mu: float, sigma: float):
    rng = np.random.default_rng()
    values = rng.normal(mu, sigma, Roulette.ALL_VALUES_COUNT)
    values_sum = sum(values)
    values_normalized = [float(value / values_sum) for value in values]
    return values_normalized


def get_probability_distribution_data(probability_distribution: Iterable[float]):
    mean = sum([index * value for index, value in enumerate(probability_distribution)])
    squared_mean = sum([(index ** 2) * value for index, value in enumerate(probability_distribution)])
    variance = squared_mean - mean ** 2
    return (mean, variance)


def evaluate_number_against_probability_distribution(value: float, probability_distribution: Iterable[f
    probability_sum = 0
    for index, probability in enumerate(probability_distribution):
        probability_sum += probability
        if probability_sum > value:
            return index
    return len(probability_distribution) - 1


def get_expected_probability_for_strategies(betting_strategies: Iterable[RouletteBet]):
    return sum([betting_strategy.get_expected_probability() for betting_strategy in betting_strategies]
        betting_strategies)
```

Listing 2: Utilitarele pentru distribuții

În continuare, în 3 definim funcția `generate_roulette_probability_distribution_data` care generează distribuțiile de probabilități și le salvează în fișiere. Cele 4 seturi de probabilități pentru rulete mâsluite sunt generate în baza distribuțiilor normale cu valorile lui $(\mu, \sigma) \in \{(1, 0.05), (1, 0.1), (1, 0.15), (1, 0.2)\}$. Astfel, distribuțiile probabilităților sunt gradual mai variate, simulând rulete din ce în ce mai mâsluite. De asemenea, definim și funcția `generate_probability_distribution_plot` care generează grafice pentru distribuțiile de probabilități, pentru o vizualizare mai simplă a datelor.

```python
# Generate probability distribution data

def generate_roulette_probability_distribution_data():
    distribution_params = [(1, 0.05), (1, 0.1), (1, 0.15), (1, 0.2)]

    usable_data = [[float(1 / Roulette.ALL_VALUES_COUNT) for _index in range(Roulette.ALL_VALUES_COUNT)]

    distributions_raw_data = [['Roulette value', 'Correct probability'] + [f"Fixed probability {index +
                                                                           range(len(distribution_param
    distributions_raw_data += [[] for _index in range(Roulette.ALL_VALUES_COUNT)]

    for index in range(Roulette.ALL_VALUES_COUNT):
        distributions_raw_data[index + 1].append(index)
        distributions_raw_data[index + 1].append(float(1 / Roulette.ALL_VALUES_COUNT))

    for distribution_param in distribution_params:
        distribution = generate_roulette_probability_distribution_normal(distribution_param[0], distrib
        usable_data.append(distribution)
        for index, value in enumerate(distribution):
            distributions_raw_data[index + 1].append(value)

    distributions_raw_file = open("../data/distributions-raw.csv", "w", newline='')
    distributions_raw_file_writer = csv.writer(distributions_raw_file, delimiter=',')
    distributions_raw_file_writer.writerows(distributions_raw_data)

    distributions_data = [['Probability distribution', 'Weighted mean', 'Variance']]
    for distribution_index, distribution in enumerate(usable_data):
        mean, variance = get_probability_distribution_data(distribution)
        distributions_data.append([distribution_index + 1, mean, variance])

    distributions_file = open("../data/distributions.csv", "w", newline='')
    distributions_file_writer = csv.writer(distributions_file, delimiter=',')
    distributions_file_writer.writerows(distributions_data)

    return usable_data

# Distribution plot generation

def generate_probability_distribution_plot(index: int, probability_distribution: Iterable[float]):
    xpoints = [value for value in range(Roulette.ALL_VALUES_COUNT)]
    ypoints = probability_distribution

    plt.rcParams["figure.figsize"] = [20, 5]
    plt.rcParams["figure.autolayout"] = True
    plt.bar(xpoints, ypoints)
    plt.xticks(xpoints)
    plt.savefig(f"../img/probability-distribution-{index}")
    plt.clf()
```

Listing 3: Funcția care generează distribuțiile probabilităților și salvează datele în fișiere `.csv` și funcția care generează graficele distribuțiilor

În continuare, vom realiza simularea Monte Carlo. În metoda `simulate_roulette_game` din 4, având un număr de teste, o distribuție de probabilități și o strategie, simulăm jocul de ruletă și numărăm câte rezultate favorabile am avut. După ce se termină simularea, completăm matricea de rezultate cu datele pentru simulare, dar și pentru inegalitățile Cebâșev și Chernoff-Hoeffding. Am adăugat, de asemenea, metoda utilitară `initialize_simulation_report_data` care generează header-ul fiecărui raport.

```
def simulate_roulette_game(distribution_data_index: int, sample_size: int, betting_strategies_name: str
                           betting_strategies: Iterable[RouletteBet], probability_distribution: Iterabl
                           results: Iterable, row_index: int, column_index: int):
    favorable_outcomes = 0
    for _sample_index in range(sample_size):
        betting_strategy = np.random.choice(betting_strategies)
        sample_value = np.random.rand()
        roulette_value = evaluate_number_against_probability_distribution(sample_value, probability_dis
        if betting_strategy.evaluate_action(roulette_value):
            favorable_outcomes += 1
    results[distribution_data_index][row_index][column_index] = float(favorable_outcomes / sample_size)

    # Chebyshev
    expected_probability = get_expected_probability_for_strategies(betting_strategies)
    results[distribution_data_index][row_index][column_index + 1] = (expected_probability * (
            1 - expected_probability)) / (
                                                                        error_margin * sample_size)

    # Chernoff
    exponent = float((-2) * sample_size * (error_margin ** 2))
    results[distribution_data_index][row_index][column_index + 2] = 2 * math.exp(exponent)

    print(
        f"Finished Monte Carlo simulation for distribution {distribution_data_index + 1} - {sample_size


def initialize_simulation_report_data(sample_sizes: Iterable[int], betting_strategies_list: Iterable,
                                      inequalities: Iterable[str]):
    report_data = [['Sample size'], ['Expected']]
    for strategy_name, betting_strategies in betting_strategies_list.items():
        report_data[0].append(strategy_name)
        report_data[1].append(get_expected_probability_for_strategies(betting_strategies))

        for inequality in inequalities:
            report_data[0].append(f"{strategy_name} {inequality}")
            report_data[1].append("-")

    for sample_size in sample_sizes:
        report_data.append([sample_size] + [0] * len(betting_strategies_list) * (len(inequalities) + 1)
    return report_data
```

Listing 4: Funcția care realizează simularea Monte Carlo și funcția care inițializează raportul `.csv`

În cele din urmă avem partea principală a programului în 5 și 6, spartă în două secțiuni mai mici: prima dintre ele generează distribuțiile de probabilități, generează graficele asociate și apoi începe simulările Monte Carlo multithreaded, și cea de a doua parte, în care se generează toate graficele asociate simulărilor Monte Carlo, după ce toate thread-urile au fost unite cu thread-ul principal.

