”Лабрадорная” работа 1 по Теории вероятностей

Мочеков Семён

1 Распространение новости. Задача 1.1.

Пусть событие ω — никто не услышал новость дважды за r шагов передачи новости.

Новость передаётся одному человеку

На каждом шаге испытания, тот кто узнал новость должен передавать её тем, кто её ещё не слышал. То есть на k-ом шаге его "жертва" среди $n-k$ жителей, а выбирает он из n жителей, ведь себя выбрать не может. Так как среди жителей выбор происходит равновероятно, приходим к следующей формуле:

$$P(\omega) = \frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \ \dots \ \cdot \frac{n-(r-1)}{n} = \frac{n!}{(n-r)! \cdot n^r} = C_n^r \cdot \frac{r!}{n^r}$$

Новость передаётся m человек

Теперь на каждом шаге число доступных для выбора людей, чтобы никто не услышал новость во второй раз, уменьшается на m. В этом случае мы приходим к задаче о выборе среди групп людей. Если выбор каждого человека равновероятен, то выборы группы из m человек так же равновероятны. Всего всевозможных групп m людей из n выбираемых: C_n^m. Как и прежде, себя выбрать нельзя. Тогда:

$$P(\omega) = \frac{C_n^m}{C_n^m} \cdot \frac{C_{n-m}^m}{C_n^m} \cdot \frac{C_{n-2m}^m}{C_n^m} \cdot \ \dots \ \cdot \frac{C_{n-(r-1)m}^m}{C_n^m} = \frac{(n-m)!^r}{(n-rm)! \cdot n!^{r-1}}$$

Но это всё не правда. На самом деле такое рассуждение верно, если на каждом шаге лишь один глашатай. Более разумно полагать, что каждый в последней услышавшей группе рассказывает новость. То есть надо, чтобы каждый в группе не попадал в уже выбранных людей, то есть на каждом шаге k m^k людей рассказывают каждый m, чтобы не попасть во всех $m, m^2, \dots, m^{k-1}$ знающих. Тогда итог ужасен:

$$P(\omega) = \frac{C_n^m}{C_n^m} \cdot \left(\frac{C_{n-m}^m}{C_n^m} \cdot \frac{C_{n-2m}^m}{C_n^m} \cdot \ \dots \ \cdot \frac{C_{n-m^2}^m}{C_n^m} \right) \cdot \ \dots$$

$$\dots \cdot \left(\frac{C_{n-m^{r-1}-m^{r-2}- \dots -m+1}^m}{C_n^m} \cdot \ \dots \ \cdot \frac{C_{n-m^{r-1}-m^{r-2}- \dots -m+1-m(m^{r-1}-1)}^m}{C_n^m} \right)$$

2 Беспорядок. Задача 2.1.

Пусть θ — в точности m писем попадут в свои конверты.

Каждому письму можно однозначно сопоставить конверт, по-сути это соответствие содержания письма и адресата, если письмо попадёт не в свой конверт, то кто-то получит письмо, которое ему не полагалось. По сути задача сводится к нахождению вероятности того, что лишь m элементов — неподвижные точки, k-ому письму соответствует k-ый конверт. С использованием комбинаторных вычислений задача может быть решена применением понятия *беспорядок* — перестановка без неподвижных точек, чьё число выражается как субфакториал n: $!n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$. Всего же перестановок писем $n!$. Выберем m писем, которые будут на своих местах, остальные $n-m$ будут не на своих. Тогда:

$$P(\theta) = \frac{C_n^m \cdot !(n-m)}{n!} = \frac{\frac{n!}{m! \cdot (n-m)!} \cdot (n-m)! \cdot \sum_{k=0}^{n} \frac{(-1)^k}{k!}}{n!} = \frac{1}{m!} \sum_{k=0}^{n} \frac{(-1)^k}{k!}$$

Данная формула совпадает с первыми n членами разложение e^t в ряд Тейлора при $t = -1$, что даёт нам возможность вычисления вероятности θ при больших n как:

$$P(\theta) \approx \frac{e^{-1}}{m!}$$

3 Товарищи. Задача 3.1.

Пусть ζ — хотя бы два товарища встретятся. Так же пусть $\hat{\zeta}$ — никто из товарищей не встретится.

