Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет «Робототехника и комплексная автоматизация»

Кафедра «Системы автоматизированного проектирования»

**Домашнее задание №2 Часть 1 по дисциплине**

**«Теория вероятностей и математическая статистика»**

**Вариант 14**

Выполнил:

студент группы РК6-36Б

Петраков С.А.

Москва

2020

Оглавление

[Задача 1. 3](#_Toc59089449)

[1. Построить графики вероятности P(k). Графики строятся для числа опытов n = 6, 9 и 12 c расчётом вероятностей по формуле Бернулли. 3](#_Toc59089450)

[2. Для n = 6 также строится график функции распределения F(х). 5](#_Toc59089451)

[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. 6](#_Toc59089452)

[4. Построить график вероятности того, что абсолютное число извлечений красных шаров отклонится от математического ожидания не более, чем на R1. При построении графика использовать n = 25, 50, 100. 9](#_Toc59089453)

[5. Построить график вероятности того, что относительное число извлечений красных шаров отклонится от математического ожидания не более, чем на R1 / (R1+G1+B1). При построении графика использовать n = 100, 200, 400. 10](#_Toc59089454)

[6. Рассчитать допустимый интервал числа успешных испытаний k (симметричный относительно математического ожидания), обеспечивающий попадание в него с вероятностью P = R1 / (R1+G1+B1) при n = 1000. 11](#_Toc59089455)

[7. Построить график зависимости минимально необходимого числа испытаний n, для того, чтобы обеспечить вероятность появления не менее, чем N1=R1+G1+B1 красных шаров с вероятностями P = 0,7; 0,8; 0,9; 0,95. 11](#_Toc59089456)

[Задача 2. 14](#_Toc59089457)

[1. Построить график вероятности P(k). 14](#_Toc59089458)

[2. Построить график функции распределения F(x). 15](#_Toc59089459)

[3. Рассчитать математическое ожидание числа извлечённых красных шаров k. 15](#_Toc59089460)

[4. Рассчитать дисперсию числа извлечённых красных шаров k. 16](#_Toc59089461)

[Задача 3. 17](#_Toc59089462)

[1. Рассчитать значения P(k). 17](#_Toc59089463)

[2. Рассчитать математическое ожидание числа извлечений k. 18](#_Toc59089464)

[3. Рассчитать дисперсию числа извлечений k. 18](#_Toc59089465)

# Задача 1.

Рассматривается извлечение шаров с возвращением из первой корзины (см. исходные данные к ДЗ №1: R1, G1, B1). Выполняется серия из n экспериментов, подсчитывается число k извлечений красных шаров.

**R1 = 11**

**G1 = 10**

**B1 = 11**

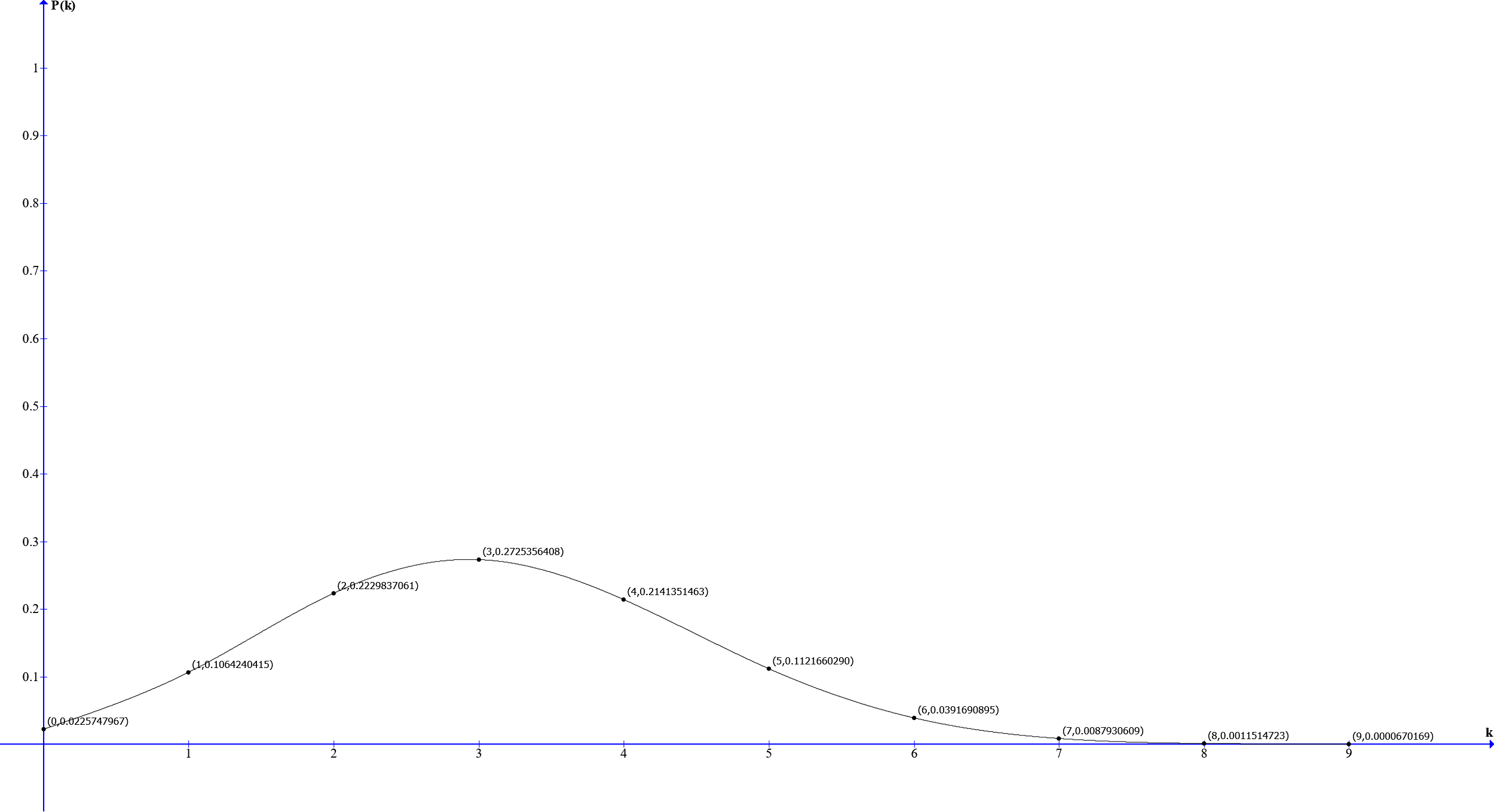
## Построить графики вероятности P(k). Графики строятся для числа опытов n = 6, 9 и 12 c расчётом вероятностей по формуле Бернулли.

Формула Бернулли:

