Горелкина
РК6-32Б
Вариант №4

Домашнее задание №2 по курсу теории вероятностей и математической статистики. Часть №1. Дискретные случайные величины.

Исходные данные

R_1	G_1	B_1	R_2	G_2	B_2	R_3	G_3	B_3
11	6	8	10	5	10	8	5	6

Задача 1. Рассматривается извлечение шаров с возвращением из первой корзины. Выполняется серия из n экспериментов, подсчитывается число k извлечений красных шаров.

1. Построить графики вероятности $P(k)$. Графики строятся для числа опытов $n = 6,\ 9,\ 12$ с расчётом вероятностей по формуле Бернулли.

2. Для $n = 6$ также строится график функции распределения $F(x)$.

3. Для $n = 25,\ 50,\ 100,\ 200,\ 400,\ 1000$ строится огибающая графика $P(k)$, при этом для каждого графика рассчитываются не менее 7 точек с использованием локальной теоремы Муавра-Лапласа.

4. Построить график вероятности того, что абсолютное число извлечений красных шаров отклонится от математического ожидания не более, чем на R_1. При построении графика использовать $n = 25,\ 50,\ 100$.

5. Построить график вероятности того, что относительное число извлечений красных шаров отклонится от математического ожидания не более, чем на $\dfrac{R_1}{R_1 + G_1 + B_1}$. При построении графика использовать $n = 100,\ 200,\ 400$.

6. Рассчитать допустимый интервал числа успешных испытаний k (симметричный относительно математического ожидания), обеспечивающий попадание в него с вероятностью $P = \dfrac{R_1}{R_1 + G_1 + B_1}$ при $n = 1000$.

7. Построить график зависимости минимально необходимого числа испытаний n, для того, чтобы обеспечить вероятность появления не менее, чем $N_1 = R_1 + G_1 + B_1$ красных шаров с вероятностями $P = 0.7,\ 0.8,\ 0.9,\ 0.95$.

Решение:

I. Для построения графиков запишем в общем виде функцию $P(k)$ по формуле Бернулли для произвольного числа испытаний n, а затем построим необходимые частные случаи.

$$P(k) = C_n^k \cdot \left(\frac{R_1}{R_1 + G_1 + B_1}\right)^k \cdot \left(1 - \frac{R_1}{R_1 + G_1 + B_1}\right)^{n-k} =$$

$$= C_n^k \cdot \left(\frac{11}{25}\right)^k \cdot \left(1 - \frac{11}{25}\right)^{n-k} = C_n^k \cdot \left(\frac{11}{25}\right)^k \cdot \left(\frac{14}{25}\right)^{n-k}$$

Мы получили выражение для $P(k)$. Для построения графиков будет использован математический пакет $MATLAB$. Скрипт и графики с таблицами значений функций вероятностей находятся в приложении 1.1.

II. По определению функции распределения $F(x) = \sum P_i$. Значения P_i уже найдены. Для построения графика будет использован математический пакет $MATLAB$. Скрипт и график с таблицей значений функции распределения находятся в приложении 1.1.

III. Локальная теорема Муавра-Лапласа:

$P(X = k) \approx \frac{1}{\sqrt{npq}}\varphi\left(\frac{k - np}{\sqrt{npq}}\right)$. Для построения огибающих воспользуемся центральной предельной теоремой, из которой следует: $Bin(n, p) \approx N(np, npq)$. $N = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-(x-\mu)^2}{2\sigma^2}}$ - распределение Гаусса.

Дальнейшие вычисления и графики находятся в приложении 1.2. Для их построения был использован язык программирования $Python$ 3.8.1.

IV. Вероятность того, что случайная величина будет попадать в некоторый интервал (k_1, k_2) при достаточно большом количестве испытаний n находится с помощью интегральной теоремы

Муавра-Лапласа и равна $P_n(k_1, k_2) \approx \Phi\left(\frac{k_2 - np}{\sqrt{npq}}\right) - \Phi\left(\frac{k_1 - np}{\sqrt{npq}}\right)$. В нашем случае требуется построить график $P_n(|k - M(k)| \le R_1)$. Данная функция будет зависеть от n, а график построим по трём точкам. Известно, что вероятность схемы Бернулли подчиняется биномиальному закону распределения. Поэтому $M(k) = n \cdot p$ и $P_n(|k - n \cdot p| \le R_1) = 2 \cdot \Phi\left(\frac{R_1}{\sqrt{npq}}\right)$.Расчёты необходимых вероятностей реализованы с помощью языка программирования *Python* 3.8.1 и таблицы со значениями функции Лапласа и находятся в приложении 1.2.

V. Аналогично пункту IV воспользуемся интегральной теоремой Муавра-Лапласа. Только в данном случае, так как мы берем случайную величину $\frac{k}{n}$, то и матожидание возьмём $M\left(\frac{k}{n}\right) = \frac{M(k)}{n} = \frac{n \cdot p}{n} = p$. Получаем, что

$P_n\left(\left|\frac{k}{n} - p\right| \le \frac{R_1}{N_1}\right) = 2 \cdot \Phi\left(\frac{n \cdot R_1}{N_1 \cdot \sqrt{npq}}\right)$. Расчёты необходимых вероятностей реализованы с помощью языка программирования *Python* 3.8.1 и таблицы со значениями функции Лапласа и находятся в приложении 1.2.

VI. Пусть у нас имеется некоторый интервал $(M(k) - \alpha, M(k) + \alpha)$, симметричный относительно матожидания. Тогда вероятность попадания величины k в этот интервал равна (по интегральной теореме Муавра-Лапласа): $P_n((M(k) - \alpha \le k \le M(k) + \alpha) = P_n(|k - M(k)| \le \alpha) = P_n(|k - n \cdot p| \le \alpha) = P_n\left(\left|\frac{k}{n} - p\right| \le \frac{\alpha}{n}\right) = 2 \cdot \Phi\left(\frac{\alpha}{\sqrt{p \cdot q \cdot n}}\right)$. По условию $P_n = \frac{11}{25} = 2 \cdot \Phi\left(\frac{\alpha}{\sqrt{\frac{11}{25} \cdot \frac{14}{25} \cdot 1000}}\right)$.