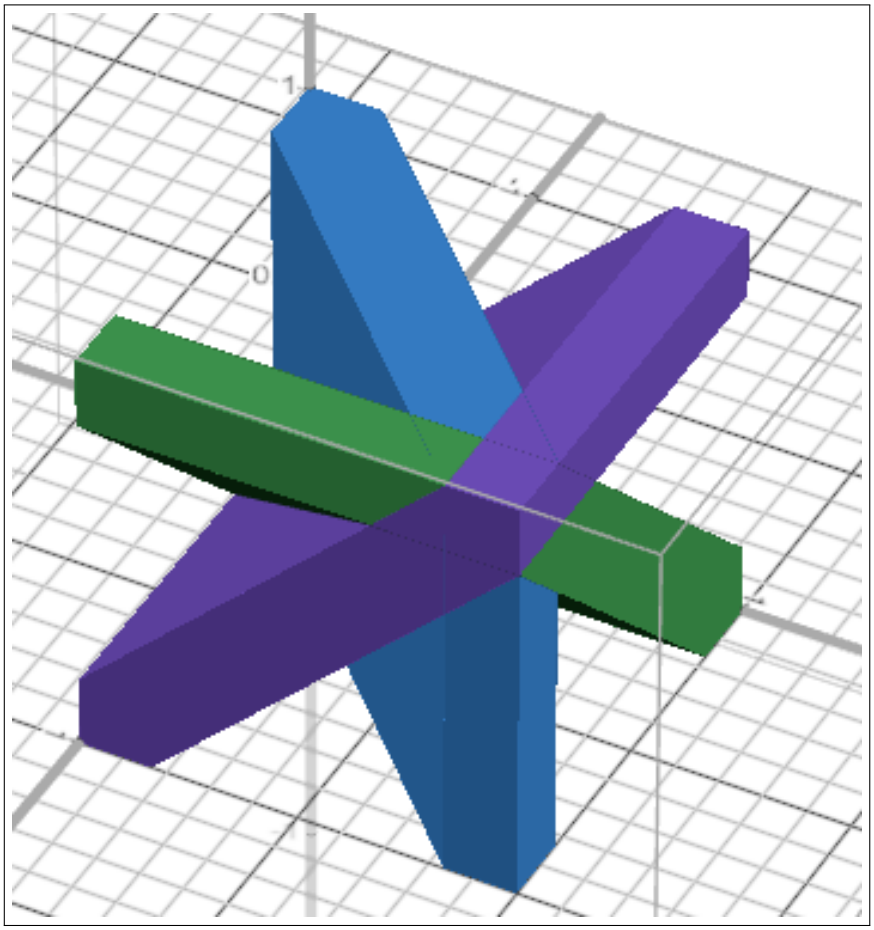
Тогда взаимосвязь вероятностей данных событий: $P(\zeta) = 1 - P(\hat{\zeta})$. Так как всё пространство событий составляет комбинации встреч трёх товарищей и факт любой возможной встречи это всё пространство без пустой встречи — *никто не пришёл на фан встречу*. Найдём $P(\hat{\zeta})$.

Так как приходят они в случайные моменты времени, то эти моменты распределены равномерно на протяжении часа $[0; 1]$, а ожидают друг друга они после прихода не более чем $\tau = \dfrac{1}{6}$ от часа.

Воспользуемся геометрической парадигмой теории вероятности. Пусть T_i — момент прихода i-го товарища. Тогда $\hat{\zeta}$ равносильна:

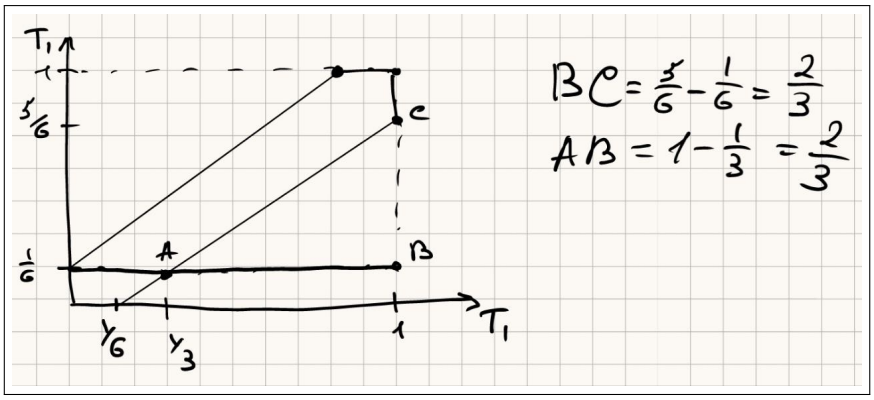
$$\begin{cases} |T_1 - T_2| > \tau \\ |T_1 - T_3| > \tau \\ |T_2 - T_3| > \tau \end{cases}$$

Изобразим эту систему в ДПСК $\mathbb{R}^3$, в осях $OT_1T_2T_3$:



В единичном кубе она задаёт пустое, не цветное пространство. А закрашенное в свою очередь — исходные условия, параллелепипеды по каждой из осей.