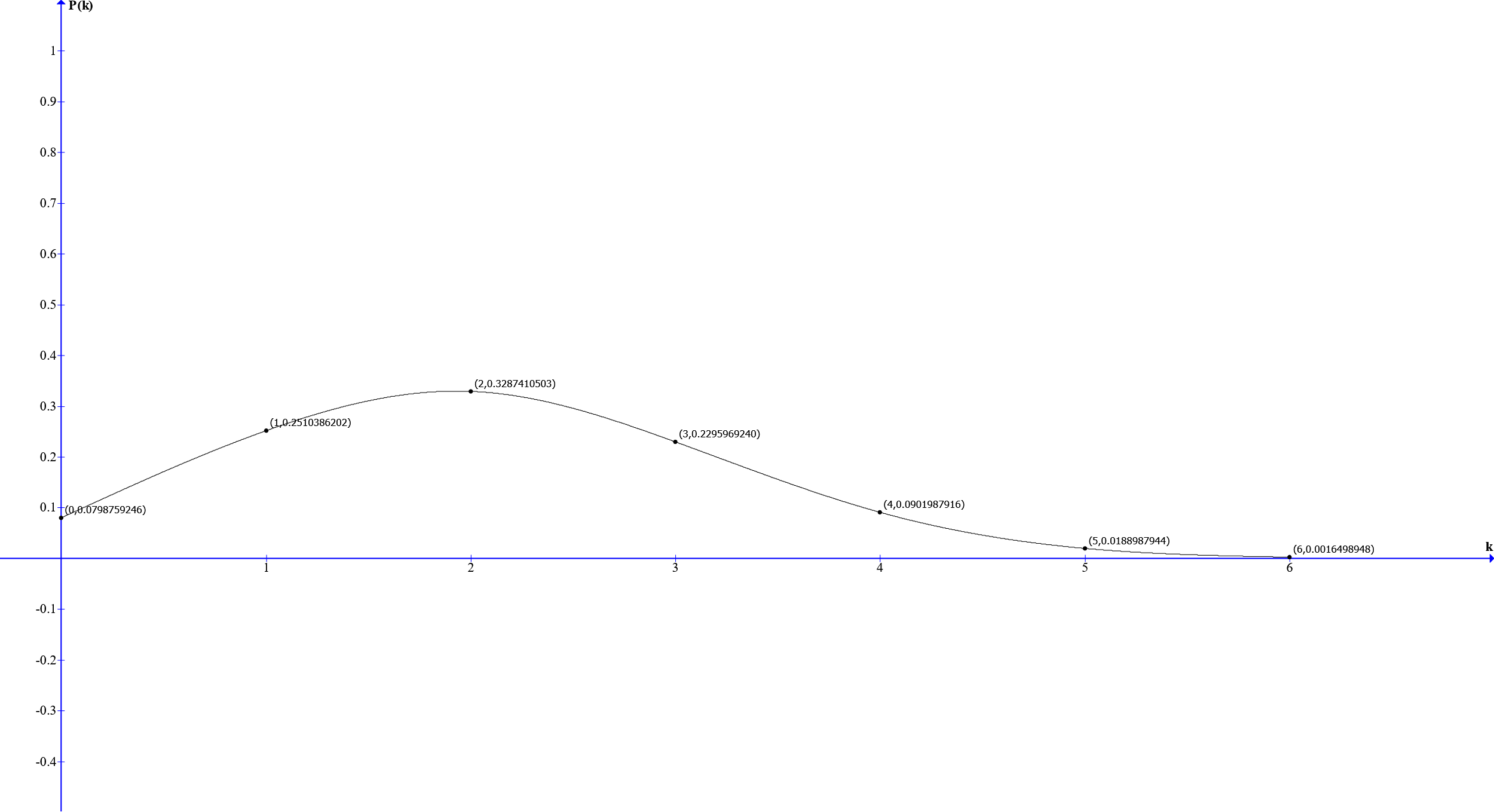
, где

– вероятность появления события в одном испытании.

