Лабораторная работа №1

Морозов Антон, M32381

Задача 1. Вариант 2 Цепь писем.

Расссматривается генеральная совокупность, состоящая из $(n+1)$ человек. Человек, которого условимся называть "прародителем пишет два письма случайно выбранным адресатам, которые образуют "первое поколение". Те в свою очередь делают то же самое, в результате чего образуется "второе поколение". Вообще каждый из людей, входящих в "r-е поколение посылает два письма случайно выбранным адресатам. Найти вероятность того, что "прародитель"невходит ни в одно из "поколений"с номерами $1, 2, \dots, r$. Найти медиану распределения, предпологая n достаточно большим.

Аналитическое решение:

Представим аналитическое решение в качестве реккуренты, пусть для поколения k и каждого количества людей в нем p, с какой вероятностью такое может случится, тогда получим соотношение для $k+1$ поколения: зафиксируем количество людей в текущем поколении i. Теперь зафиксируем в предыдущем поколении j людей, тогда с какой вероятностью мы могли перейти из $(k, j) \to (k+1, i)$. Понятно, что это будет $P(k, j) \cdot P((k, j) \to (k+1, i))$. $P((k, j) \to (k+1, i))$ - вероятность перехода, тогда как ее посчитать.

У нас должно быть i людей, тогда изначально выберим их из n. Каждому человеку должно прийти хотя бы одно письмо, то есть для каждого человека есть непустое множество людей, которые прислали ему письмо — число стирлинга второго рода. И мы хотим поделить на все возможные варианты — $(n+1)^{2j}$

$$P(k, j) \cdot P((k, j) \to (k+1, i)) = P(k, j) \cdot \frac{S(2j, i)\binom{n}{i}}{(n+1)^{2j}}$$

Тогда получается, что

$$P(k, i) = \sum_{j=1}^{min(n, 2^{i-1})} P(k, j) \cdot \frac{S(2j, i)\binom{n}{i}}{(n+1)^{2j}}$$

```
def expected_task_1(n: int, r: int):
dp = [[Decimal(0)] * (n + 1) for _ in range(r + 1)]
dp[0][1] = Decimal(1)
for k in range(1, r + 1):
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            dp[k][i] += dp[k - 1][j] / dec_pow(n + 1, 2 * j) * stirling(2 * j, i)
        dp[k][i] *= dec_comb(n, i)
return [sum(dp[k]) for k in range(r + 1)]
```

В коде, границы везде до n, так как число стирлинга обнулит, если такое невозможно

Теперь вычисление медианы. При достаточно большом n, мы считаем, что повторения считаются очень редко, поэтому у нас в k поколении будет 2^k человек, тогда всего вершин при r поколениях — $2^{r+1} - 2$.

Для полного дерева, получаем, что для каждой внутренней вершины вероятность, тогда что его дети не прародитель — $\frac{\binom{n}{2}}{\binom{n+1}{2}} = \frac{n-1}{n+1}$. Всего внутренних вершин $2^r - 2 \Rightarrow (\frac{n-1}{n+1})^{2^r-2}$

Теперь найдем медиану $(\frac{n-1}{n+1})^{2^r-2} = \frac{1}{2} \Rightarrow \ln\frac{n-1}{n+1}(2^r-2) = -\ln 2 \Rightarrow (\frac{n-1}{n+1} - 1)(2^r - 2) = -\ln 2$
$\Rightarrow r = \log_2(\frac{n+1}{2}\ln 2 + 2)$ при $n \to \inf \Rightarrow r \to \inf$

Симуляция: Здесь просто проэмулировал процесс, то есть шел по каждому уровню, для каждого человека генерировал двух адресатов, после чего проверял, есть ли среди них $n+1$(или 0 в программе)

```python
def experiments(cur_n: int, n: int, cur_r: int, r: int):
    if cur_r == r:
        return 1
    people = set()
    for i in range(cur_n):
        first = random.randint(0, n)
        second = random.randint(0, n)
        people.add(first)
        people.add(second)
        if first == 0 or second == 0:
            return 0
    return experiments(len(people), n, cur_r + 1, r)

def task_1():
    rs = [1, 3, 5, 7, 8, 10, 12]
    ns = [10, 50, 100]
    for n in ns:
        anss_for_n = expected_task_1(n, max(rs))
        for r in rs:
            cnt = 0
            for _ in range(NUMBER_EXPERIMENTS):
                cnt += experiments(1, n, 0, r)
            print(f'n={n} | r={r} | {cnt / NUMBER_EXPERIMENTS} | exp={anss_for_n[r]}')
```