Тогда "пустые" пространства — пирамиды, которые в силу симметрии имеют равный объём. Всего таких пирамидок 6. Так как они все симметричны, то рассмотрим одну плоскость, в которой и найдём необходимые для объёма рёбра. То есть $S_{\text{треугольник в плоскости}} = \dfrac{1}{2} \cdot AB \cdot BC, h = \dfrac{2}{3}$



Объём совокупности этих фигур:

$$V_{\hat{\zeta}} = 6 \cdot V_{\text{пирамида}} = 6 \cdot \frac{1}{3} \cdot h \cdot S_{\text{треугольник в плоскости}} = 6 \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{2}{3} = \frac{8}{27} = P(\hat{\zeta})$$

Тогда искомая вероятность:

$$P(\zeta) = 1 - P(\hat{\zeta}) = 1 - \frac{8}{27} = \frac{19}{27}$$

4 Два Бернулли. Задача 4.1.

Исследуем биномиальные распределения.
По определению условной вероятности:

$$P(S_1 = k \mid S_1 + S_2 = m) = \frac{P(S_1 = k \ \cap \ S_1 + S_2 = m)}{P(S_1 + S_2 = m)}$$

Так как серии испытаний независимы, то:

$$P(S_1 = k \mid S_1 + S_2 = m) = \frac{P(S_1 = k) \cdot P(k + S_2 = m)}{P(S_1 + S_2 = m)}$$

Вычислим числитель:

$$P(S_1 = k) = C_n^k \cdot p^k (1-p)^{n-k}$$

$$P(k + S_2 = m) = P(S_2 = m - k) = C_n^{m-k} \cdot p^{m-k} \cdot (1-p)^{n-(m-k)}$$

Знаменатель же, эквивалентен $Binom(n+n, p)$, тогда:

$$P(S_1 + S_2 = m) = C_{2n}^m \cdot p^m \cdot (1-p)^{2n-m}$$

Тогда получим:

$$P(S_1 = k \mid S_1 + S_2 = m) = \frac{C_n^k \cdot C_n^{m-k}}{C_{2n}^m}$$

По итогу получено гипергеометрическое распределение $HG(2n, n, m)$.