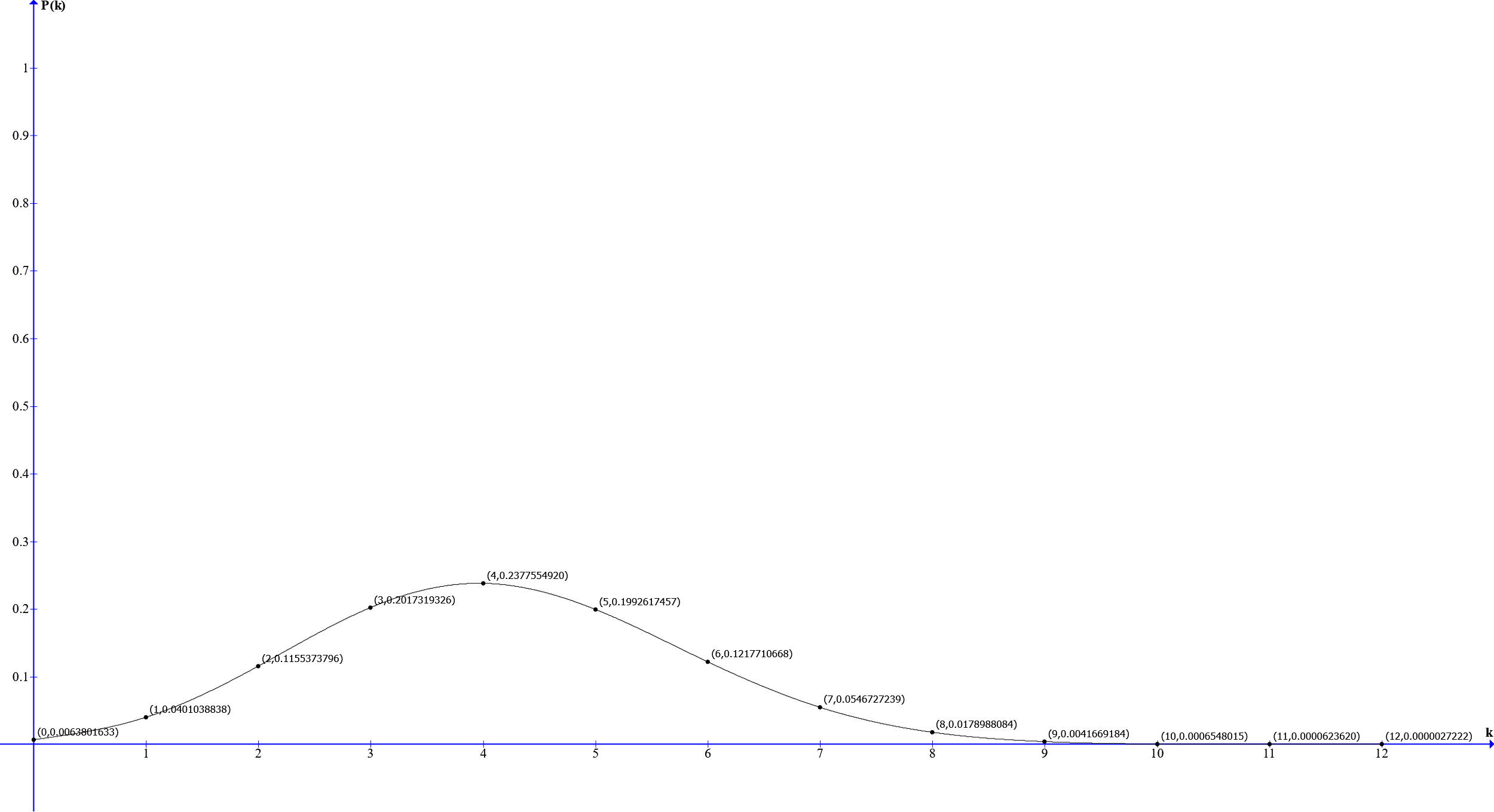
n=6



n=9



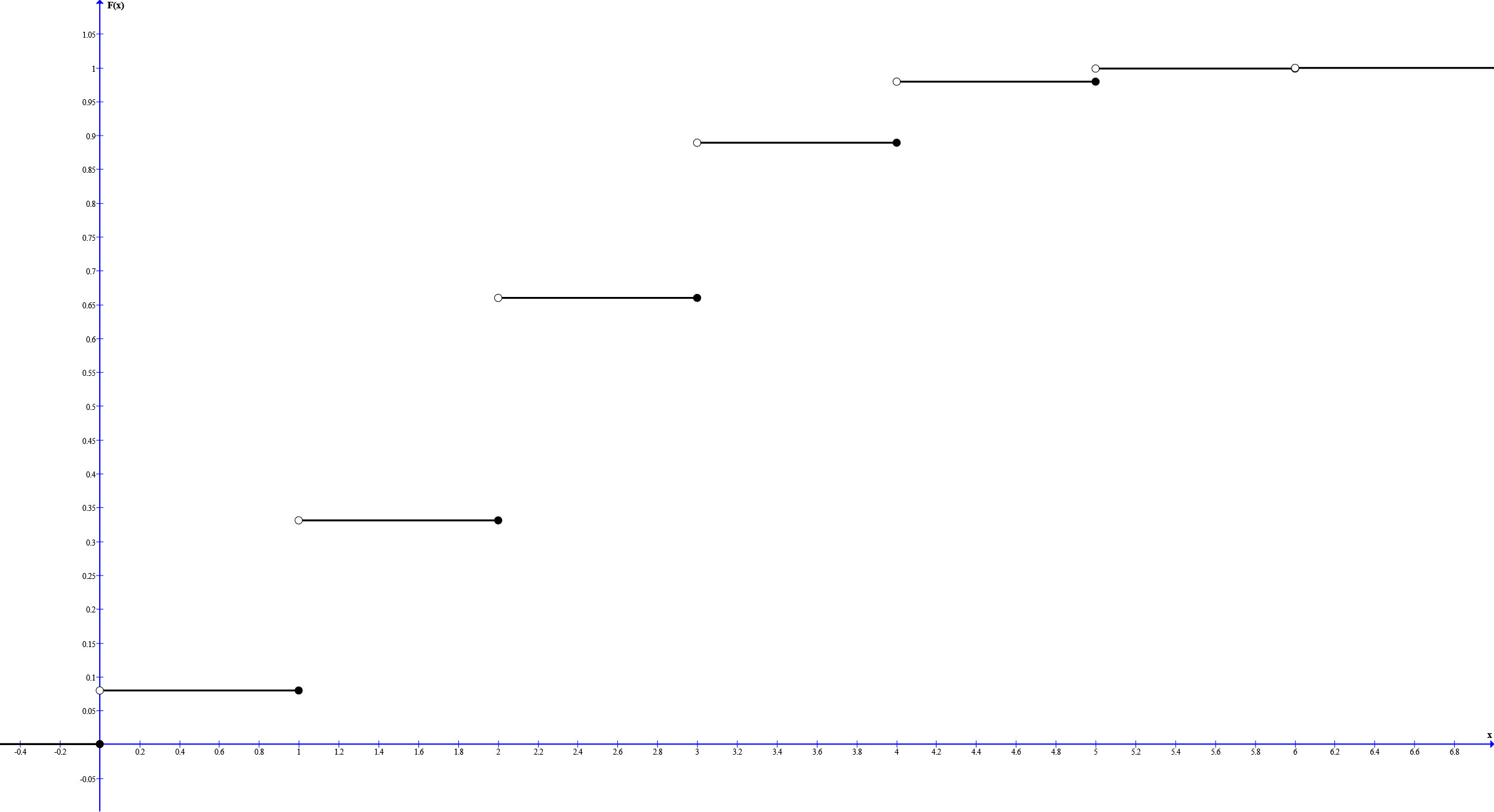
n=12



## Для n = 6 также строится график функции распределения F(х).



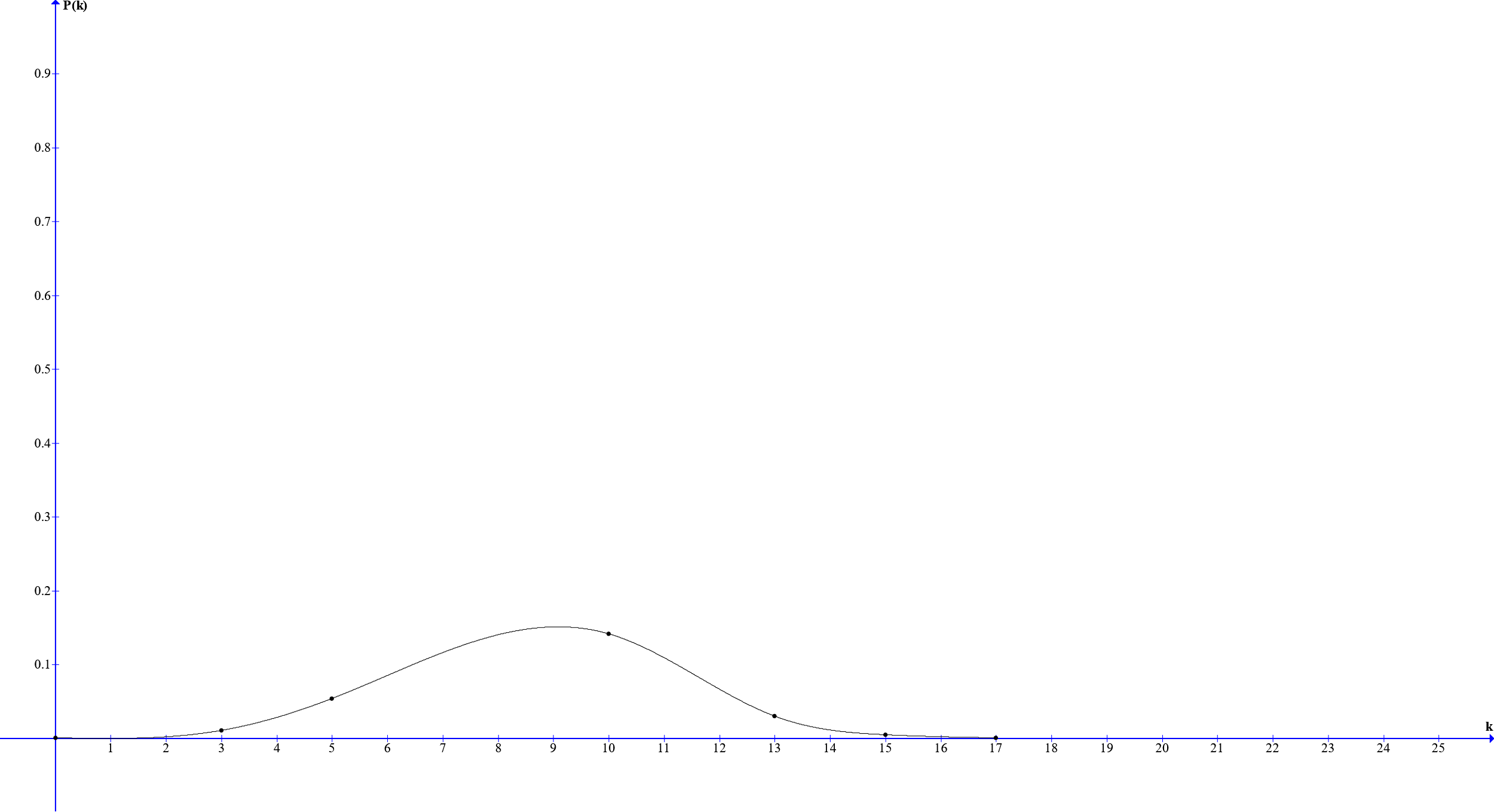
1. …



## 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.

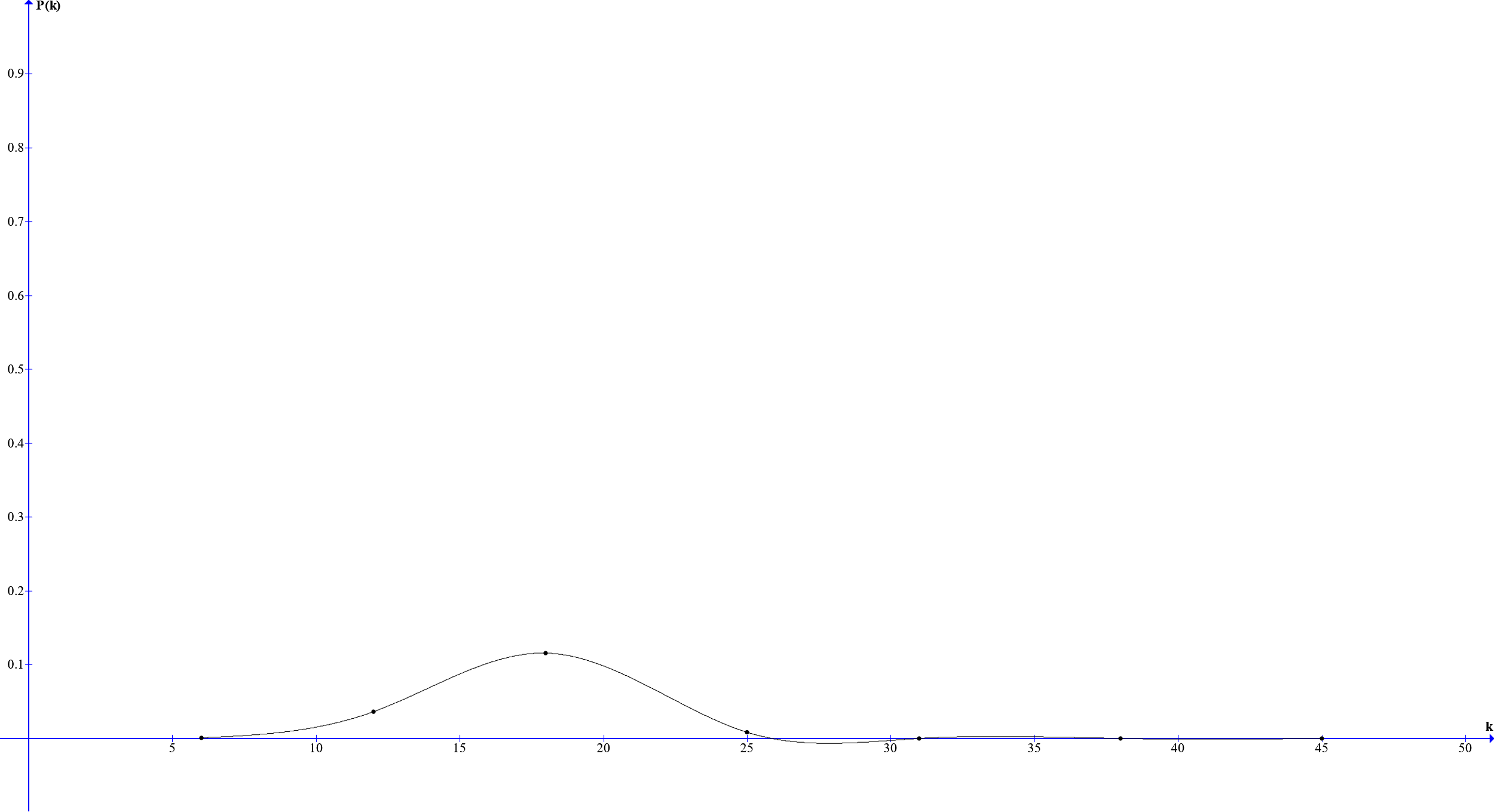
Для n = 25:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| k = | 0 | 3 | 5 | 10 | 13 | 15 | 17 |
| P(k) | 0.000253 | 0.010359 | 0.053731 | 0.141149 | 0.029771 | 0.004379 | 0.000337 |