Ожидаемые результаты и эксперементальные совпадают. Также можно заметить, что при не таких больших значениях n, но медиана по полученной формуле указывает примерно туда, куда нужно

$$n = 10, r = \log_2(\frac{11}{2}\ln 2 + 2) = 2.5391...$$

$$n = 50, r = \log_2(\frac{51}{2}\ln 2 + 2) = 4.2983...$$

$$n = 100, r = \log_2(\frac{101}{2}\ln 2 + 2) = 5.2096...$$

```
n=10  | r=1  | 0.8152 | exp=0.8264462809917355371900826445
n=10  | r=3  | 0.2586 | exp=0.3132412835512816158111831457
n=10  | r=5  | 0.0222 | exp=0.0432559990840245670158773224З
n=10  | r=7  | 0.0014 | exp=0.0032052260118031552431692558 90
n=10  | r=8  | 0.0005 | exp=0.0007811033933208131224492016758
n=10  | r=10 | 0.0    | exp=0.00004207717145931874893037563089
n=10  | r=12 | 0.0    | exp=0.000002142715486914946317608010187

n=50  | r=1  | 0.9579 | exp=0.9611687812379853902345251818
n=50  | r=3  | 0.7648 | exp=0.7633574051221373203427166931
n=50  | r=5  | 0.3288 | exp=0.3519091996153390350979010692
n=50  | r=7  | 0.0544 | exp=0.05740157510537574981842496028
n=50  | r=8  | 0.0123 | exp=0.01604831400148247050832997667
n=50  | r=10 | 0.0007 | exp=0.0008633367283459058468866742673
n=50  | r=12 | 0.0    | exp=0.00003881159404733624494448464571

n=100 | r=1  | 0.9811 | exp=0.9802960494069208901088128615
n=100 | r=3  | 0.8764 | exp=0.8715320468354926401540244351
n=100 | r=5  | 0.5561 | exp=0.5667141885640822807333355646
n=100 | r=7  | 0.1457 | exp=0.1624111223844540690645628868
n=100 | r=8  | 0.0556 | exp=0.05653258089242939824957314725
n=100 | r=10 | 0.0036 | exp=0.003687359579017270869291605376
n=100 | r=12 | 0.0001 | exp=0.0001705184966264069118893918183
```

Задача 2. Вариант 3 Случайная точка имеет равномерное распределение в правильном n-угольнике. Найти P_n вероятность P_n, что точка A находится ближе к границе многоугольника, чем к его диагоналям. Найти числа C, a, что

$$P_n = Cn^{\alpha}(1 + o(1)), n \to \inf$$

Аналитическое решение: 1. Диагональ – отрезок соединяющий любые не смежные вершины.

Заметим, что если мы соединим диагонали, которые соединяют точки через одну(Рис. 1), то если точка окажется в синий области, то чтобы дойти до стороны, если придется пересеч диагональ, а значит, что идти до диагонали ближе. Теперь можно заметить, что многоугольник можно независимо разбить на n теугольников, таких, что вершины - центр многоугольника и сторона (Это можно сделать, потому что каждй белый треугольник полностью лежит в большом)

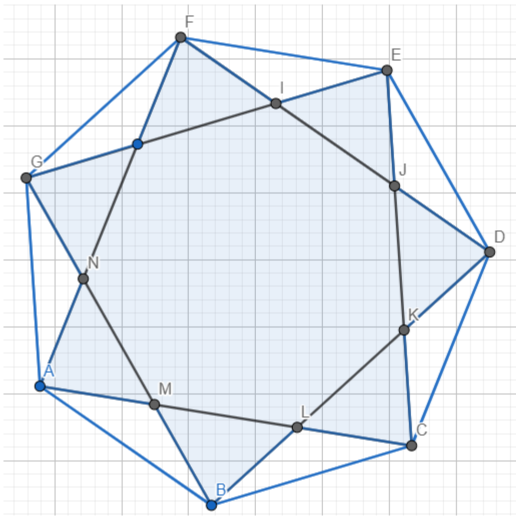

Рис. 1: Диагонали

2. Зафиксируем сторону многоугольника, к которой точка, тогда ближайшие диагонали к нему, это если мы возьмем 4 вершины подряд и соеденим через одну, как на рис. 2. Мы соединили EF, GF. Все остальные диагонали будут выше этих и будут лежать дальше от стороны GD.