Тогда $\Phi\left(\frac{25 \cdot \alpha}{10 \cdot \sqrt{11 \cdot 14 \cdot 10}}\right) = \frac{11}{50} = 0.22$. По таблице значений функции Лапласа находим, что аргумент равен 0.58. То есть

$\dfrac{\sqrt{385}\cdot\alpha}{308} = 0.58 \Leftrightarrow \alpha = \dfrac{58\cdot\sqrt{385}}{125} \approx 9.104$. Таким образом, получаем интервал $(M(k) - 9.104, M(k) + 9.104)$, $M(k) = n\cdot p = \dfrac{1000\cdot 11}{25} = 440$. Искомый интервал равен $(430.896,\ 449.104)$.

VII. Снова воспользуемся интегральной теоремой Муавра-Лапласа. Будем находить следующую вероятность $P_n(N_1 \le k)$, которая должна принимать указанные в условии значения. Так как испытаний должно быть максимум n, то можем переписать вероятность следующим образом $P_n(N_1 \le k \le n) = \Phi(x_2) - \Phi(x_1)$, $x_2 = \dfrac{n - n\cdot p}{\sqrt{n\cdot p\cdot q}}$, $x_1 = \dfrac{25 - n\cdot p}{\sqrt{n\cdot p\cdot q}}$. Понятно, что $x_2 \ge \dfrac{25 - 25\cdot\dfrac{11}{25}}{\sqrt{25\cdot\dfrac{11}{25}\cdot\dfrac{14}{25}}} \approx \approx 5.64$. То есть фактически $\Phi(x_2) \approx 0.5$. Это дает нам возможность вычислить значения $\Phi(x_1)$ для четырех заданных значений P_n.

Первый случай $P_n = 0.7:\ \Phi(x_1) = 0.5 - 0.7 = -0.2,\ x_1 = -0.52 = \dfrac{625 - 11\cdot n}{\sqrt{154\cdot n}},\ n = 62.$

Второй случай $P_n = 0.8:\ \Phi(x_1) = 0.5 - 0.8 = -0.3,\ x_1 = -0.84 = \dfrac{625 - 11\cdot n}{\sqrt{154\cdot n}},\ n = 65.$

Третий случай $P_n = 0.9:\ \Phi(x_1) = 0.5 - 0.9 = -0.4,\ x_1 = -1.28 = \dfrac{625 - 11\cdot n}{\sqrt{154\cdot n}},\ n = 69.$

Четвёртый случай $P_n = 0.95:\ \Phi(x_1) = 0.5 - 0.95 = -0.45,\ x_1 = = -1.64 = \dfrac{625 - 11\cdot n}{\sqrt{154\cdot n}},\ n = 73.$

Дальнейшее построение графика выполнено с помощью языка программирования *Python* 3.8.1 скрипт и график находятся в приложении 1.2.

Задача 2. Рассматривается извлечение шаров без возвращения из второй корзины Выполняется серия из $n = G_2 + B_2$ экспериментов, подсчитывается число k извлечений красных шаров.

1. Построить график вероятности $P(k)$.

2. Построить график функции распределения $F(x)$.

3. Рассчитать математическое ожидание числа извлечённых красных шаров k.

4. Рассчитать дисперсию числа извлечённых красных шаров k.

Решение:

Для данного варианта $n = 15$, красных шаров $R_2 = 10$ во второй корзине, а всего шаров во второй корзине $N_2 = 25$. Для построения графиков запишем в общем виде формулу для $P(k)$.

$$P(k) = \frac{\begin{pmatrix} k \\ R_2 \end{pmatrix} \cdot \begin{pmatrix} n-k \\ G_2 + B_2 \end{pmatrix}}{\begin{pmatrix} n \\ R_2 + G_2 + B_2 \end{pmatrix}} = \frac{R_2! \cdot n! \cdot R_2! \cdot n!}{N_2! \cdot k! \cdot (n-k)! \cdot (R_2 - k)! \cdot k!} =$$

$$= \frac{10! \cdot 10! \cdot 15! \cdot 15!}{25! \cdot k! \cdot k! \cdot (10-k)! \cdot (15-k)!} =$$

$$= \frac{1}{3268760} \cdot \frac{15!}{k! \cdot (15-k)!} \cdot \frac{10!}{k! \cdot (10-k)!}$$

Последние 2 множителя в аналитическом выражении легко можно задать неким рекуррентным соотношением $F_k = F_{k-1} \cdot \dfrac{10-k}{k+1}$ и $F_k = F_{k-1} \cdot \dfrac{15-k}{k+1}$ для $k \geq 1$, для $k = 0$ будем считать, что $F_k = 1$.

Матожидание и дисперсию случайной величины будем считать по формулам для гипергеометрического распределения:

$$M(k) = \frac{n \cdot R_2}{N_2}, \quad D(k) = \frac{n \cdot \dfrac{R_2}{N_2} \cdot \left(1 - \dfrac{R_2}{N_2}\right) \cdot (N_2 - n)}{N_2 - 1}.$$

Дальнейшие вычисления были осуществлены с помощью языка программирования *Python* 3.8.1. Скрипт с графиками функций и вычислениями находится в приложении 2.

Задача 3. Рассматривается извлечение шаров без возвращения из третьей корзины. Выполняется серия из k экспериментов, которая прекращается, когда извлечены все R_3 красных шаров.

1. Рассчитать значения $P(k)$.

2. Рассчитать математическое ожидание числа извлечений k.