```
# Main part of the program

print("Generating distributions!")
distributions_data = generate_roulette_probability_distribution_data()

print("Generating distribution plots!")
for distribution_data_index in range(len(distributions_data)):
    generate_probability_distribution_plot(distribution_data_index + 1,
                                           distributions_data[distribution_data_index])

sample_unit = 1000
sample_sizes = [sample_unit * index for index in range(1, 51)]
betting_strategies_list = {
    'Single': Roulette.SINGLE_BETS,
    'Row': Roulette.ROW_BETS,
    'Column': Roulette.COLUMN_BETS,
    'Dozen': Roulette.DOZEN_BETS,
    'Color': Roulette.COLOR_BETS,
    'Parity': Roulette.PARITY_BETS,
    'Half': Roulette.HALF_BETS,
    'All': Roulette.ALL_BETS
}

print("Starting Monte Carlo simulation!")

threads = [[[None] * (len(betting_strategies_list) + 1)] * (len(sample_sizes) + 1)] * len(distributions
distribution_report_data = []
distribution_report_data_inequalities = ['Chebyshev', 'Chernoff']
distribution_report_data_inequalities_count = len(distribution_report_data_inequalities)
distribution_report_data_header_size = 2

for distribution_data_index, distribution_data in enumerate(distributions_data):
    distribution_report_data.append(
        initialize_simulation_report_data(sample_sizes, betting_strategies_list, distribution_report_da
    for sample_size_index, sample_size in enumerate(sample_sizes):
        betting_strategies_index = 0
        for betting_strategies_name, betting_strategies in betting_strategies_list.items():
            threads[distribution_data_index][sample_size_index][betting_strategies_index] = Thread(
                target=simulate_roulette_game,
                args=(
                    distribution_data_index,
                    sample_size,
                    betting_strategies_name,
                    betting_strategies,
                    distribution_data,
                    distribution_report_data,
                    sample_size_index + distribution_report_data_header_size,
                    betting_strategies_index * (distribution_report_data_inequalities_count + 1) + 1
                )
            )
            threads[distribution_data_index][sample_size_index][betting_strategies_index].start()
            betting_strategies_index += 1
```

Listing 5: Partea principală a programului care generează distribuțiile de probabilități, realizează simulările Monte Carlo multithreaded și stochează rezultatele

```
for distribution_data_index, distribution_data in enumerate(distributions_data):
    for sample_size_index, sample_size in enumerate(sample_sizes):
        betting_strategies_index = 0
        for betting_strategies_name, betting_strategies in betting_strategies_list.items():
            threads[distribution_data_index][sample_size_index][betting_strategies_index].join()
            betting_strategies_index += 1

    distribution_report_file = open(f"../data/distribution-{distribution_data_index + 1}-report.csv", "
    distribution_report_file_writer = csv.writer(distribution_report_file, delimiter=',')
    distribution_report_file_writer.writerows(distribution_report_data[distribution_data_index])

    plt.rcParams["figure.figsize"] = [25, 5]
    plt.rcParams["figure.autolayout"] = True

    probabilities = np.empty((len(betting_strategies_list), len(sample_sizes)))
    for betting_strategies_index, betting_strategies in enumerate(betting_strategies_list.items()):
        for sample_size_index, sample_size in enumerate(sample_sizes):
            probabilities[betting_strategies_index][sample_size_index] = \
                distribution_report_data[distribution_data_index][
                    sample_size_index + distribution_report_data_header_size][
                    betting_strategies_index * (distribution_report_data_inequalities_count + 1) + 1]

    graph_data_list = [(
        [probabilities[0]], [1 / 37], 'single'
    ), (
        [probabilities[1]], [3 / 37], 'three'
    ), (
        [probabilities[7]], [get_expected_probability_for_strategies(Roulette.ALL_BETS)], 'all'
    ), (
        [probabilities[2], probabilities[3]], [12 / 37], 'twelve'
    ), (
        [probabilities[4], probabilities[5], probabilities[6]], [18 / 37], 'eighteen'
    ), (
        probabilities, [1 / 37, 3 / 37, get_expected_probability_for_strategies(Roulette.ALL_BETS), 12
        'total'
    )]

    for graph_data in graph_data_list:
        for probability in graph_data[0]:
            plt.plot(np.array(sample_sizes), np.array(probability), marker='o')

        for horizontal_line in graph_data[1]:
            plt.axhline(y=horizontal_line, color='r', linestyle='-')