Пересечение EF, GF принадлежит высоте теругольника AH и пусть равна J (Это легко следует из побобия треугольников и теоремы Чевы). Тогда если мы рассмотрим треугольник GDJ, то можно заметить, что основание – это стороная, а стороны диагонали, и все что все это треугольника, но в треугольнике ABG (то есть в черырехугольнике $AGJD$ лежит дальше от стороны, чем от диагоналей). Тогда мы знаем, что биссетрисса равноудаленна от сторон, тогда в треугольнике DGJ нужно провести биссектрисы из уголов D, G и любая точка ограниченная биссектрисами и стороной, будет ближе к стороне.

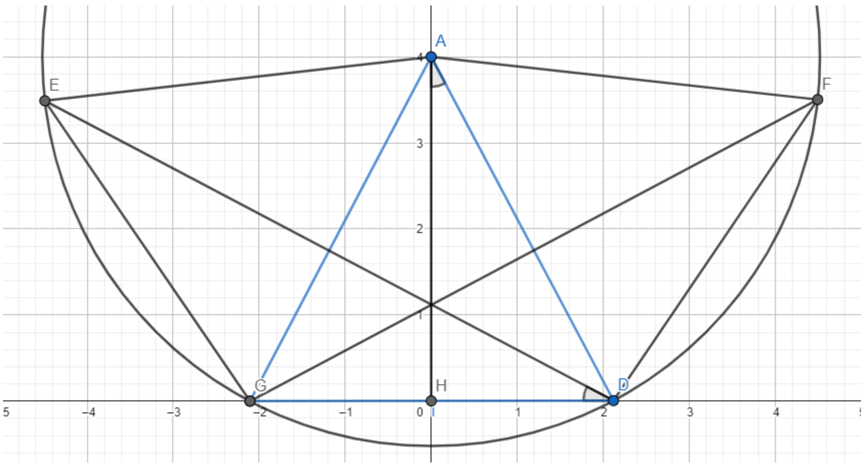

Рис. 2: Разбиение многоугольника

3. Теперь применем наш пункт 1 и посчитаем геометрическую вероятность. (Рис. 2)

Из-за симметричности относительно высоты, можно свести задачу к половинке треугольника. Как мы знаем теперь, чтобы посчитать вероятность нужно найти отношение площадей фиолетового треуголька и S_{ADH}.

Угол $ADG = EDG = \frac{\pi}{n}$, как вписанный и половина центрального углы. По тригонометрическим соображениям $\tan\frac{\pi}{2n} = \frac{|IH|}{a/2}$

$$S_{HDI} = \frac{1}{2}\frac{a}{2}|IH| = \frac{a^2}{4}\tan\frac{\pi}{2n}$$

$$S_{AHD} = \frac{1}{2}\frac{a}{2}|AH| = \frac{a^2}{4}\frac{1}{\tan\frac{\pi}{n}}$$

$$P_n = \frac{S_{HDI}}{S_{AHD}} = \frac{\frac{a^2}{4}\tan\frac{\pi}{2n}}{\frac{a^2}{4}\frac{1}{\tan\frac{\pi}{n}}} = \tan\frac{\pi}{2n}\tan\frac{\pi}{n}$$

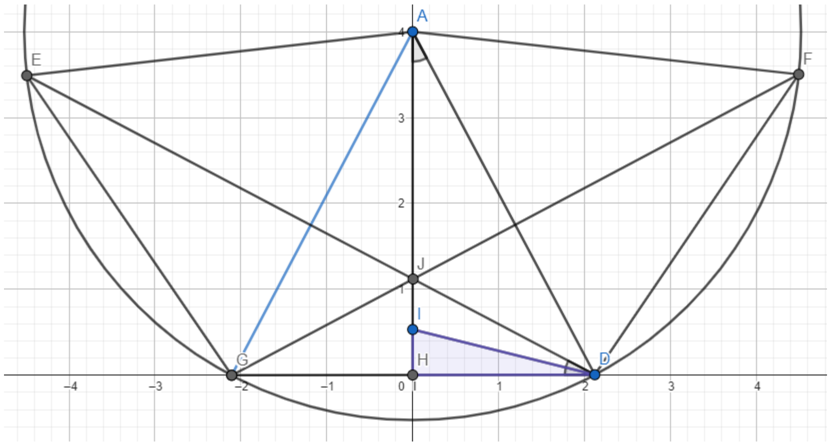

Рис. 3: Отношение площадей

$$P_n = \tan\frac{\pi}{2n}\tan\frac{\pi}{n} \underset{n\to\inf}{=} (\frac{\pi}{2n} + o(1))(\frac{\pi}{n} + o(1)) = \frac{\pi^2}{2}n^{-2}(1 + o(1))$$