3. Рассчитать дисперсию числа извлечений k

Решение:

Заметим, что k не может быть меньше $R_3 = 8$. Также очевидно, k не может быть больше $N_3 = R_3 + G_3 + B_3 = 19$. Поэтому для целого $k \in (-\infty, 7] \ \cup \ [20, +\infty) \ \ P(k) = 0$. Чтобы найти необходимые значения вероятностей, будем рассуждать так: последний извлеченный из корзины шар должен быть красным и при этом последним из красных. Поэтому, если учесть, что было проведено всего k экспериментов, то до извлечения последнего шара было проведено, соответственно, $k-1$ экспериментов. Отсюда найдём вероятность того, что последний извлеченный шар оказался красным и последним из красных пи k экспериментах: $\dfrac{1}{N_3 - (k-1)}$. Тогда среди извлеченных до этого шаров должно быть $R_3 - 1$ красных шаров. Понятно, что вероятность достать 7 красных шаров из 19 без возвращения будет равна $\dfrac{8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2}{19 \cdot 18 \cdot 17 \cdot 16 \cdot 15 \cdot 14 \cdot 13} = \dfrac{8! \cdot 12!}{19!} = (N_3 - R_3 + 1) \cdot \dbinom{R_3}{N_3}^{-1}$. При этом было извлечено ещё некоторое количество шаров синего и зелёного цветов, вероятность этого события будет равна $\dfrac{11 \cdot 10 \cdot 9...}{12 \cdot 11 \cdot 10...} = \dfrac{N_3 - (k-1)}{N_3 - (R_3 - 1)}$. Важно ещё учесть, что зелёные и синие шары (а мы их в общем-то не различем) могли быть извлечены в различных комбинациях с красными шарами. Всего расположить $k - 1 - R_3 + 1$ шаров на $k-1$ позициях существует $\dbinom{k - R_3}{k-1}$ способов. Составим аналитическое выражене для $P(k)$ и упростим его:

$$P(k) = \binom{k-R_3}{k-1} \cdot \frac{N_3 - k + 1}{N_3 - R_3 + 1} \cdot \binom{R_3}{N_3}^{-1} \cdot \frac{N_3 - R_3 + 1}{N_3 - k + 1} =$$

$$= \frac{(k-1)! \cdot R_3! \cdot (N_3 - R_3)!}{(k-R_3)! \cdot (R_3 - 1)! \cdot N_3!} = \frac{8! \cdot 11! \cdot (k-1)!}{7! \cdot 19! \cdot (k - R_3)!} =$$

$$= \frac{1}{75582} \cdot \left(\frac{(k-1)!}{7! \cdot (k-8)!} \right)$$

Рассмотрим альтернативное решение данной задачи. Будем решать следующую задачу: пусть все наши N_3 шаров расположены в виде некоторой цепочки из красных и некрасных шаров (зелёных и синих, но мы их не различаем). k-ая позиция в цепочке означает k-ое извлечение. Чтобы посчитать количество всех возможных таких цепочек, найдем сочетание $\binom{R_3}{N_3}$. По условию нам требуется, чтобы на k-ом извлечении мы достали последний красный шар. Поэтому разобьём нашу цепочку на две части. Первая часть будет состоять из $k-1$ шара, а вторая часть будет состоять из $n-(k-1) = n-k+1$ шара. Среди $k-1$ шара у нас должно быть $R_3 - 1$ красных шара. Первая часть цепочки является аналогом событий, происходящих до k-го извлечения. Количество возможных первых частей цепочки, подходящих под условие, равно $\binom{R_3 - 1}{k-1}$. Вторая часть цепочки содержит один красный шар. Но мы знаем, что он располагается только в одной позиции, то есть k-ой. Поэтому общая вероятность будет равна

$$P(k) = \frac{\binom{R_3 - 1}{k-1}}{\binom{R_3}{N_3}} = \frac{(k-1)! \cdot R_3! \cdot (N_3 - R_3)!}{(k-R_3)! \cdot (R_3-1)! \cdot N_3!} =$$

$$\frac{8! \cdot 11! \cdot (k-1)!}{7! \cdot 19! \cdot (k-R_3)!} = \frac{1}{75582} \cdot \left(\frac{(k-1)!}{7! \cdot (k-8)!} \right).$$

Второй множитель в получившемся выражении можно упростить, если рассматривать его значения для различных $k =$

8, 9, 10 ...

Для $k = 8$ имеем $\dfrac{7!}{0! \cdot 7!} = \dfrac{7}{7} = 1.$

Для $k = 9$ имеем $\dfrac{8!}{1! \cdot 7!} = \dfrac{8}{1} = 8.$

Для $k = 10$ имеем $\dfrac{9!}{7! \cdot 2!} = \dfrac{9 \cdot 8}{2 \cdot 1} = 36.$

Заметим закономерность: данный множитель можно задать неким рекуррентным соотношением $F_k = F_{k-1} \cdot \dfrac{k-1}{k-8}$ для $k \geq 9$, для $k = 8$ будем считать, что $F_k = 1$.

Матожидание и дисперсию случайной величины будем считать по формулам:

$$M(k) = \sum k \cdot P(k), \quad D(k) = M(k^2) - M^2(k).$$

Дальнейшие вычисления были осуществлены с помощью языка программирования $Python$ 3.8.1. Скрипт и результаты находятся в приложении 3.

Приложение 1.1.

Задание 1. Зададим массивы с биномиальными коэффициентами для трех случаев, чтобы не вычислять одни и те же значения несколько раз. Массивы заполняются, исходя из рекуррентных соотношений.

```
NCK_1(1,1) = 1;
NCK_1(1,7) = 1;
NCK_2(1,1) = 1;
NCK_2(1,10) = 1;
NCK_3(1,1) = 1;
NCK_3(1,13) = 1;

for i = 1:1:3
   NCK_1(1,i+1) = NCK_1(1,i) * (7 - i) / i;
   NCK_1(1,7-i) = NCK_1(1,i) * (7 - i) / i;
end

for i = 1:1:5
   NCK_2(1,i+1) = NCK_2(1,i) * (10 - i) / i;
   NCK_2(1,10-i) = NCK_2(1,i) * (10 - i) / i;
end

for i = 1:1:12
   NCK_3(1,i+1) = NCK_3(1,i) * (13 - i) / i;
   NCK_3(1,13-i) = NCK_3(1,i) * (13 - i) / i;
end
```