        plt.grid(True)
        plt.xticks(sample_sizes, [str(math.floor(sample_size / sample_unit)) for sample_size in sample_
        plt.savefig(f"../img/monte-carlo-simulation-{distribution_data_index + 1}-{graph_data[2]}")
        plt.clf()
```

Listing 6: Partea principală a programului care generează graficele simulărilor

5 Ruleta corectă

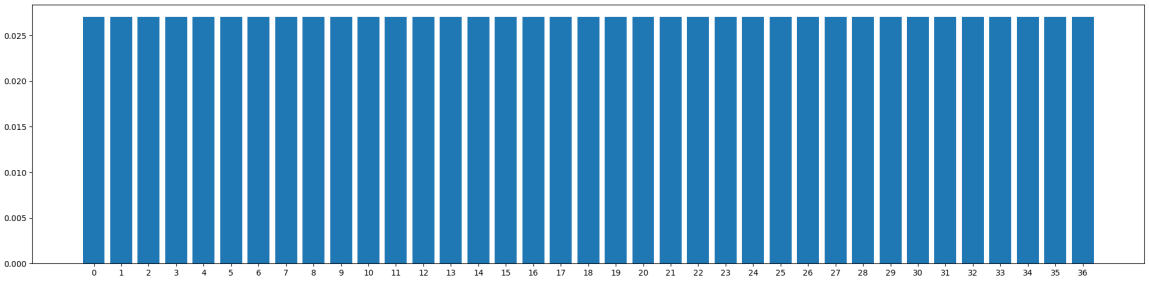

Figura 2: Distribuția corectă

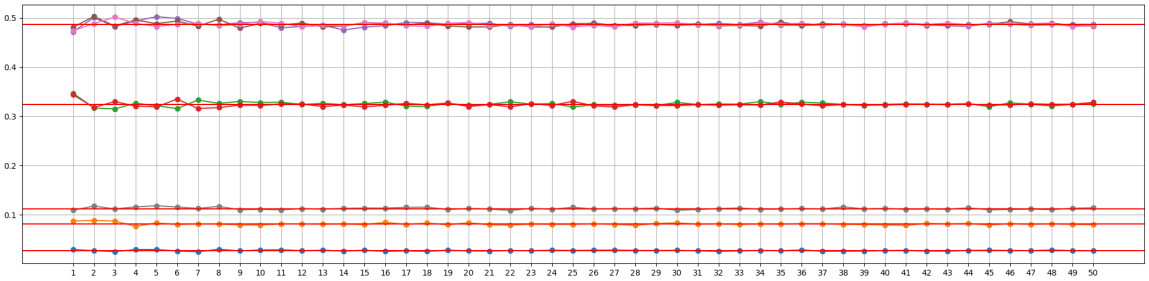

Figura 3: Simulare distribuția corectă - toate strategiile

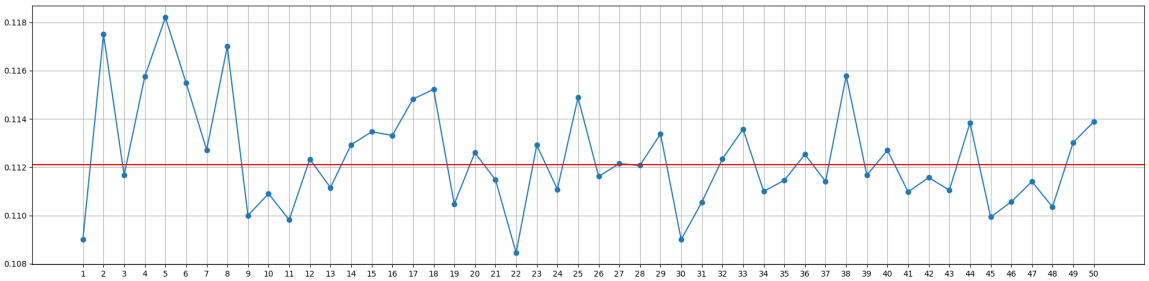

Figura 4: Simulare distribuția corectă - strategia mixtă

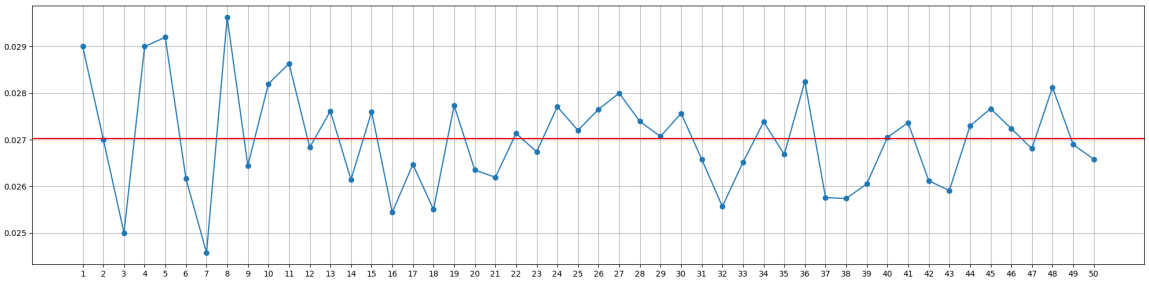

Figura 5: Simulare distribuția corectă - strategia single

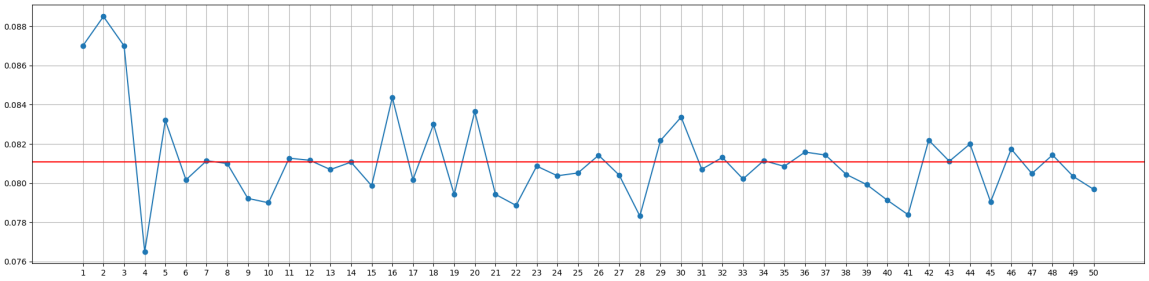

Figura 6: Simulare distribuția corectă - strategia rând

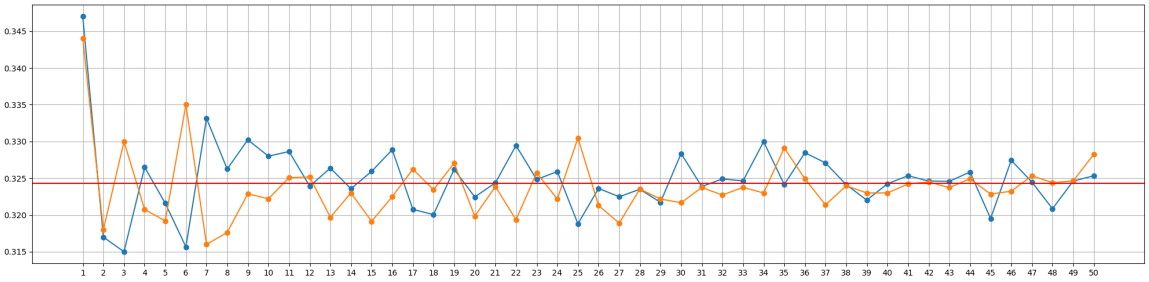