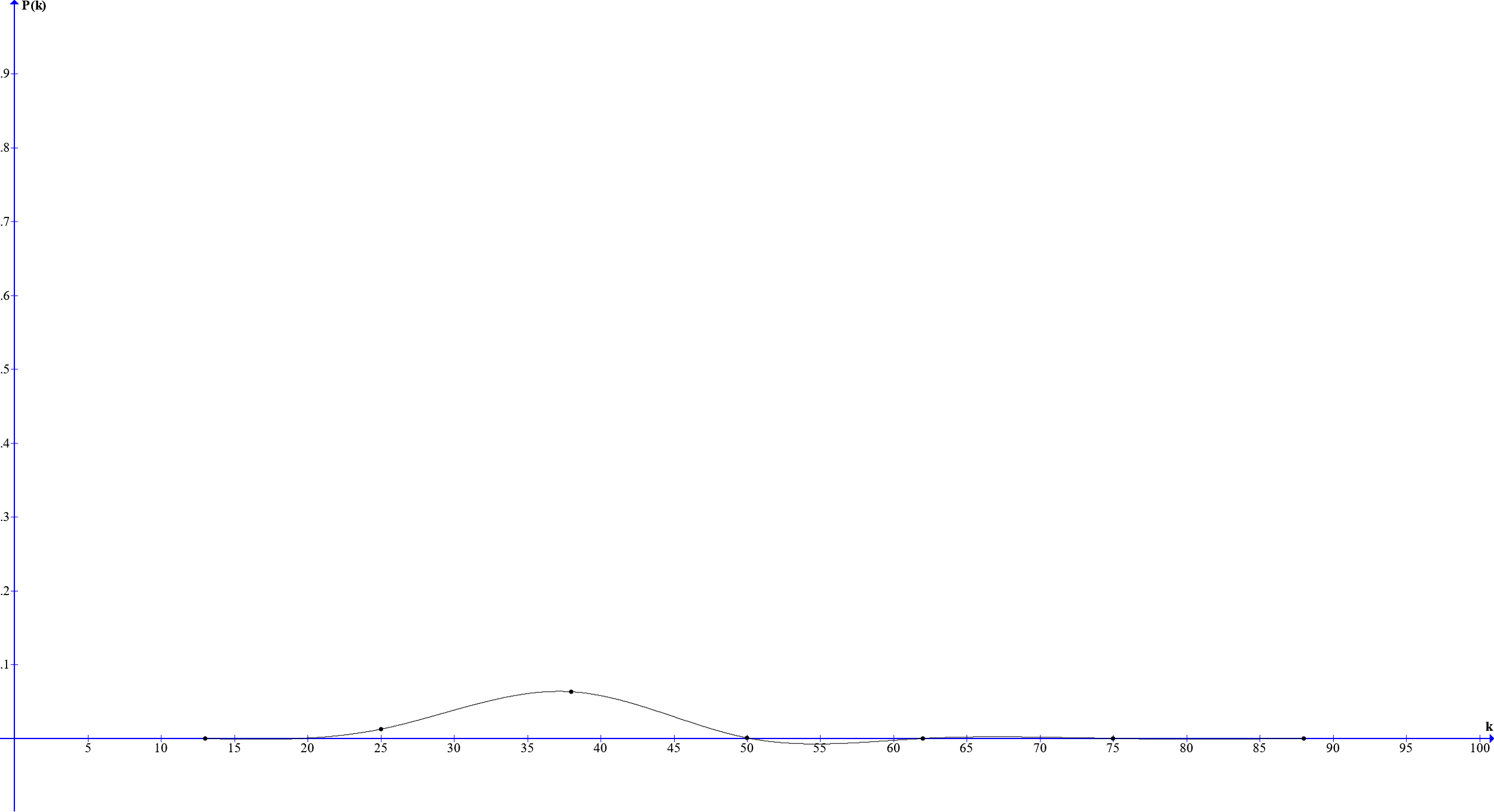
Для n = 50:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| k = | 6 | 12 | 18 | 25 | 31 | 38 | 45 |
| P(k) | 0.000476 | 0.036296 | 0.11541 | 0.007861 | 0.000030 | 0.000030 | 0.000030 |



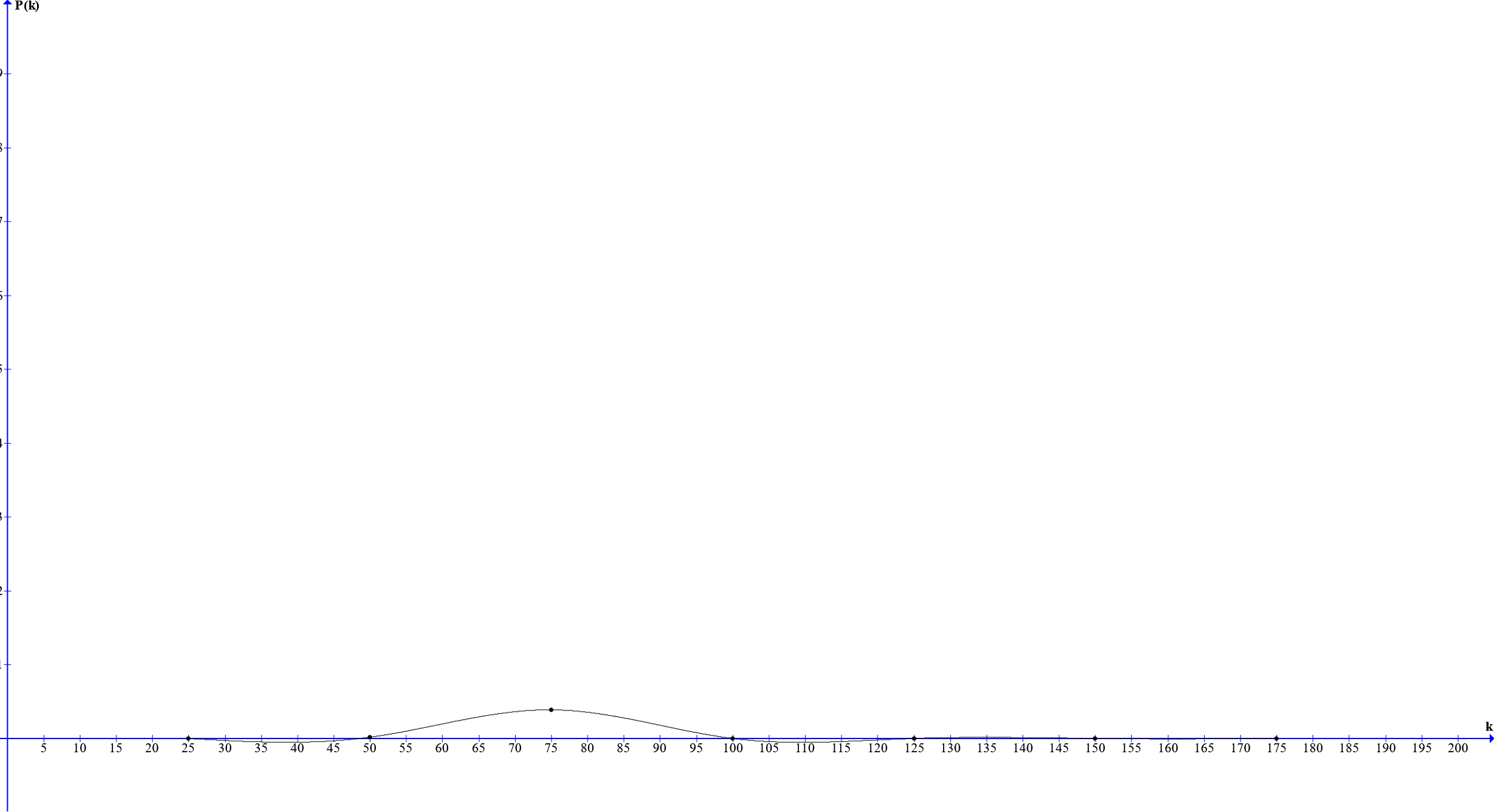
Для n = 100:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| k = | 13 | 25 | 38 | 50 | 62 | 75 | 88 |
| P(k) | 0.000021 | 0.012064 | 0.062932 | 0.000379 | 0.000021 | 0.000021 | 0.000021 |