Задаем векторы случайных величин для трех случаев соответственно.

```
x1 = 0:1:6;
x2 = 0:1:9;
x3 = 0:1:12;
```

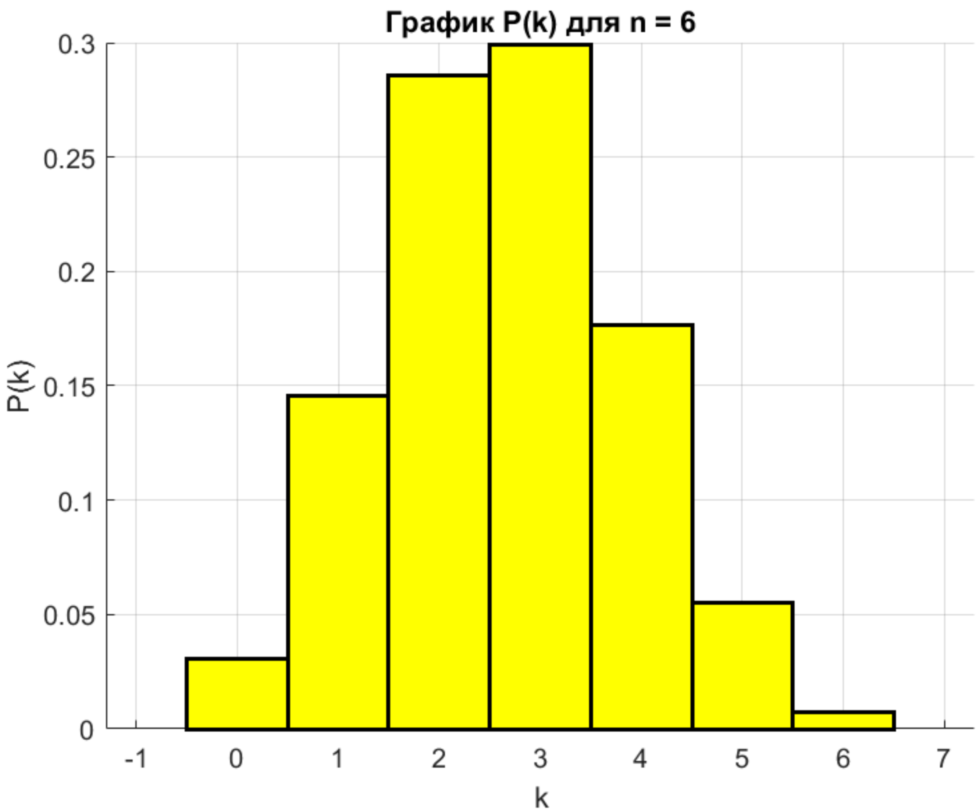
Заполняем массив вероятностей для первого случая n = 6.

```
P_k1 = (11/25).^x1 .* (14/25).^(6-x1) .* NCK_1(1:7);
fprintf( 'P(k) для n = 6: %.4f  %.4f  %.4f  %.4f  %.4f  %.4f  %.4f  \r\n', P_k1)
```

```
P(k) для n = 6: 0.0308  0.1454  0.2856  0.2992  0.1763  0.0554  0.0073
```

Выводим график для n = 6.

```
figure(1)
hold on
grid
bar(x1, P_k1, 1, "EdgeColor"," black", "FaceColor", "yellow", 'LineWidth', 1.5)
title('График P(k) для n = 6')
xlabel('k')
ylabel('P(k)')
hold off
```

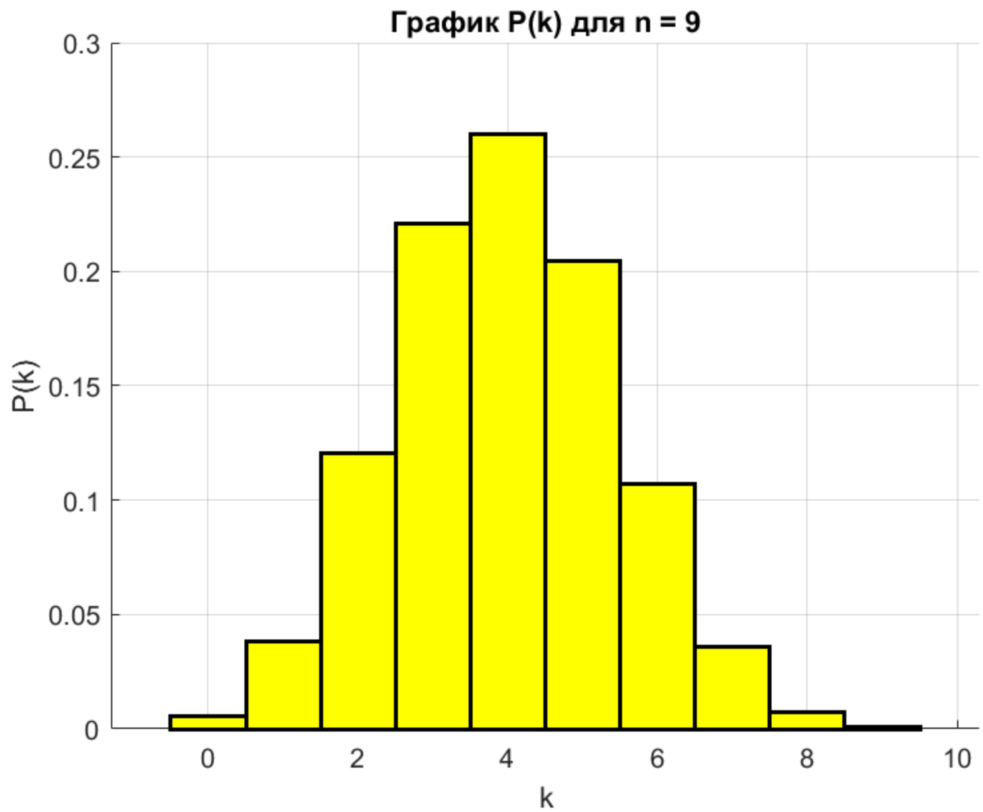


Аналогично поступаем с n = 9 и n = 12.

```
P_k2 = (11/25).^x2 .* (14/25).^(9-x2) .* NCK_2(1:10);
fprintf( ['P(k) для n = 9: %.4f  %.4f  %.4f  %.4f  %.4f  \n' ...
          '                %.4f  %.4f  %.4f  %.4f  %.4f  \r\n'], P_k2)
```

```
P(k) для n = 9: 0.0054  0.0383  0.1204  0.2207  0.2601
                0.2044  0.1070  0.0360  0.0071  0.0006
```