Симуляция

Чтобы симулировать такой процесс и было проще, воспрользуемся той же идеей и сведем генерацию случайной точки в n-угольнике к треугольнику. Тогда предположим, я генерирую точку в треугольнике (равнобедренном с правильными углами), то я могу выбрать равномерно половину треугольника и равномерно выбрать один из n треугольников в исходном многоугольнике.

Теперь чтобы сгенерировать в треугольнике, считаем $\frac{a}{2} = 1$, тогда сгенерирую $x \in [0, 1]$, потом сгенерирую $y \in [0, \frac{1-x}{\tan\pi/n}]$ (Рис. 3)

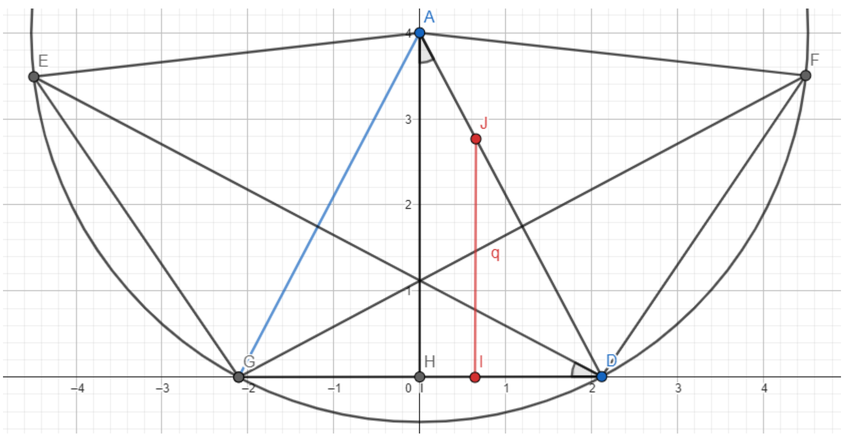

Рис. 4: Генерация случайной токи

Для фиксированного n брал 10^5 экспериментов.

```python
def expected_task_2(n: int):
    return math.tan(math.pi / n) * math.tan(math.pi / 2 / n)


def dist_line_point(A: float, B: float, x: float, y: float):  # line: x/A + y/B = 1
    return math.fabs(1 / A * x + 1 / B * y - 1) / math.sqrt(1 / A ** 2 + 1 / B ** 2)


def triangle_n(n: int):
    cnt = 0
    for _ in range(NUMBER_EXPERIMENTS):
        x_point = random.random()
        max_y_point = (1 - x_point) / math.tan(math.pi / n)
        y_point = random.random() * max_y_point
        if y_point < dist_line_point(1, math.tan(math.pi / n), x_point, y_point):
            cnt += 1
    return cnt / NUMBER_EXPERIMENTS


def task_2():
    for n in range(3, 20):
        print(f'n={n}: {triangle_n(n)} | expected={expected_task_2(n)}')
```

Результаты совпали с ожидаемыми аналитическими.

```
n=3: 1.0     | expected=0.9999999999999997
n=4: 0.41396 | expected=0.414213562373095
n=5: 0.23647 | expected=0.23606797749978964
n=6: 0.15275 | expected=0.15470053837925152
n=7: 0.11021 | expected=0.10991626417474237
n=8: 0.08296 | expected=0.08239220029239397
n=9: 0.06432 | expected=0.06417777247591214
n=10: 0.05245 | expected=0.0514622242382672
n=11: 0.04251 | expected=0.04221711622640544
n=12: 0.03509 | expected=0.03527618041008304
n=13: 0.03062 | expected=0.0299278309497274
n=14: 0.02642 | expected=0.0257168632725539
n=15: 0.02159 | expected=0.02234059486502927
n=16: 0.01864 | expected=0.019591158208318336
n=17: 0.01765 | expected=0.017321837516788223
n=18: 0.01527 | expected=0.015426611885744986
n=19: 0.01399 | expected=0.013827282710936875
```

Задача 3. Вариант 1 Пусть имеются две независимые серии испытаний Бернулли на n опытов в каждой с вероятностью успеха p, S_i – количество успехов в n испытаниях в i-ой серии. Найти вероятность $P(S_1 = k|S_1 + S_2 = m)$.