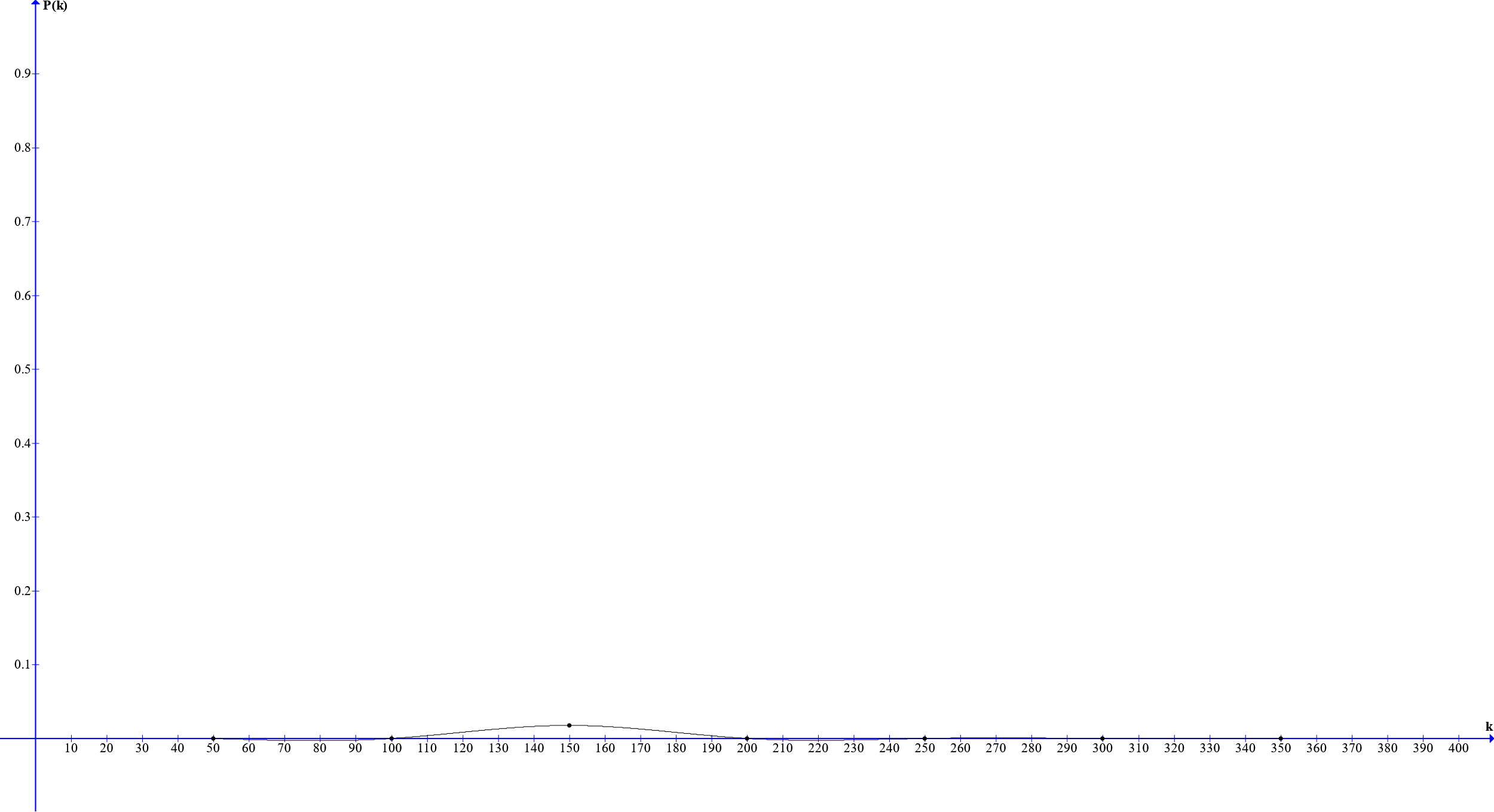
Для n = 200:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| k = | 25 | 50 | 75 | 100 | 125 | 150 | 175 |
| P(k) | 0.000015 | 0.001206 | 0.038544 | 0.000015 | 0.000015 | 0.000015 | 0.000015 |



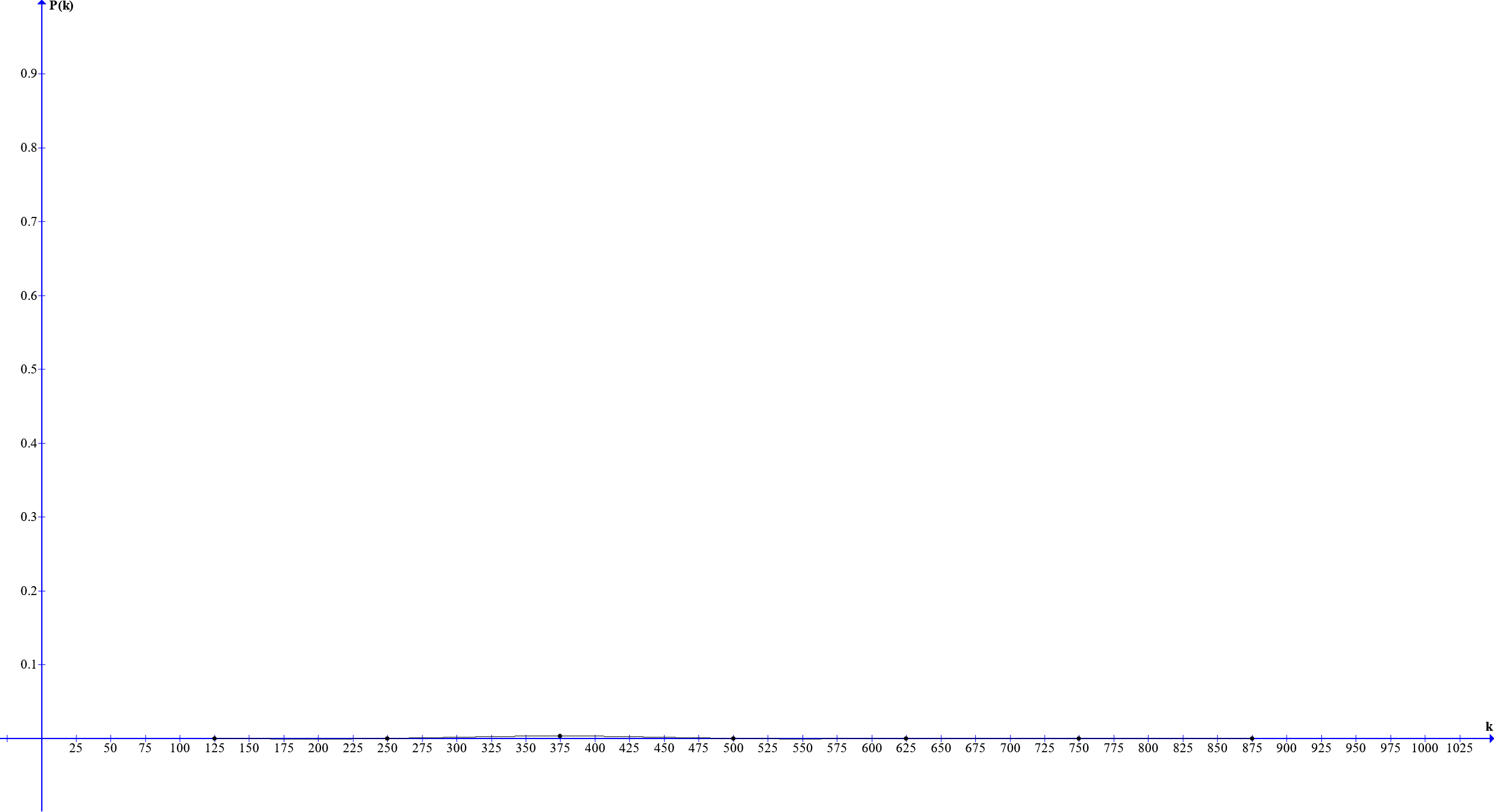
Для n = 400:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| k = | 50 | 100 | 150 | 200 | 250 | 300 | 350 |
| P(k) | 0.000011 | 0.000021 | 0.017570 | 0.000011 | 0.000011 | 0.000011 | 0.000011 |



Для n = 1000:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| k = | 125 | 250 | 375 | 500 | 625 | 750 | 875 |
| P(k) | 0.000007 | 0.000007 | 0.003056 | 0.000007 | 0.000007 | 0.000007 | 0.000007 |