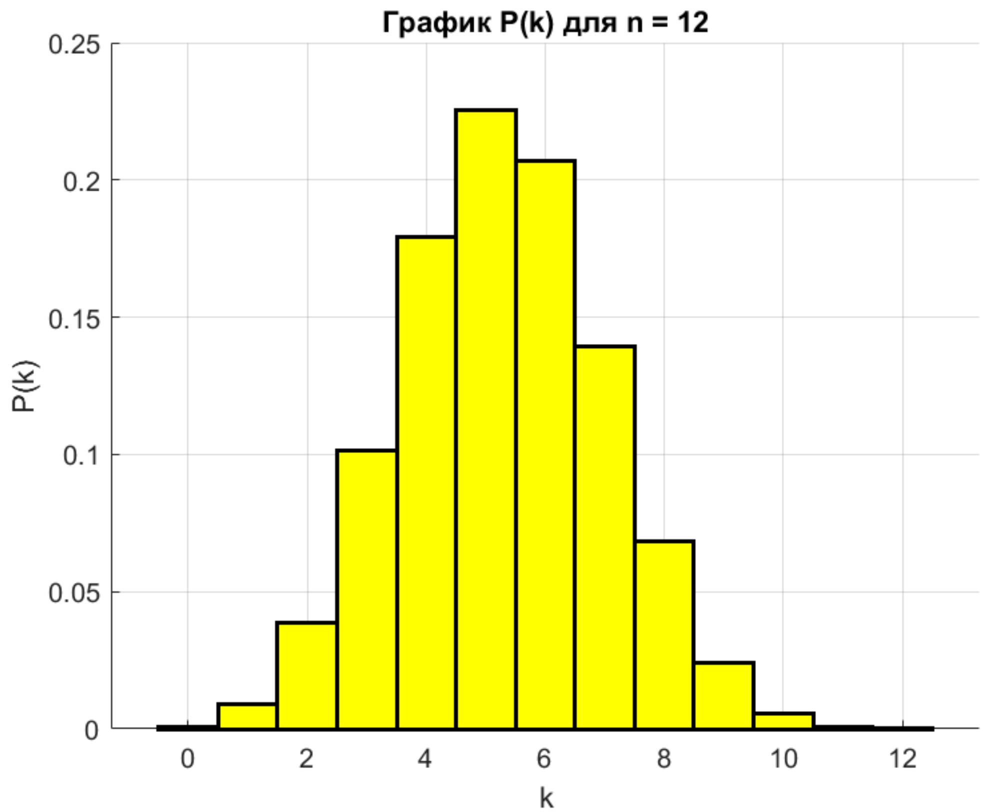
```
figure(2)
hold on
grid
bar(x2,P_k2,1,"EdgeColor","black","FaceColor","yellow", 'LineWidth', 1.5)
title('График P(k) для n = 9')
xlabel('k')
ylabel('P(k)')
hold off
```




```
P_k3 = (11/25).^x3 .* (14/25).^(12-x3) .* NCK_3(1:13);
fprintf( ['P(k) для n = 12: %.4f  %.4f  %.4f  %.4f  \n' ...
          '                 %.4f  %.4f  %.4f  %.4f  \n' ...
          '                 %.4f  %.4f  %.4f  %.4f  %.4f \r\n'], P_k3)
```

```
P(k) для n = 12: 0.0010  0.0090  0.0388  0.1015
                 0.1794  0.2256  0.2068  0.1393
                 0.0684  0.0239  0.0056  0.0008  0.0001
```

```
figure(3)
hold on
grid
bar(x3,P_k3,1,"EdgeColor","black","FaceColor","yellow", 'LineWidth', 1.5)
title('График P(k) для n = 12')
xlabel('k')
ylabel('P(k)')
hold off
```



Задание 2. Заполним массив функции распределения. Для этого используем функцию частичного суммирования cumsum.

```
for x = 1:7
    F_x = cumsum(P_k1);
end
```

Выводим значения функции распределения и график функции распределения для n = 6.

```
fprintf( 'F(x) для n = 6: %.4f  %.4f  %.4f  %.4f  %.4f  %.4f  %.4f  \r\n', F_x)
```

```
F(x) для n = 6: 0.0308  0.1762  0.4618  0.7610  0.9373  0.9927  1.0000
```

```

figure(4)
hold on
grid
stairs(x1, F_x,"LineWidth",1.5,"Color",[0.4660 0.6740 0.1880])
title('График F(x) для n = 6')
xlabel('x')
ylabel('F(x)')
hold off
```