5 Вычисления. Задача 5

Реализуем Python скрипт для вычисления требуемых вероятностей.

```python
from scipy import stats
import math
from tabulate import tabulate
from scipy.special import gammaln

def binomial_log_probability(n, p, k):
    log_comb = gammaln(n + 1) - gammaln(k + 1) - gammaln(n - k + 1)
    log_prob = log_comb + k * math.log(p) + (n - k) * math.log(1 - p)
    return log_prob

def binomial_probability(n, p, k):
    if n < 10000:
        prob = math.comb(n, k) * (p ** k) * ((1 - p) ** (n - k))
    else:
        prob = binomial_log_probability(n, p, k)
        return math.exp(prob) if prob > -700 else 0.0
    return prob

def binomial_leq_k(n, p, k):
    return sum(binomial_probability(n, p, i) for i in range(k + 1))

def most_likely_success(n, p):
    lower_bound = p * (n + 1) - 1
    upper_bound = p * (n + 1)

    if upper_bound.is_integer():
        k1 = int(lower_bound)
        k2 = int(upper_bound)

        if 0 <= k1 <= n and 0 <= k2 <= n:
            p1 = binomial_probability(n, p, k1)
            p2 = binomial_probability(n, p, k2)
            return k1 if p1 >= p2 else k2
        elif 0 <= k1 <= n:
            return k1
        else:
            return k2
    else:
        k_star = math.ceil(lower_bound)
        return min(max(0, k_star), n)

def poisson_approximation(n, p, k):
    lam = n * p
    log_prob = -lam + k * math.log(lam)
               - sum(math.log(i) for i in range(1, k + 1))
    return math.exp(log_prob) if log_prob > -700 else 0.0

def poisson_leq_k(n, p, k, strt = 0):
    return sum(poisson_approximation(n, p, i) for i in range(strt, k + 1))
```

```
def local_moivre_laplace(n, p, k):
    q = 1 - p
    npq = n * p * q
    mean = n * p
    return (1 / math.sqrt(2 * math.pi * npq)) *
            math.exp(-0.5 * ((k - mean) ** 2) / npq)

def integral_moivre_laplace(n, p, a, b):
    q = 1 - p
    npq = n * p * q
    a_approx = a - 0.5
    b_approx = b + 0.5

    z_a = (a_approx - n*p) / math.sqrt(npq)
    z_b = (b_approx - n*p) / math.sqrt(npq)

    return stats.norm.cdf(z_b) - stats.norm.cdf(z_a)

def analyze_bernoulli(n_values, p_values):
    results = []
    for n in n_values:
        for p in p_values:
            q = 1 - p
            npq = n * p * q

            lower = max(0, math.floor(n/2 - math.sqrt(npq)))
            upper = min(n, math.ceil(n/2 + math.sqrt(npq)))
            k_star = most_likely_success(n, p)

            exact_interval = binomial_leq_k(n, p, upper) - binomial_leq_k(n, p,
               lower - 1)
            exact_leq_5 = binomial_leq_k(n, p, 5)
            exact_k_star = binomial_probability(n, p, k_star)

            poisson_interval = poisson_leq_k(n, p, upper, lower)
            poisson_le_5 = poisson_leq_k(n, p, 5)
            poisson_k_star = poisson_approximation(n, p, k_star)

            local_interval = sum(local_moivre_laplace(n, p, k)
                                 for k in range(lower, upper+1))
            local_le_5 = sum(local_moivre_laplace(n, p, k)
                             for k in range(6) if not isinstance(
                                local_moivre_laplace(n, p, k), str))
            local_k_star = local_moivre_laplace(n, p, k_star)

            integral_interval = integral_moivre_laplace(n, p, lower, upper)
            integral_le_5 = integral_moivre_laplace(n, p, 0, 5)
            integral_k_star = integral_moivre_laplace(n, p, k_star, k_star)

            results.append({
                'n': n,
                'p': p,
                'interval': f"[{lower}, {upper}]",
                'k_star': k_star,
                'exact_interval': exact_interval,
```

```
                'exact_leq_5': exact_leq_5,
                'exact_k_star': exact_k_star,
                'poisson_interval': poisson_interval,
                'poisson_le_5': poisson_le_5,
                'poisson_k_star': poisson_k_star,
                'local_interval': local_interval,
                'local_le_5': local_le_5,
                'local_k_star': local_k_star,
                'integral_interval': integral_interval,
                'integral_le_5': integral_le_5,
                'integral_k_star': integral_k_star
            })
    return results

def format_value(value):
    if abs(value) < 1e-10:
        return f"{value:.10e}"
    return round(float(value), 10)

def print_results(results, filename="bernoulli_analysis_results.txt"):
    headers = ["n", "p", "Interval", "k*",
               "Exact P(interval)", "Poisson P(interval)", "Local ML P(interval)", "Integral ML P(interval)",
               "Exact P(<=5)", "Poisson P(<=5)", "Local ML P(<=5)", "Integral ML P(<=5)",
               "Exact P(k*)", "Poisson P(k*)", "Local ML P(k*)", "Integral ML P(k*)??"]

    table_data = []
    for r in results:
        row = [
            r['n'], r['p'], r['interval'], r['k_star'],
            format_value(r['exact_interval']),
            format_value(r['poisson_interval']),
            format_value(r['local_interval']),
            format_value(r['integral_interval']),
            format_value(r['exact_leq_5']),
            format_value(r['poisson_le_5']),
            format_value(r['local_le_5']),
            format_value(r['integral_le_5']),
            format_value(r['exact_k_star']),
            format_value(r['poisson_k_star']),
            format_value(r['local_k_star']),
            format_value(r['integral_k_star'])
        ]
        table_data.append(row)

    table_output = tabulate(table_data, headers=headers, tablefmt="grid",
                            floatfmt=".12f",
                            numalign="right")
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(table_output)
    print(f"Saved to {filename}")
```

```
152 if __name__ == ''__main__'':
153     n_values = [100, 1000, 10000]
154     p_values = [0.001, 0.01, 0.1, 0.25, 0.5]
155 
156     results = analyze_bernoulli(n_values, p_values)
157     print_results(results)
```