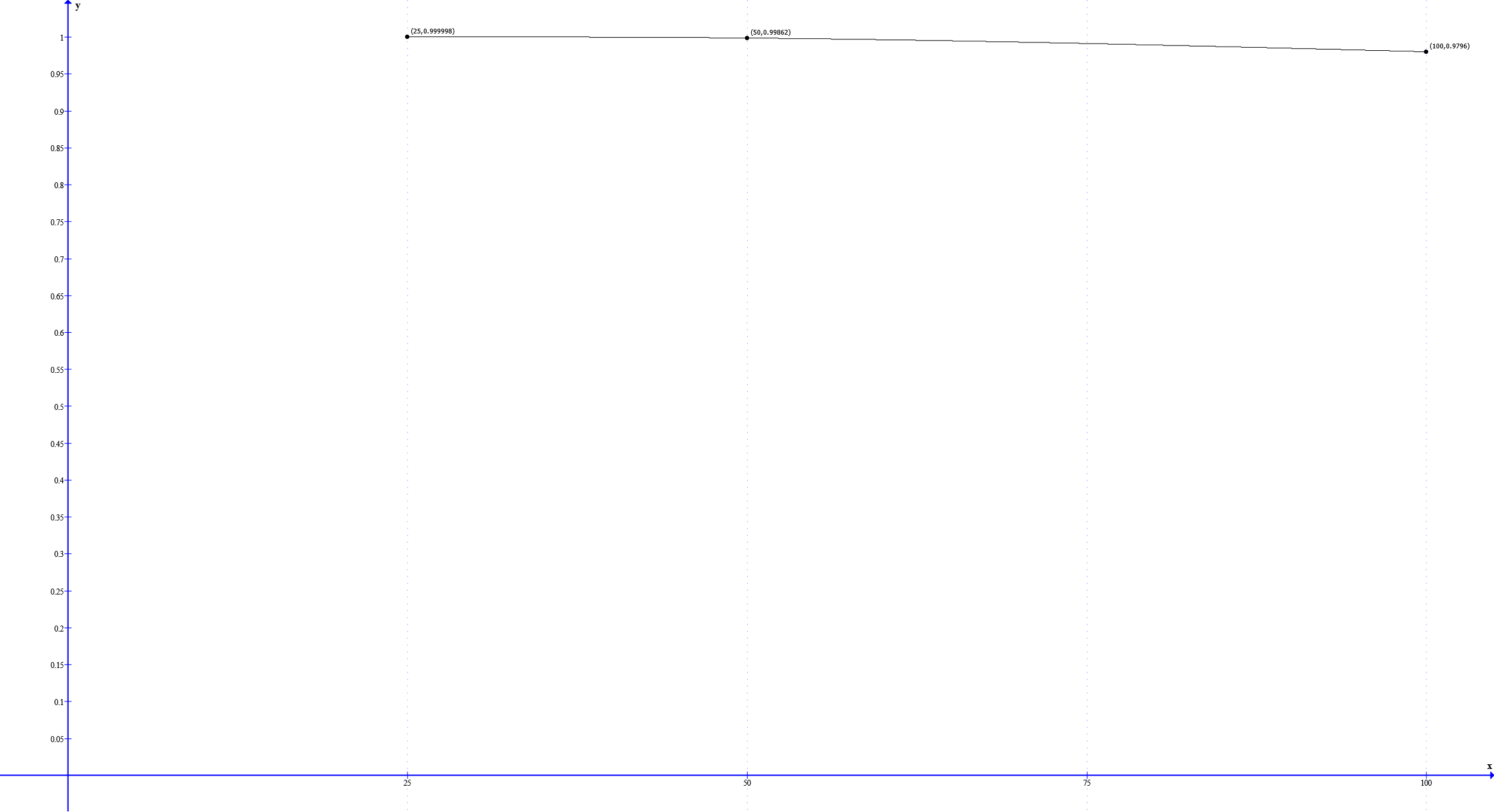


## Построить график вероятности того, что абсолютное число извлечений красных шаров отклонится от математического ожидания не более, чем на R1. При построении графика использовать n = 25, 50, 100.

n=25

n=50

n=100

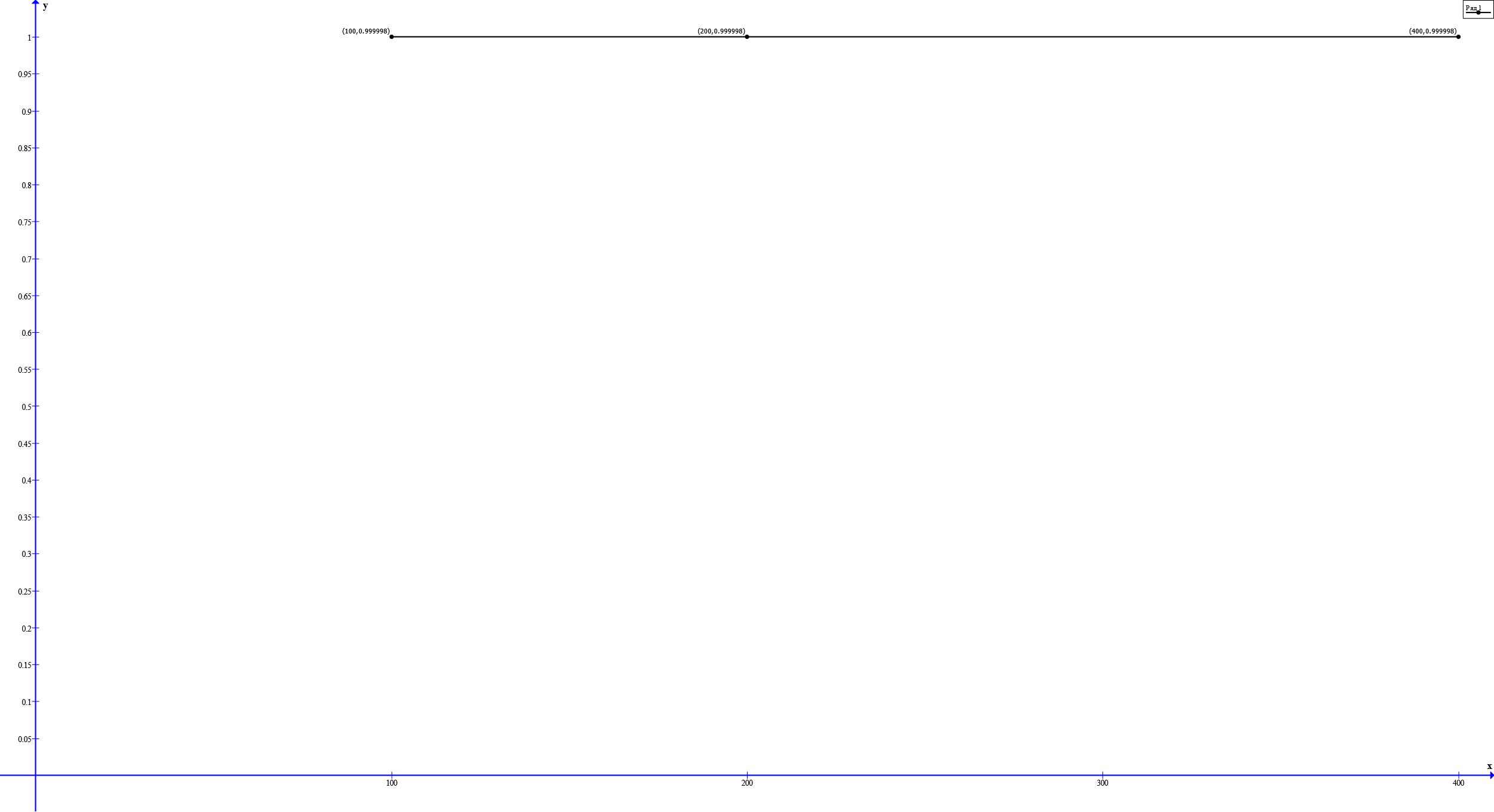


## Построить график вероятности того, что относительное число извлечений красных шаров отклонится от математического ожидания не более, чем на R1 / (R1+G1+B1). При построении графика использовать n = 100, 200, 400.

n=100

n=200

n=400



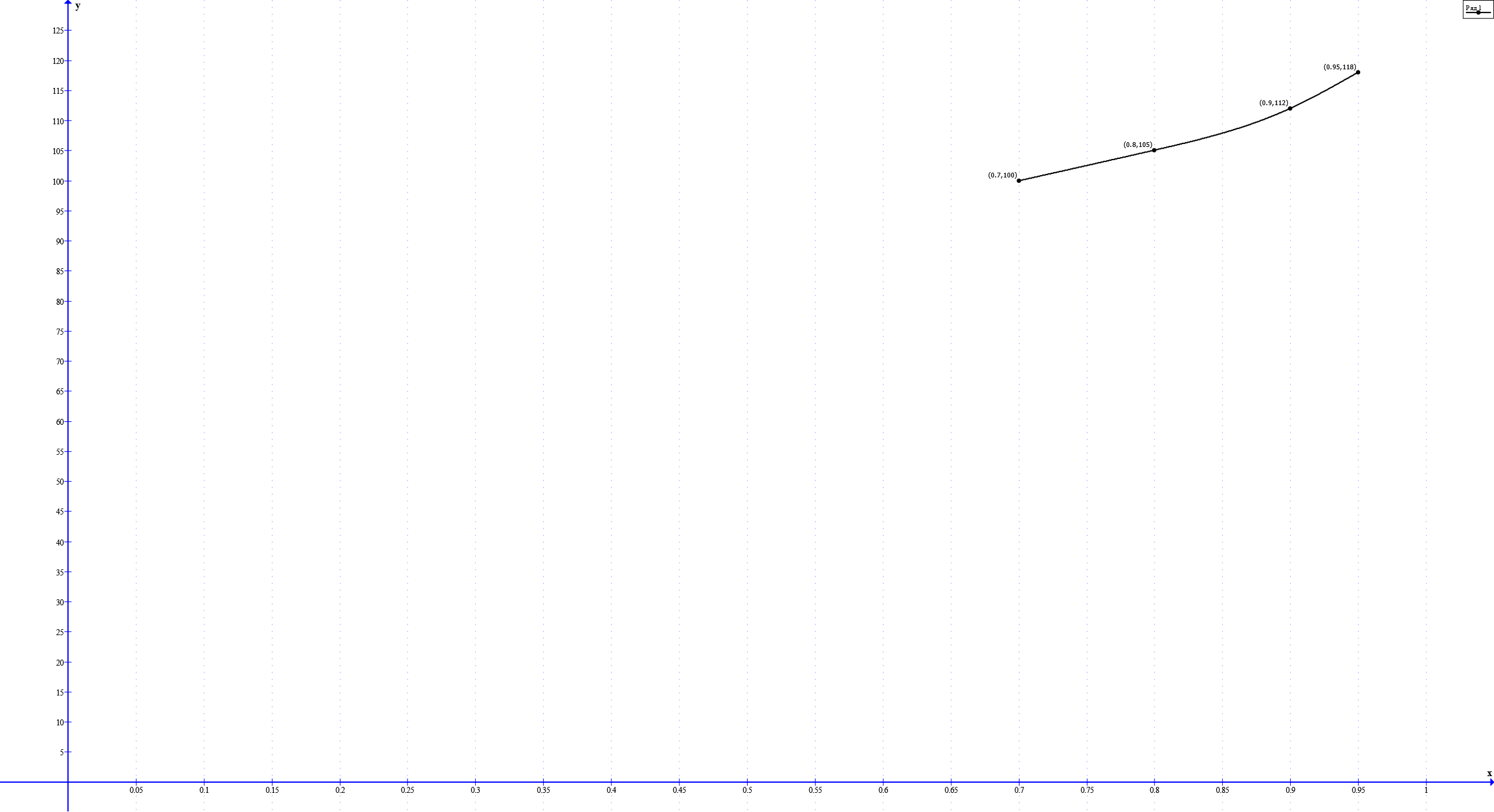
## Рассчитать допустимый интервал числа успешных испытаний k (симметричный относительно математического ожидания), обеспечивающий попадание в него с вероятностью P = R1 / (R1+G1+B1) при n = 1000.