Figura 7: Simulare distribuția corectă - strategiile coloană și duzină

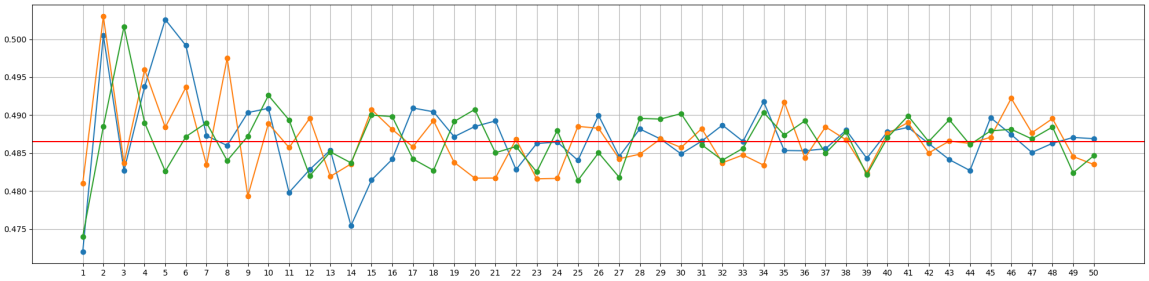

Figura 8: Simulare distribuția corectă - strategiile jumătate, paritate și culoare

6 Ruleta mâsluită 1

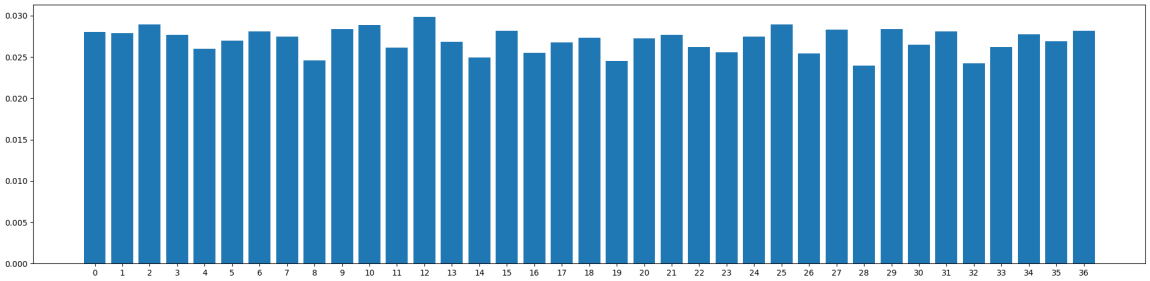

Figura 9: Distribuția mâsluită 1

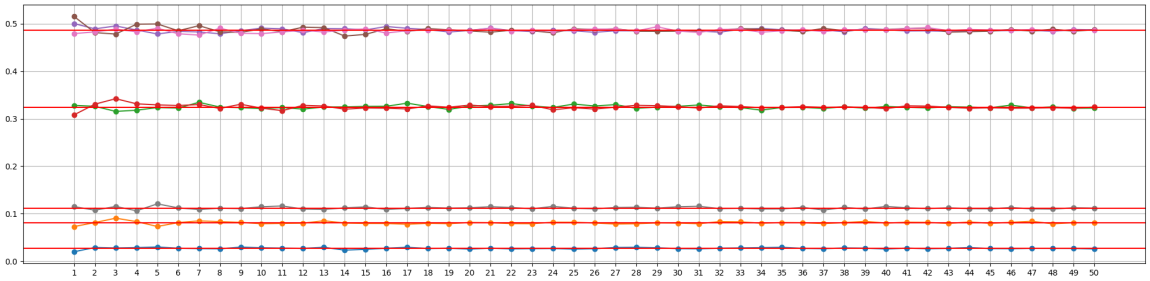

Figura 10: Simulare distribuția mâsluită 1 - toate strategiile

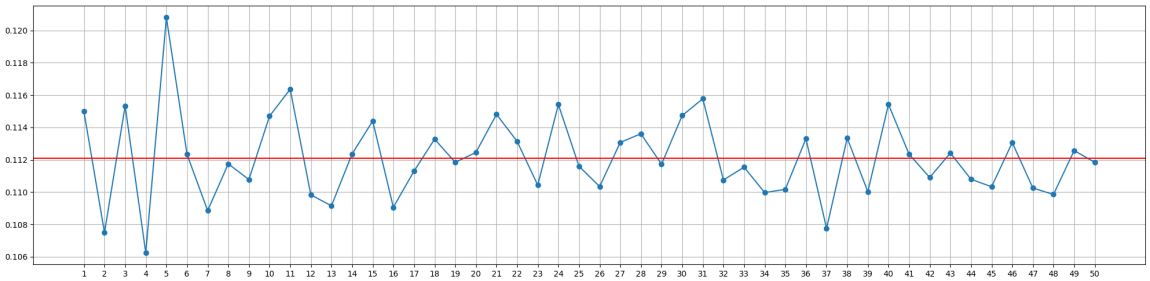

Figura 11: Simulare distribuția mâsluită 1 - strategia mixtă

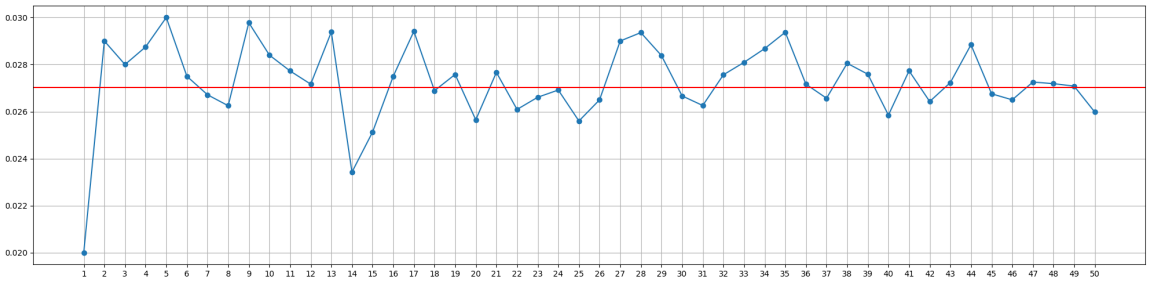

Figura 12: Simulare distribuția mâsluită 1 - strategia single

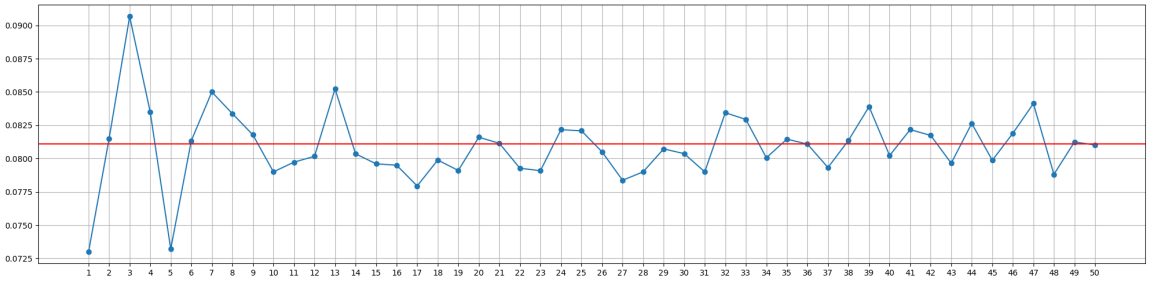

Figura 13: Simulare distribuția mâsluită 1 - strategia rând

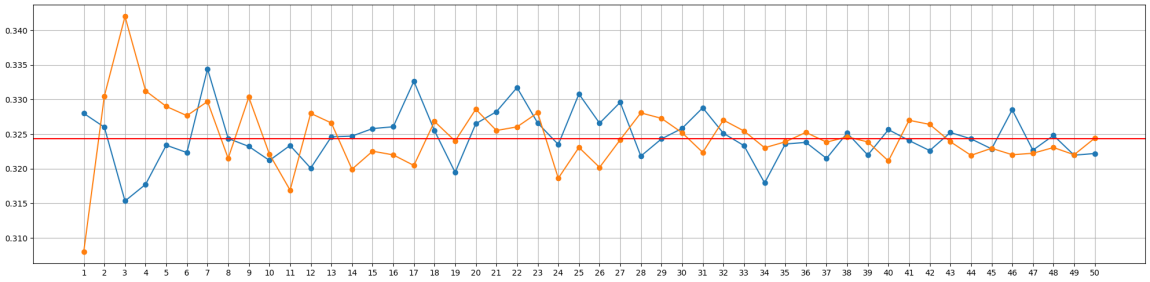

Figura 14: Simulare distribuția mâsluită 1 - strategiile coloană și duzină

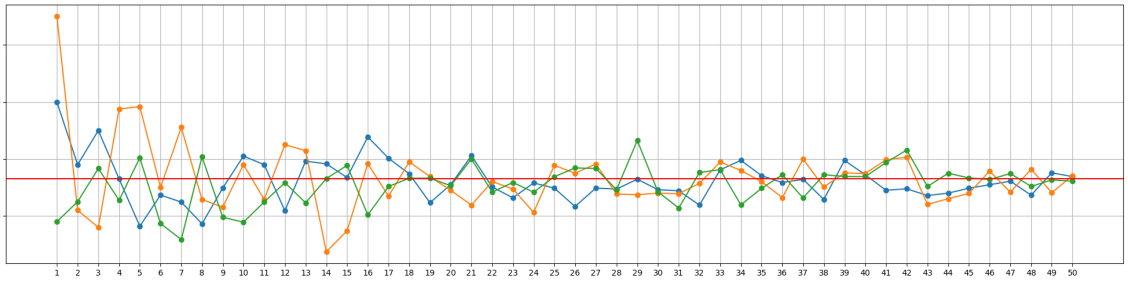