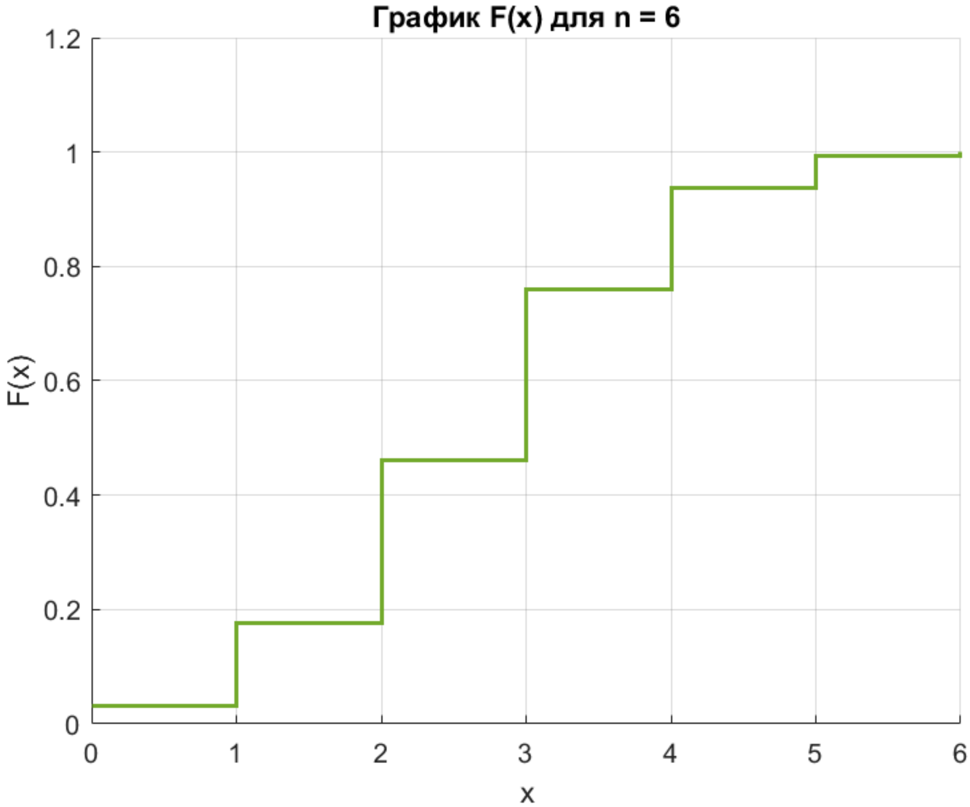


График F(x) для n = 6
F(x)
x
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6

Приложение 1.2.

Задание 3. Для решения данной задачи воспользуемся модулями matplotlib, math и numpy для построения графиков и математических расчетов.

In [1]:

```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt
from math import exp
from math import pi
```

Зададим функцию local, исходя из локальной теоремы Муавра-Лапласа.

In [2]:

```
p = 11 / 25
q = 14 / 25

def local(k, n):
    c = sqrt(n * p * q)
    x = (k - n * p) / c
    phi = exp(-x ** 2 / 2)
    return phi / c / sqrt(2 * pi)
```

Зададим фунцию нормального распределения для построения огибающих.

In [3]:

```
def gauss(x, n):
    med = n * 11 / 25
    sig = sqrt(n * 11 * 14 / 625)
    return 1 / sqrt(2 * pi) / sig * np.exp(-(x - med) ** 2 / 2 / sig ** 2 )
```

Зададим значения k, по которым будем строить графики для n = 25, 50, 100, 200, 400, 1000. Так как по условию требуется минимум 7 точек, то возьмём разные значения k с различным шагом для каждого n.

In [4]:

```
n = [25, 50, 100, 200, 400, 1000]
step = [2, 3, 4, 5, 6, 10]
k = [ [i for i in range(0, n[j], step[j])] for j in range(6) ]
```

Теперь необходимо задать списки со значениями функций local для каждого n.

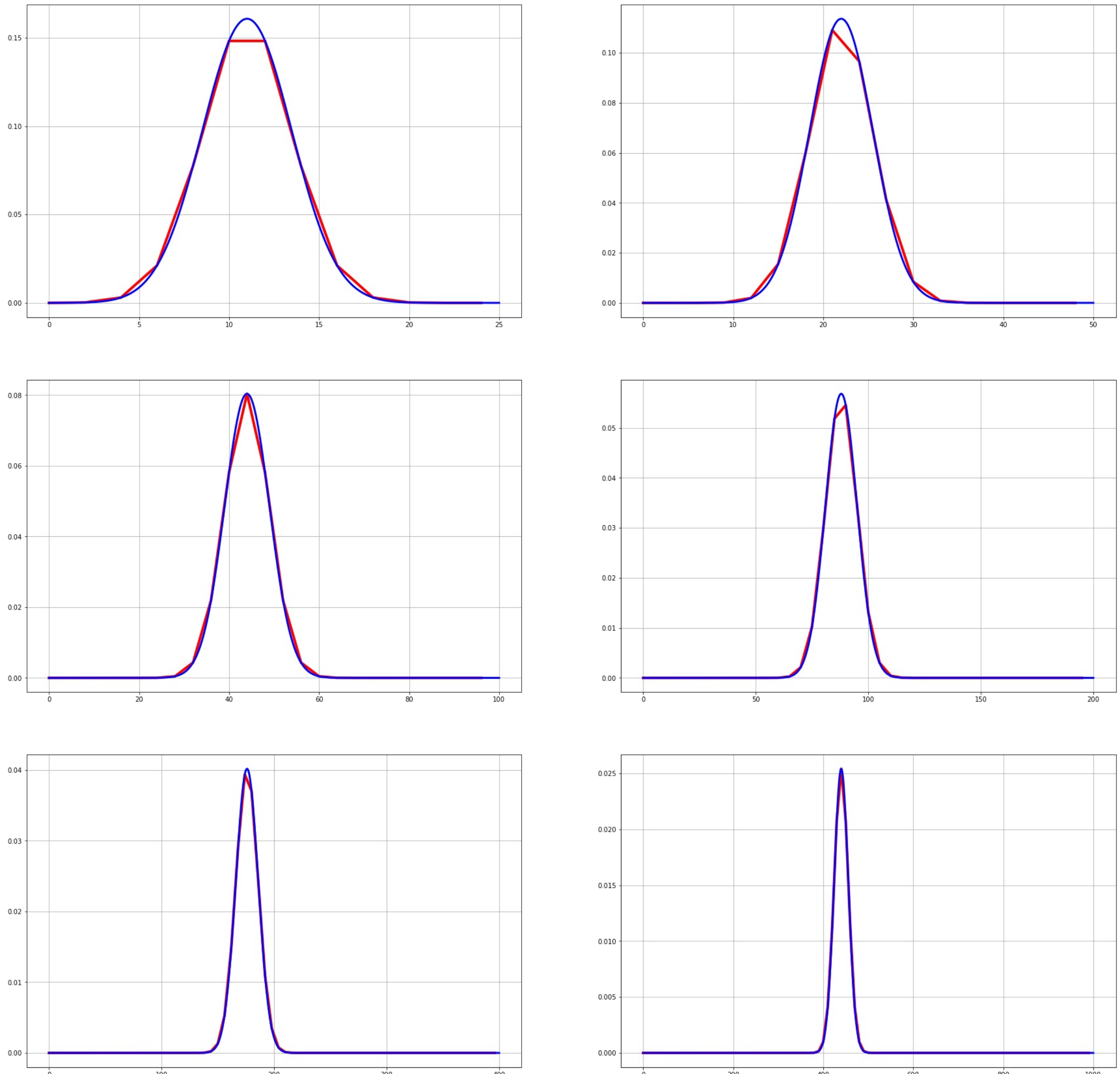
In [5]:

```
P_k = [ [local(x, n[i]) for x in k[i]] for i in range(6) ]
```