A-Событие попадания в интервал

Тогда искомый интервал:

## Построить график зависимости минимально необходимого числа испытаний n, для того, чтобы обеспечить вероятность появления не менее, чем N1=R1+G1+B1 красных шаров с вероятностями P = 0,7; 0,8; 0,9; 0,95.

При , , тогда выражение для P примет вид:

******

# Задача 2.

Рассматривается извлечение шаров без возвращения из второй корзины (см. исходные данные к ДЗ №1: R2, G2, B2). Выполняется серия из n=G2+B2 экспериментов, подсчитывается число k извлечений красных шаров.

**R2 = 10**

**G2 = 9**

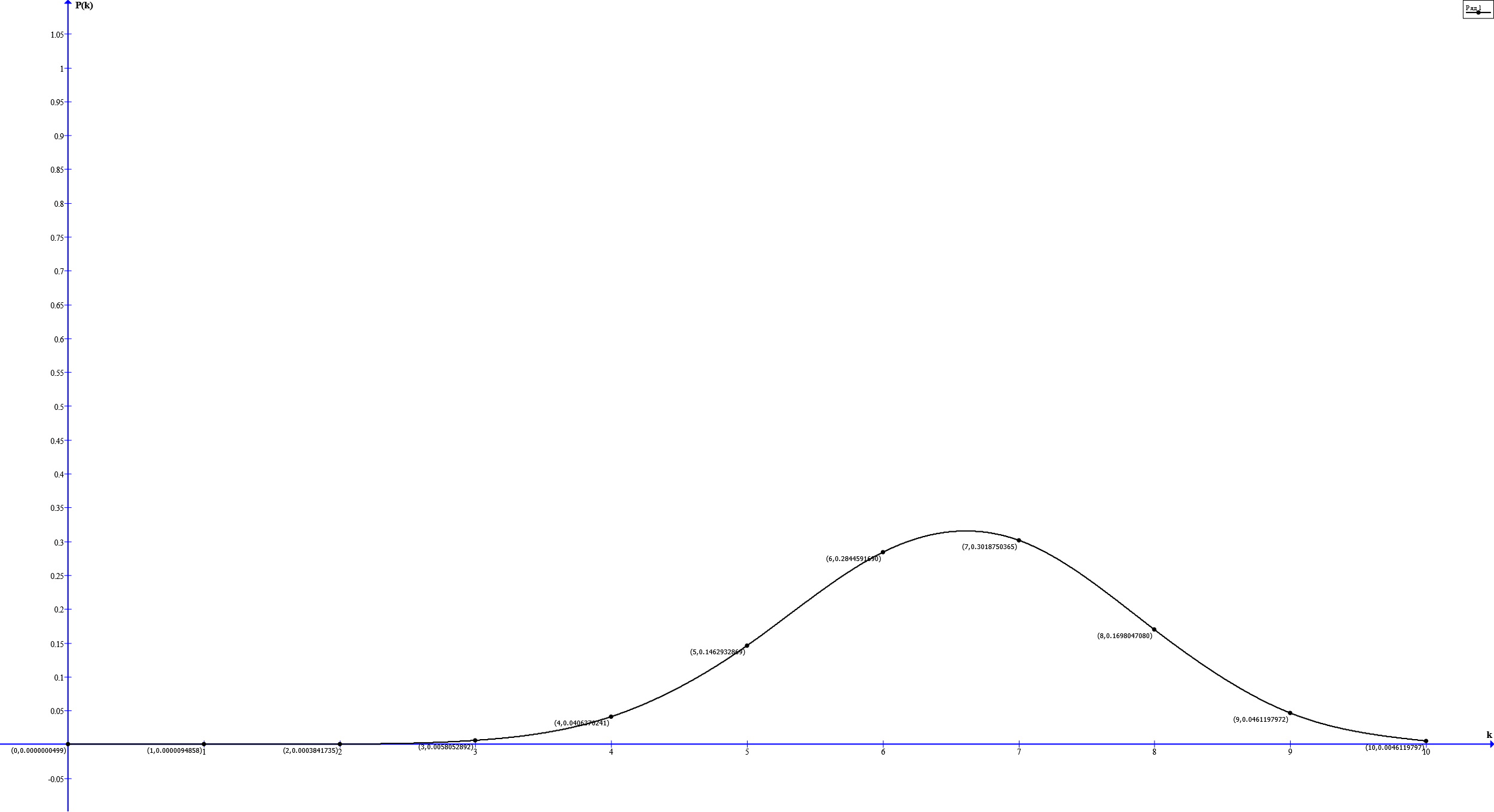
**B2 = 10**

**n=19**

**N=29**

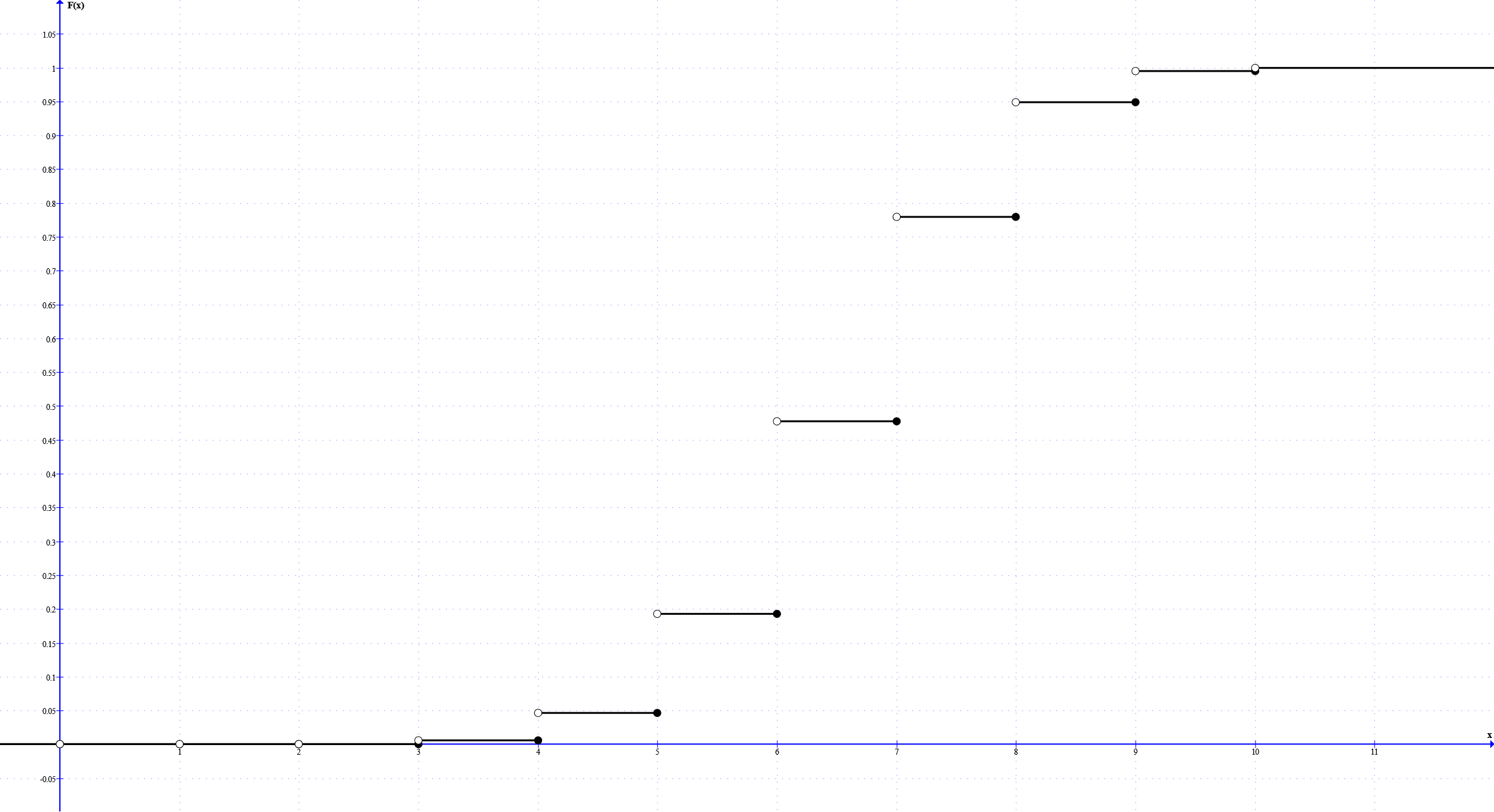
## Построить график вероятности P(k).