Figura 15: Simulare distribuția mâsluită 1 - strategiile jumătate, paritate și culoare

7 Ruleta mâsluită 2

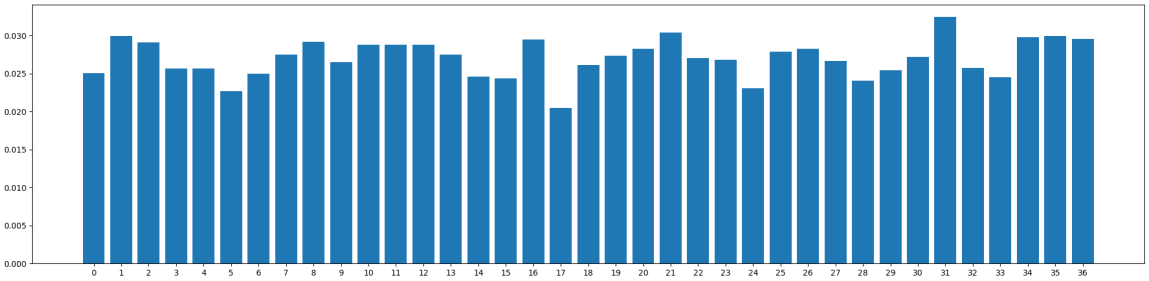

Figura 16: Distribuția mâsluită 2

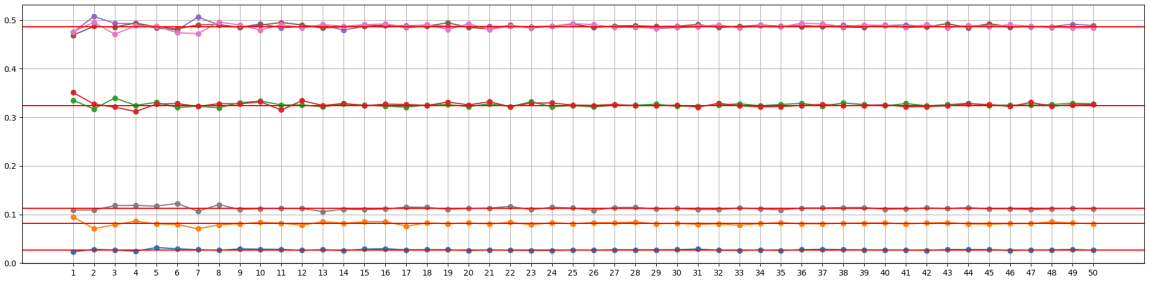

Figura 17: Simulare distribuția mâsluită 2 - toate strategiile

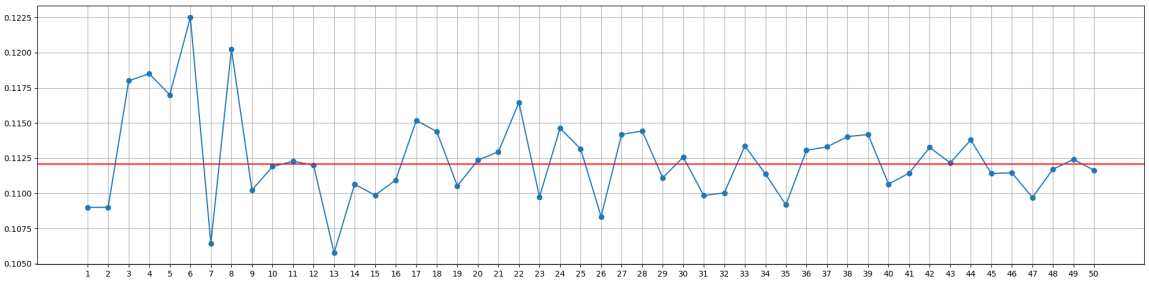

Figura 18: Simulare distribuția mâsluită 2 - strategia mixtă

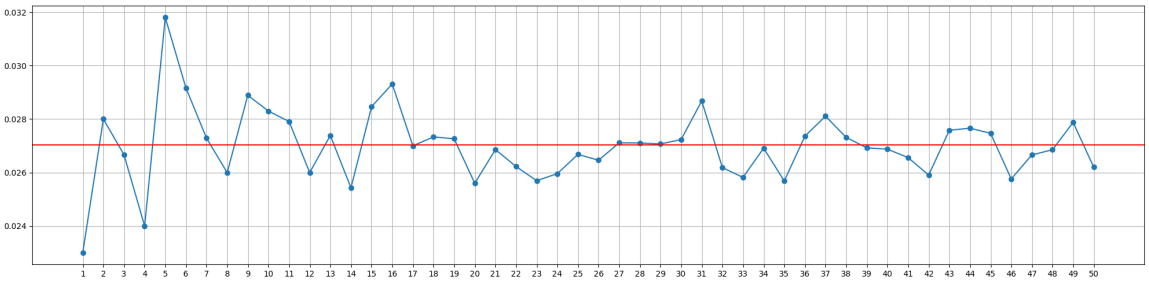

Figura 19: Simulare distribuția mâsluită 2 - strategia single

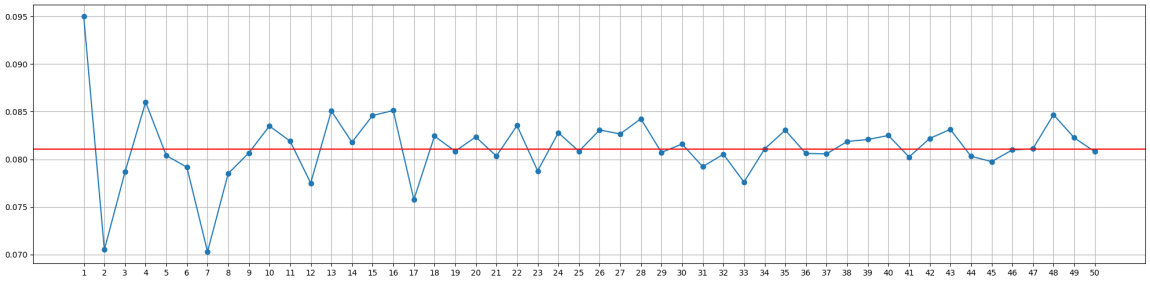

Figura 20: Simulare distribuția mâsluită 2 - strategia rând

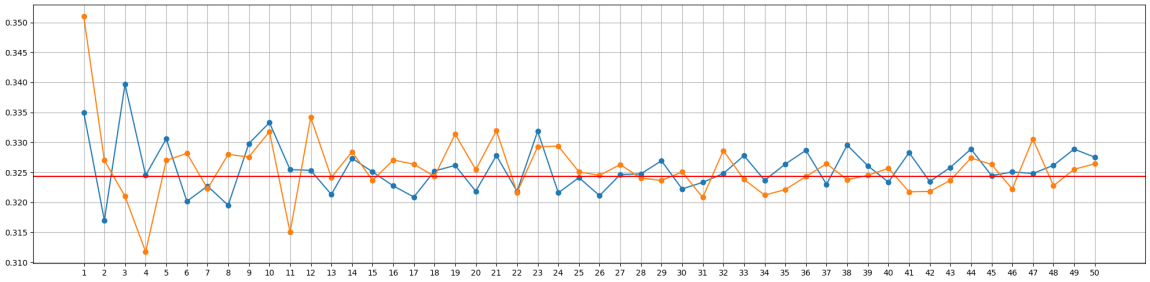

Figura 21: Simulare distribuția mâsluită 2 - strategiile coloană și duzină

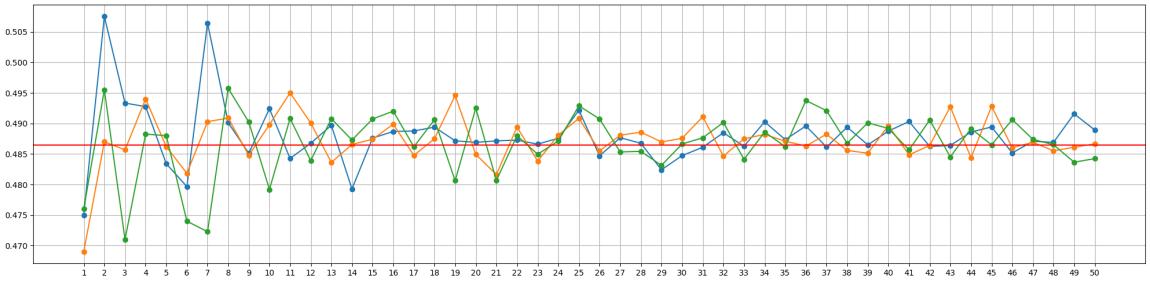

Figura 22: Simulare distribuția mâsluită 2 - strategiile jumătate, paritate și culoare

8 Ruleta mâsluită 3

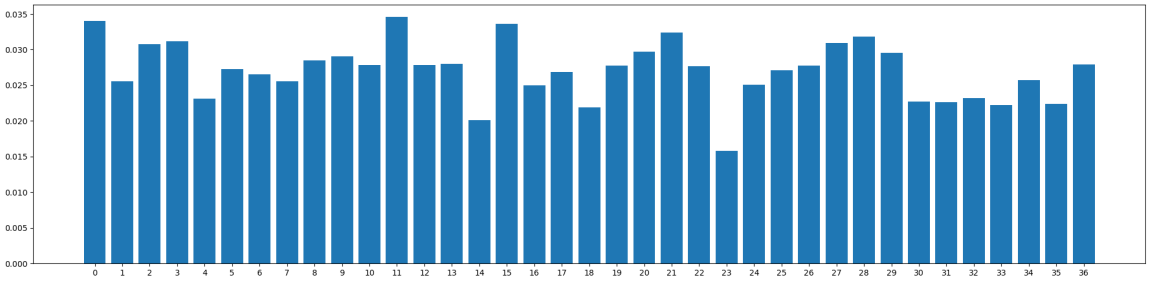