Построим необходимые графики кривых.

In [6]:

```
fig, ax = plt.subplots(3, 2, figsize=(30, 30))

font = {'family' : 'normal',
'weight' : 'bold',
'size' : 30}

matplotlib.rc('font', **font)

def graph(j):
    x = np.linspace(0, n[j], 1000)
    ax[j // 2, j % 2].plot(k[j], P_k[j], 'r', lw = 4)
    ax[j // 2, j % 2].plot(x, gauss(x, n[j]), 'b', lw = 3)
    ax[j // 2, j % 2].grid(True)

for j in range(6):
    graph(j)
```



Задание 4. Рассчитаем значения аргументов функции Лапласа для трёх указанных значений n.

In [7]:

```
R_1 = 11
p = 11 / 25
q = 14 / 25

print('x1 = ', R_1 / sqrt(25 * p * q), sep = '')
print('x2 = ', R_1 / sqrt(50 * p * q), sep = '')
print('x3 = ', R_1 / sqrt(100 * p * q), sep = '')
```

```
x1 = 4.432026302139591
x2 = 3.1339158526400435
x3 = 2.2160131510697956
```

Зададим значения вероятностей по таблице для функции Лапласа.

In [8]:

```
n = [25, 50, 100]
P = [2 * 0.499968, 2 * 0.49865, 2 * 0.4861]
P
```

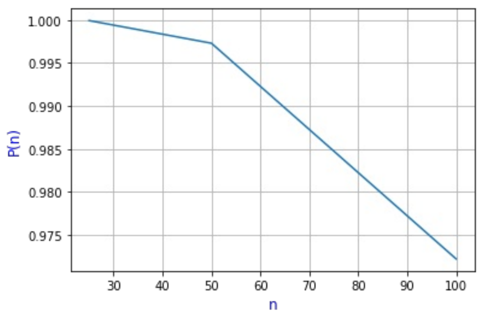
Out[8]:

```
[0.999936, 0.9973, 0.9722]
```

Построим необходимый график.

In [9]:

```
fig = plt.figure()
plt.plot(n, P)
plt.xlabel('n', fontsize=12, color='blue')
plt.ylabel('P(n)', fontsize=12, color='blue')
plt.grid(True)
```



Задание 5. Воспользуемся значениями из предыдущего задания, немного изменив их.

In [10]:

```
R_1_N = 11 / 25

print('x1 = ', R_1_N * 100 / sqrt(100 * p * q), sep = '')
print('x2 = ', R_1_N * 200 / sqrt(200 * p * q), sep = '')
print('x3 = ', R_1_N * 400 / sqrt(400 * p * q), sep = '')
```

```
x1 = 8.864052604279182
x2 = 12.535663410560174
x3 = 17.728105208558365
```

Зададим значения вероятностей по таблице для функции Лапласа.

In [11]:

```
n = [100, 200, 400]
P = [2 * 0.499999, 2 * 0.499999, 2 * 0.499999]
P
```

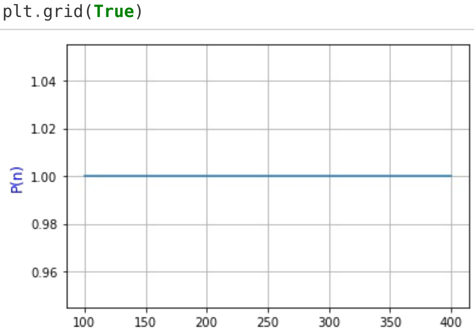
Out[11]:

```
[0.999998, 0.999998, 0.999998]
```

Построим необходимый график

In [12]:

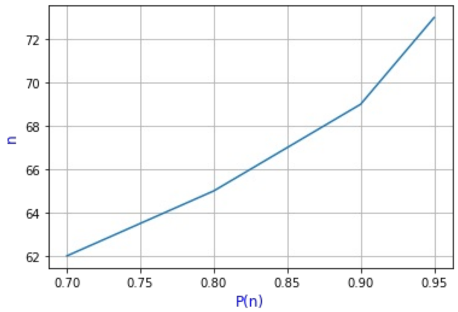
```
fig = plt.figure()
plt.plot(n, P)
plt.xlabel('n', fontsize=12, color='blue')
plt.ylabel('P(n)', fontsize=12, color='blue')
plt.grid(True)
```



Задание 7. Построим график зависимости n от P_n по найденным четырем точкам

In [13]:

```
n = [62, 65, 69, 73]
P_n = [0.7, 0.8, 0.9, 0.95]
fig = plt.figure()
plt.plot(P_n, n)
plt.xlabel('P(n)', fontsize=12, color='blue')
plt.ylabel('n', fontsize=12, color='blue')
plt.grid(True)
```



Приложение 2

Для решения данной задачи воспользуемся модулем matplotlib для построения графиков и математических расчетов.