n	p	Interval	k*	Exact P(interval)	Poisson P(interval)	Local ML P(interval)	Integral ML P(interval)	Exact P(<=5)	Poisson P(<=5)	Local ML P(<=5)	Integral ML P(<=5)	Exact P(k*)	Poisson P(k*)	Local ML P(k*)	Integral ML P(k*)??
100	0.001000000000	[49, 51]	1	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.999999998900	0.999999998700	1.222480853600	0.971172765700	0.090569784500	0.090483741800	0.021901507900	0.102833469300
100	0.010000000000	[49, 51]	1	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.999465466500	0.999405815200	0.942439313900	0.934162938800	0.369729637600	0.367879441200	0.400952077900	0.384697444300
100	0.100000000000	[47, 53]	10	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.057576886500	0.067085962900	0.065686702600	0.066574572200	0.131865346800	0.125110035700	0.132980760100	0.132367665200
100	0.250000000000	[45, 55]	25	0.000010921400	0.000200075900	0.000003191400	0.000003344800	0.000000117000	0.000001397100	0.000003189600	0.000003342900	0.091799691800	0.079522951500	0.092131773200	0.091927444700
100	0.500000000000	[45, 55]	50	0.728746975900	0.563430161800	0.729468341900	0.728667870100	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.079589237400	0.056325006300	0.079788456100	0.079655674600
1000	0.001000000000	[499, 501]	1	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.999411929900	0.999405815200	0.941540142400	0.933286593300	0.368063488300	0.367879441200	0.399141901300	0.383101076300
1000	0.010000000000	[496, 504]	10	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.066139511600	0.067085962900	0.07506233170	0.07590743180	0.125740211100	0.125110035700	0.126792179900	0.126265558800
1000	0.100000000000	[490, 510]	100	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.042016790900	0.039860996800	0.042052208700	0.042032748200
1000	0.250000000000	[486, 514]	250	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.029124105900	0.025222916200	0.029134624800	0.029128151700
1000	0.500000000000	[484, 516]	500	0.703310649200	0.539432634300	0.703387641400	0.703307117200	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.025225018200	0.017838267900	0.025231325200	0.025227120600
10000	0.001000000000	[4996, 5004]	10	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.066991373400	0.067085962900	0.07597395370	0.07681452170	0.125172636700	0.125110035700	0.126219751800	0.125695279900
10000	0.010000000000	[4990, 5010]	100	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.040061805800	0.039860996800	0.040095207800	0.040078339100
10000	0.100000000000	[4970, 5030]	1000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.013296955600	0.012614611300	0.013298076000	0.013297460400
10000	0.250000000000	[4956, 5044]	2500	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.009212844600	0.007978579700	0.009213177300	0.009212972600
10000	0.500000000000	[4950, 5050]	5000	0.687504790500	0.524883167800	0.687512775000	0.687504710000	0.000000000000	0.000000000000	0.000000000000	0.000000000000	0.007978646100	0.005641801800	0.007978845600	0.007978712600

Для подсчёта вероятностей использовались выверенные формулы распределений. Чтобы предотвратить антипереполнение, была адаптирована реализация формул. Для некоторых вычисление, формула биномиального распределения и формула Пуассона, используется логарифмизация результатов, хоть это и увеличивает потерю точности. Интегральная теорема Муавра-Лапласа реализована с применением библиотечной функции распределения.

Касаемо полученных результатов. Ожидаемо некоторые вероятности обозначены нулевыми, так как в рамках конкретных условий, попадание в требуемое крайне маловероятно. То есть в таких ячейках вероятность пренебрежимо мала.

Во многих случаях вероятности в разных вычислениях близки. Однако есть моменты заметных отличий. Они связаны с условиями, требуемыми для приближённого вычисления с помощью теорем Пуассона и Муавра-Лапласа. Так, например, численно, в локальной теореме Муавра-Лапласа, на $P(S_n \leq 5)$ получена вероятность больше 1, в силу накопления суммарной ошибки. Так же в случае с интегральной теоремой Муавра-Лапласа, вычисление $P(S_n = k^*)$, как вычисление интеграла по точке, кажется странным, но чудо змеиной науки вычисляет эту вероятность, так как ясно, что это лишь попытка приближения.