Аналитическое решение:

Распишем условную вероятность по определению

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

Так как серии испытаний независимы, то по определению независимости вероятность пересечения равна произведению вероятностей:

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

$P(S_1 = k) = \binom{n}{k} p^k (1-p)^{n-k}$ – выбираю k позиций для успеха, и умножаю на соответствующие вероятности. Аналогично $P(S_2 = m - k) = \binom{n}{m-k} p^{m-k} (1-p)^{n-m+k}$.

$$P(S_1 + S_2 = m) = \sum_{k=0}^{m} P(S_1 = k) \cdot P(S_2 = m - k) = \sum_{k=0}^{m} \binom{n}{k} p^k (1-p)^{n-k} \cdot \binom{n}{m-k} p^{m-k} (1-p)^{n-m+k} =$$

$$= p^m (1-p)^{2n-m} \cdot \sum_{k=0}^{m} \binom{n}{k} \binom{n}{m-k} = \binom{2n}{m} p^m (1-p)^{2n-m}$$

Подставим в формулу:

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

Симуляция

Для симуляции я фиксировал n, p, m, дальше брал массив длинной $2n$, который содержит m единиц, и перемешивал его, после чего считал количество единиц в начале. Чтобы усреднить результат, для фиксированных параметров n, p, m брал 10^5 экспериментов. Так же я взял значение $n = 10$, чтобы в целом нормально быстро считались сочетания. $p = [10^{-3}, 0.1, 0.5]$ чтобы проверить независимость от p. $m = [1, 3, 5, 10, 15, 20]$

```python
def task_3():
    ns = [10]
    ps = [0.001, 0.1, 0.5]
    ms = [1, 3, 5, 10, 15, 20]
    for n, p, m in itertools.product(ns, ps, ms):
        a = [1] * m + [0] * (2 * n - m)
        success = [0] * 2 * n
        for _ in range(NUMBER_EXPERIMENTS):
            random.shuffle(a)
            cnt = sum(a[i] for i in range(n))
            success[cnt] += 1
        for k in range(m):
            print(f'n={n}, p={p}, m={m} P(S1={k}|S1+S2={m})={success[k]/NUMBER_EXPERIMENTS} '
                  f'| expected = {expected_task_3(n, p, m, k)}')
```

Эксперимент подтвердил результаты, полученные аналитически

```
n=10, p=0.001, m=1 P(S=0|S+S=1) = 0.50332 | expected = 0.5
n=10, p=0.001, m=3 P(S=0|S+S=3) = 0.10312 | expected = 0.10526315789473684
n=10, p=0.001, m=3 P(S=1|S+S=3) = 0.39750 | expected = 0.39473684210526316
n=10, p=0.001, m=3 P(S=2|S+S=3) = 0.39526 | expected = 0.39473684210526316
n=10, p=0.001, m=5 P(S=0|S+S=5) = 0.01594 | expected = 0.016253869969040248
...
n=10, p=0.1, m=1 P(S=0|S+S=1) = 0.49880 | expected = 0.5
n=10, p=0.1, m=3 P(S=0|S+S=3) = 0.10501 | expected = 0.10526315789473684
n=10, p=0.1, m=3 P(S=1|S+S=3) = 0.39338 | expected = 0.39473684210526316
n=10, p=0.1, m=3 P(S=2|S+S=3) = 0.39623 | expected = 0.39473684210526316
...
```

Задача 4 Рассмотрите схемы Бернулли при $n \in \{10, 100, 1000, 10000\}$ и $p \in \{0.001, 0.01, 0.1, 0.25, 0.5\}$ и рассчитайте точные вероятности (где это возможно) $P(S_n \in [n/2 - \sqrt{npq}, n/2 + \sqrt{npq}])$, S_n– количество успехов в n испытаниях, и приближенную с помощью одной из предельных теорем. Сравните точные и приближенные вероятности. Объясните результаты.