In [1]:

```
import matplotlib.pyplot as plt
```

Создаём список со значениями случайной величины. Так как максимально может быть 10 красных шаров, то k будет принимать значения от 0 до 10.

In [2]:

```
x = [ i for i in range(11) ]
```

Запишем значение константы, вычисленной при аналитическом решении задачи:

In [3]:

```
c = 1 / 3268760
```

Создадим два списка со значениями рекуррентных функций, выведенных при аналитическом решении, и список со значениями вероятностей величины k. Списки со значениями рекуррентных функций создаются, чтобы не вычислять одни и те же значения несколько раз.

In [4]:

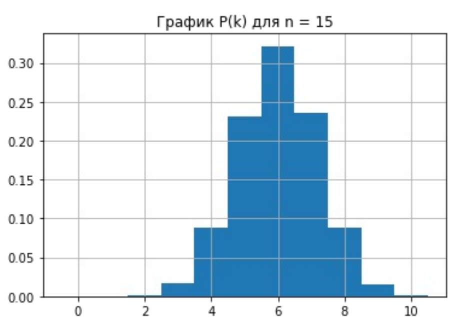
```
f_1 = [ 1 for i in range(11) ]
f_2 = [ 1 for i in range(11) ]

for i in range(10):
    f_1[i + 1] = f_1[9 - i] = f_1[i] * (10 - i) / (i + 1)
    f_2[i + 1] = f_2[i] * (15 - i) / (i + 1)

prob = [c * f_1[i] * f_2[i] for i in range(11)]

for i in range(11):
    print('P(', i, ') = ',  "%.8f" % prob[i], sep = '')
```

```
P(0) = 0.00000031
P(1) = 0.00004589
P(2) = 0.00144550
P(3) = 0.01670358
P(4) = 0.08769380
P(5) = 0.23151164
P(6) = 0.32154395
P(7) = 0.23623637
P(8) = 0.08858864
P(9) = 0.01531162
P(10) = 0.00091870
```

Теперь построим график для P(k).

In [5]:

```
fig = plt.figure()
plt.bar(x, prob, 1,)
plt.title('График P(k) для n = 15')
plt.grid(True)
```



Чтобы построить график функции распределения, запишем в отдельный список значения этой функции.

In [6]:

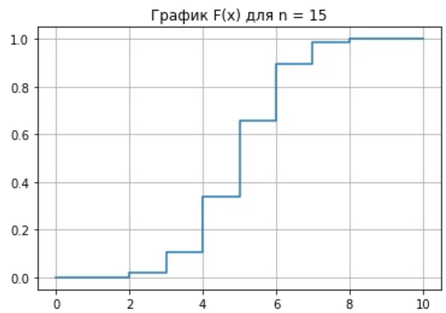
```
F_k = [ 0 for i in range(11) ]
F_k[0] = prob[0]

for i in range(10):
    F_k[i+1] = prob[i+1] + F_k[i]

for i in range(11):
    print('F(', i, ') = ',  "%.8f" % F_k[i], sep = '')
```

```
F(0) = 0.00000031
F(1) = 0.00004619
F(2) = 0.00149170
F(3) = 0.01819528
F(4) = 0.10588908
F(5) = 0.33740073
F(6) = 0.65894468
F(7) = 0.89518105
F(8) = 0.98376969
F(9) = 0.99908130
F(10) = 1.00000000
```

Теперь построим график F(x).

In [7]:

```
fig = plt.figure()
plt.step(x, F_k)
plt.title('График F(x) для n = 15')
plt.grid(True)
```



Выполним расчеты числовых характеристик случайной величины k: математического ожидания и дисперсии.

In [8]:

```
M = 15 * 10 / 25
D = 15 * 10 / 25 * 15 / 25 * 10 / 24

print('Матожидание :')
print("%.8f" % M)

print('Дисперсия :')
print("%.8f" % D)
```

```
Матожидание :
6.00000000
Дисперсия :
1.50000000
```

Приложение 3.

Задаем основные переменные и списки, которые пригодятся при дальнейших вычислениях. В переменную c записываем вычисленную в задаче 3 константу. В списке F_k хранятся значения рекуррентной фунции, данная реализация позволяет не вычислять одно и то же значения нексколько раз. В списке key хранятся случайные величины от 8 до 19. В списке prob хранятся значения вероятностей для каждой величины k.

```
sum_p = M = D = 0

c = 1 / 75582

F_k = [ 1 for i in range(12) ]

for i in range(11):
    F_k[i + 1] = F_k[i] * (i + 8) / (i + 1)

key = [ k + 8 for k in range(12) ]

prob = [ c * F_k[i] for i in range(12) ]
```

Выодим список со значениями нужных нам веротяностей и проверяем, равна ли сумма всех вероятностей единице. Эту сумму также выводим.

```
print('Список вероятностей :')

for i in range(12):
    print('P(', i+8, ') = ', "%.6f" % prob[i], sep = '')

for i in range(12):
    sum_p += prob[i]

print('Условие sum(P_i) = 1 :')
print(sum_p)
```

Рассчитываем основные числовые характеристики случайной величины k матожидания и дисперсию и выводим их.

```
for i in range(12):
    M += i * prob[i]
    D += i ** 2 * prob[i]

print('Матожидание :')
print("%.6f" % M)

print('Дисперсия :')
print("%.6f" % (D - M ** 2))
```

```
Список вероятностей :
P(8) = 0.000013
P(9) = 0.000106
P(10) = 0.000476
P(11) = 0.001588
P(12) = 0.004366
P(13) = 0.010479
P(14) = 0.022704
P(15) = 0.045408
P(16) = 0.085139
P(17) = 0.151359
P(18) = 0.257310
P(19) = 0.421053
Условие sum(P_i) = 1 :
1.0
Матожидание :
17.777778
Дисперсия :
2.172840
```