Figura 23: Distribuția mâsluită 3

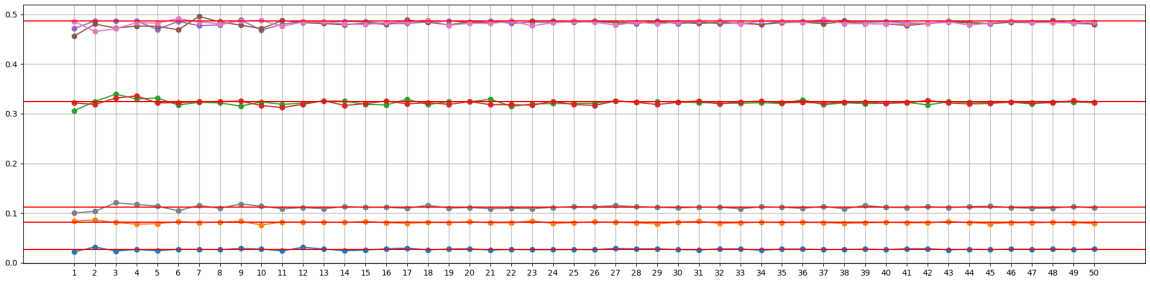

Figura 24: Simulare distribuția mâsluită 3 - toate strategiile

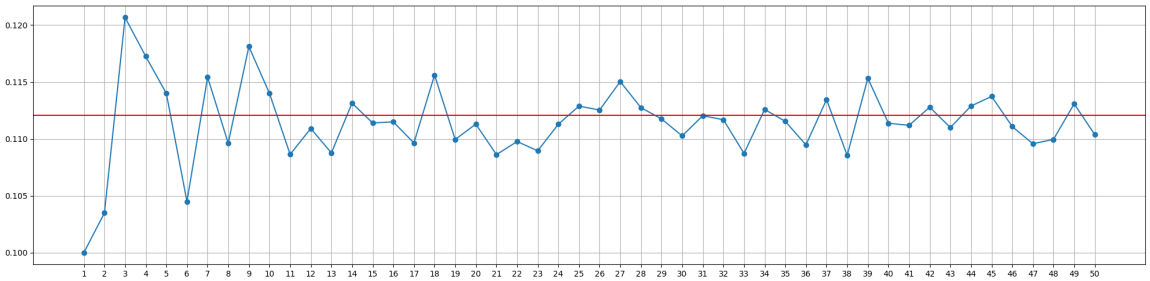

Figura 25: Simulare distribuția mâsluită 3 - strategia mixtă

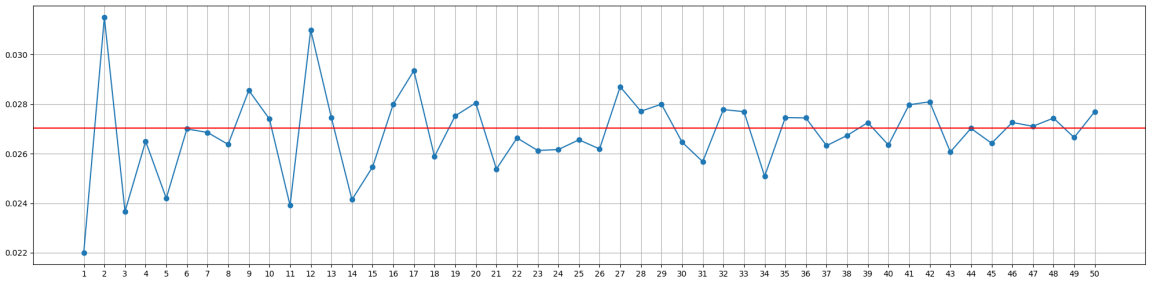

Figura 26: Simulare distribuția mâsluită 3 - strategia single

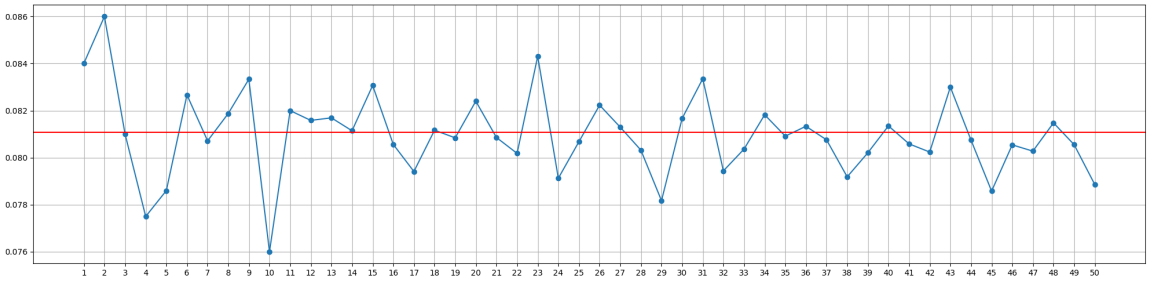

Figura 27: Simulare distribuția mâsluită 3 - strategia rând

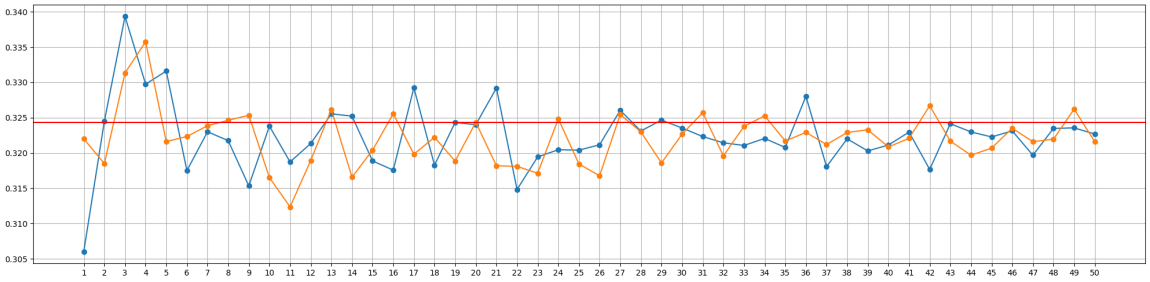

Figura 28: Simulare distribuția mâsluită 3 - strategiile coloană și duzină

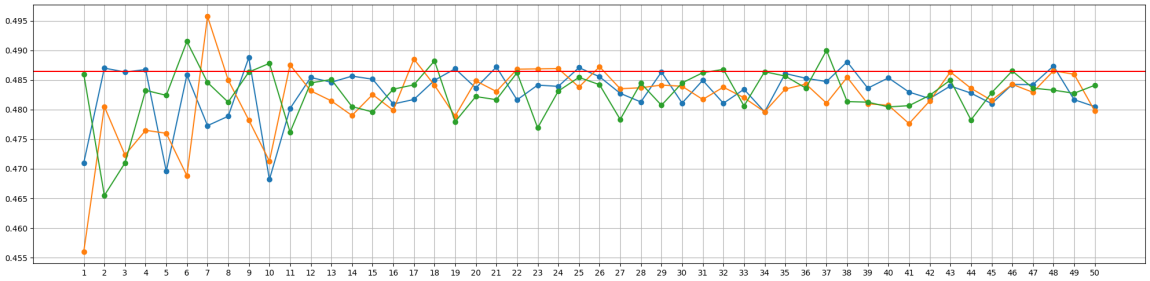

Figura 29: Simulare distribuția mâsluită 3 - strategiile jumătate, paritate și culoare

9 Ruleta mâsluită 4

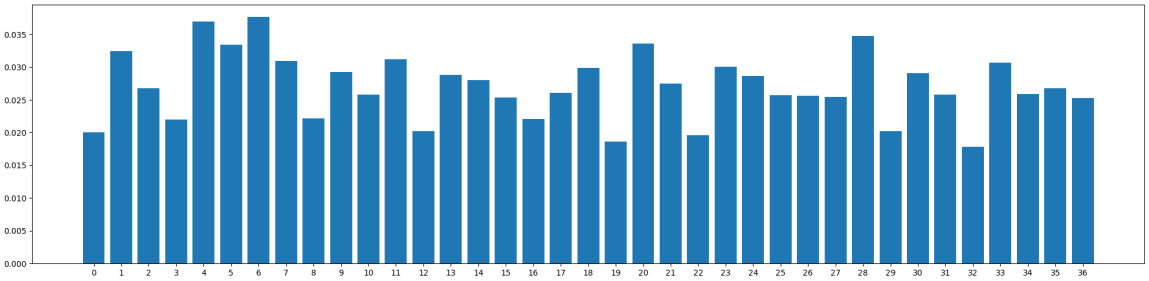

Figura 30: Distribuția mâsluită 4

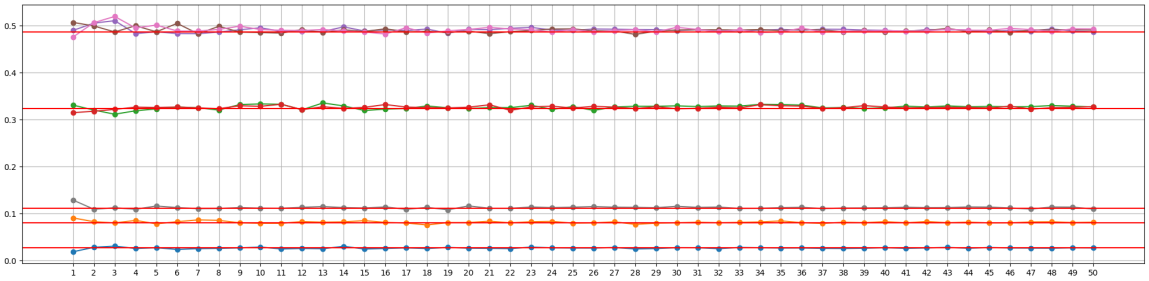

Figura 31: Simulare distribuția mâsluită 4 - toate strategiile