Решение

1. Найдем точную вероятность. Так как испытания незвисимы то просуммируем все вероятность вида $P(S_n = k), k \in [n/2 - \sqrt{npq}, n/2 + \sqrt{npq}] \cap \mathbf{Z}^+ = [l, r]$.

$$P(S_n = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$P(S_n \in [n/2 - \sqrt{npq}, n/2 + \sqrt{npq}]) = \sum_{k=l}^{r} \binom{n}{k} p^k (1-p)^{n-k}$$

Ну и так как в python была проблема с переполнением float64, использую Decimal

```
def dec_factorial(n: int):
    res = Decimal(1)
    for i in range(1, n + 1):
        res = res * Decimal(i)
    return res


def dec_comb(n, k):
    return (dec_factorial(n) / dec_factorial(k)) / dec_factorial(n - k)


def accurate_result(n, p):
    q = (1 - p)
    left_bound = math.ceil(n / 2 - math.sqrt(n * p * q))
    right_bound = math.floor(n / 2 + math.sqrt(n * p * q))
    prob = Decimal(0)
    for i in range(left_bound, right_bound + 1):
        prob += dec_comb(n, i) * Decimal(p) ** i * Decimal(q) ** (n - i)
    return prob
```

2. Найдем приближенное решение, с помощью интегральной теоремы Муавра-Лапласа.

$$P(x_1 \leqslant \frac{S_n - np}{\sqrt{npq}} \leqslant x_2) \approx \frac{1}{\sqrt{2\pi}} \int_{x_1}^{x_2} e^{\frac{-t^2}{2}} dt$$

$$P(x_1 \leqslant \frac{S_n - np}{\sqrt{npq}} \leqslant x_2) = P(x_1\sqrt{npq} + np \leqslant S_n \leqslant x_2\sqrt{npq} + np)$$

$$x_1\sqrt{npq} + np = n/2 - \sqrt{npq} \Rightarrow x_1 = \frac{\sqrt{n}(1-2p)}{2\sqrt{pq}} - 1$$

$$x_2\sqrt{npq} + np = n/2 + \sqrt{npq} \Rightarrow x_2 = \frac{\sqrt{n}(1-2p)}{2\sqrt{pq}} + 1$$

Чтобы приближение было хорошее, количество испытаний должно быть достаточно большим(желательно, чтобы $npq > 10$)

```
def approximate_result(n, p):
    c = (math.sqrt(n) * (1 - 2 * p)) / (2 * math.sqrt(p * (1 - p)))
    x1 = c - 1
    x2 = c + 1
    return NormalDist().cdf(x2) - NormalDist().cdf(x1)
```

```
n=10, p=0.001
Is npq > 10? False
Expected=2.507425174812597729835007904E-13 | Approximate=0.0
Error: 2.507425174812597729835007904E-13

n=10, p=0.01
Is npq > 10? False
Expected=2.396494925748000141934599054E-8 | Approximate=0.0
Error: 2.396494925748000141934599054E-8

n=10, p=0.1
Is npq > 10? False
Expected=0.001488034800000000596573790368 | Approximate=0.0006490249663507752
Error: 0.0008390098336492253734398394372

n=10, p=0.25
Is npq > 10? False
Expected=0.2206192016601562500 | Approximate=0.20211670793013647
Error: 0.01850249373001977559738406853

n=10, p=0.5
Is npq > 10? False
Expected=0.6562500000 | Approximate=0.6826894921370856
Error: 0.02643949213708562950841951533

...

n=10000, p=0.001
Is npq > 10? False
Expected=1.065803565256637072748035961E-11985 | Approximate=0.0
Error: 1.065803565256637072748035961E-11985

n=10000, p=0.01
Is npq > 10? True
Expected=2.167528939583351163711414237E-6996 | Approximate=0.0
Error: 2.167528939583351163711414237E-6996

n=10000, p=0.1
Is npq > 10? True
Expected=1.035946198270798702659363017E-2192 | Approximate=0.0
Error: 1.035946198270798702659363017E-2192

n=10000, p=0.25
Is npq > 10? True
Expected=5.541872325203998434898533646E-607 | Approximate=0.0
Error: 5.541872325203998434898533646E-607

n=10000, p=0.5
Is npq > 10? True
Expected=0.6875047904893208575646950320 | Approximate=0.6826894921370856
Error: 0.004815298352235228056275516668
```

Результаты в целом очень близки, при условии, что количетсво результатов большое и $npq > 10$. Из-за того, что данные считались в разных типах Decimal и float64, то некоторые значения, которые не влезают в float64 просто равняются 0