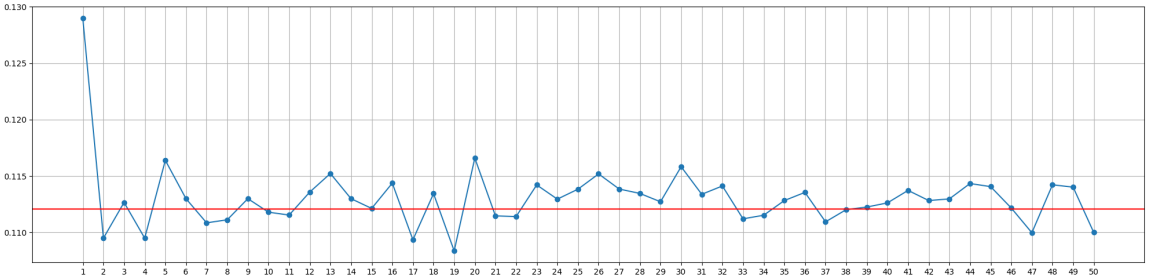

Figura 32: Simulare distribuția mâsluită 4 - strategia mixtă

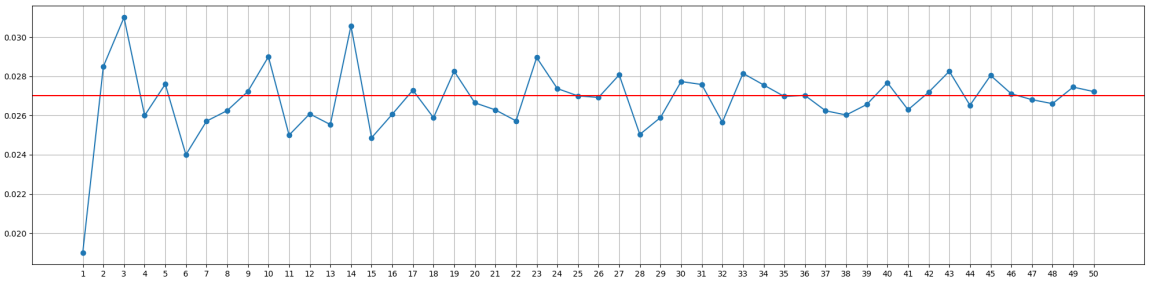

Figura 33: Simulare distribuția mâsluită 4 - strategia single

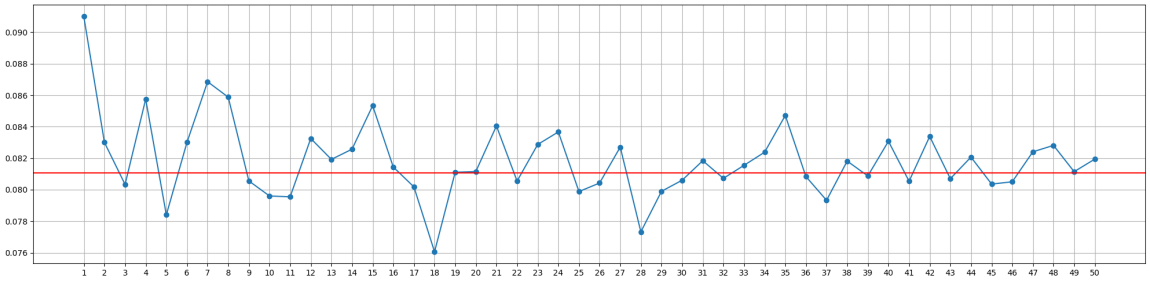

Figura 34: Simulare distribuția mâsluită 4 - strategia rând

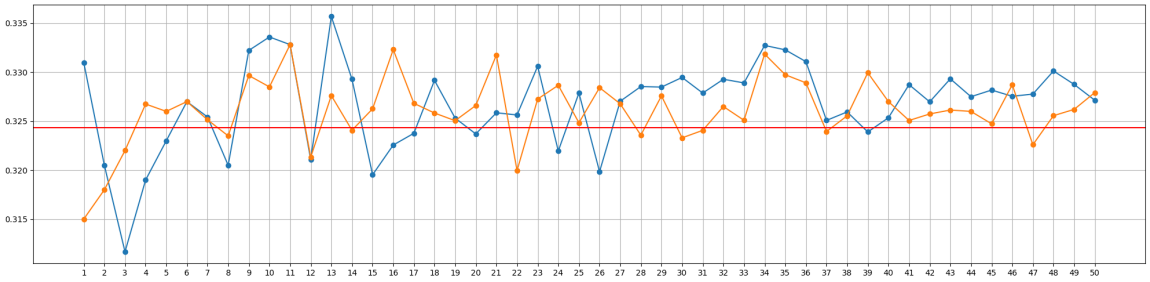

Figura 35: Simulare distribuția mâsluită 4 - strategiile coloană și duzină

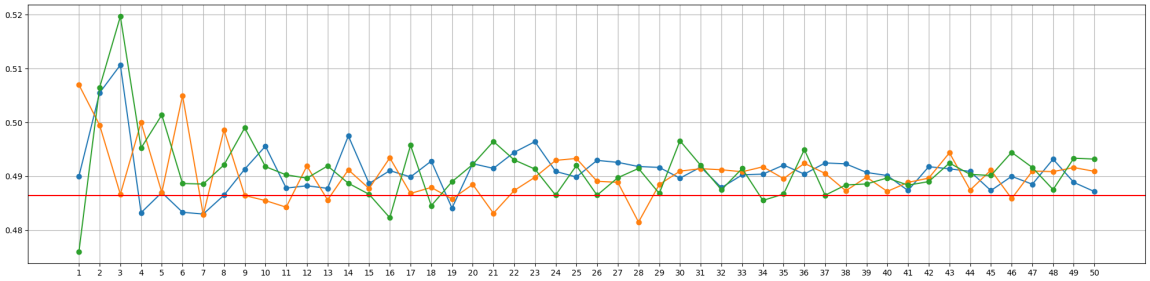

Figura 36: Simulare distribuția mâsluită 4 - strategiile jumătate, paritate și culoare

10 Concluzii

Analizând datele și privind graficele sesizăm câteva idei cheie: cu atât deviația standard a distribuției de probabilități crește, cu atât devine mai evident că ruleta este mâsluită. Din punctul de vedere al datelor generate de simulări, ruleta corectă și ruleta mâsluită 1 sunt aproape inconfundabile, cu mica excepție că în 14, valorile generate tind să scadă un pic sub medie.

În schimb, începând cu ruleta mâsluită 2, putem observa incosecvențe: la ruleta mâsluită 2 în 21 și 22 se poate observa clar că valorile tind să crească peste medie, la ruleta mâsluită 3 în 27, 28 și 29 se observă clar o abater sub medie, iar la ruleta mâsluită 4 situația este opusă, și anume în 35 și 36 valorile sunt deasupra mediei.

Ceea ce în schimb este remarcabil este că dacă comparăm graficele de la strategia mixtă, acestea sunt consecvente în toate cele cinci simulări, deci din perspectiva unui jucător care utilizează o multitudine de strategii ar fi aproape imposibil să determinăm dacă ruleta este mâsluită sau nu.

Bibliografie

[1] Wikipedia, *Roulette — Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/w/index.php?title=Roulette&oldid=1267101116, [Online; accessed 13-January-2025], 2025.