**



## Построить график функции распределения F(x).

**



## Рассчитать математическое ожидание числа извлечённых красных шаров k.

## Рассчитать дисперсию числа извлечённых красных шаров k.

# Задача 3.

Рассматривается извлечение шаров без возвращения из третьей корзины (см. исходные данные к ДЗ №1: R3, G3, B3). Выполняется серия из k экспериментов, которая прекращается, когда извлечены все R3 красных шаров.

**R3 = 9**

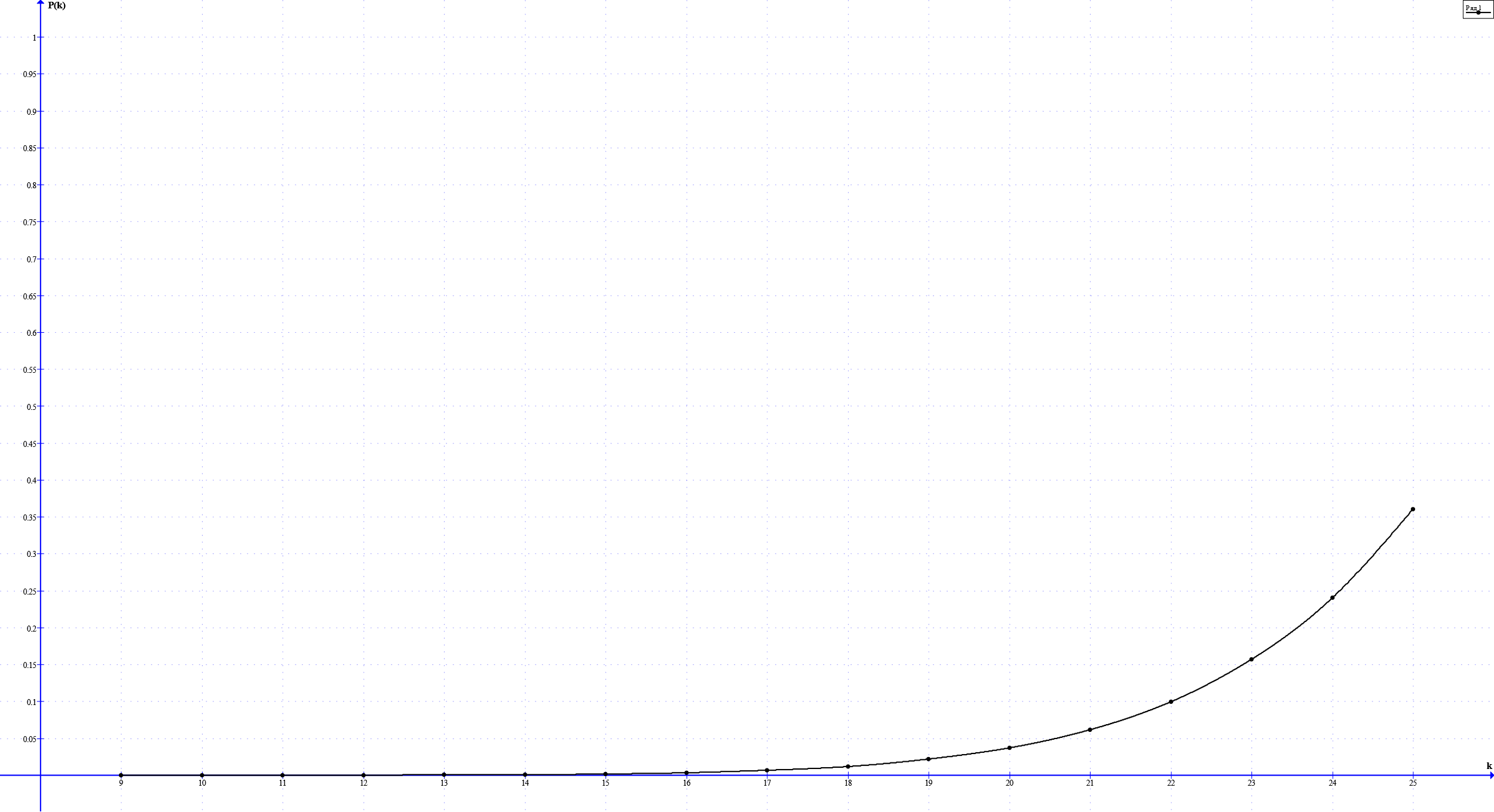
**G3 = 11**

**B3 = 5**

**N=R3+G3+B3=25**

## Рассчитать значения P(k).





## Рассчитать математическое ожидание числа извлечений k.

## Рассчитать дисперсию числа извлечений k.

# Программа

#include <iostream>

#include <iomanip>

using namespace std;

#pragma region Utility

long double fact(int N)

{

long double result = 1;

for (int i = 1; i <= N; i++) {

result = result \* i;

}

return result;

}

long double Combinations(int n, int k)

{

if (k == 0 || k == n) { return 1; }

if (k == 1 || k == n - 1) { return n; }

int numeratorStart = k > n / 2 ? k + 1 : (n - k) + 1;

int denomAmount = k > n / 2 ? n - k : k;

int amount = denomAmount > (n - numeratorStart) ? n - numeratorStart + 1 : denomAmount;

long double result = 1;

int a = 1, b = numeratorStart;

for (int i = 1; i <= amount; ++i)

{

result \*= b++;

result /= a++;

}

return result;

}

#pragma endregion

#pragma region Probabilities

long double Bernulli(long double p, int n, int k)

{

return Combinations(n, k) \* pow(p, k) \* pow(1 - p, n - k);

}

long double ProbabilityWithoutReturn(int n,int k,int N,int r)

{

return Combinations(r,k)\* Combinations(N-r, n-k)/ Combinations(N,n);

}

long double ProbabilityWithoutReturnToR(int r, int N, int k)

{

return Combinations(k-1, k-r) / Combinations(N, r);

}

#pragma endregion

#pragma region MathExpectings

long double MathExpecting(long double p, int n)

{

long double result = 0;

for (int i = 0; i <= n; i++)

{

result = result+ Bernulli(p, n, i) \* i;

}

return result;

}

long double MathExpectingSqr(long double p, int n)

{

long double result = 0;

for (int i = 0; i <= n; i++)

{

result = result + Bernulli(p, n, i) \* i \* i;

}

return result;

}

long double MathExpectingRelative(long double p, int n)

{

long double result = 0;

for (int i = 0; i <= n; i++)

{

result = result + Bernulli(p, n, i) \* i/n;

}

return result;

}

long double MathExpectingRelativeSqr(long double p, int n)

{

long double result = 0;

for (int i = 0; i <= n; i++)

{

result = result + Bernulli(p, n, i) \* i\*i/((long double)n\* (long double)n);

}

return result;

}

long double MathExpectingWithoutReturn(int r, int N, int n)

{

long double result = 0.0;

for (int i = 0; i < r; i++)

{

result += ProbabilityWithoutReturn(n,i,N,r)\*(long double)i;

}

return result;

}

long double MathExpectingWithoutReturnSqr(int r, int N, int n)

{

long double result = 0.0;

for (int i = 0; i < r; i++)

{

result += ProbabilityWithoutReturn(n, i, N, r) \* (long double)i\* (long double)i;

}

return result;

}

long double MathExpectingWithoutReturnToR(int r, int N)

{

long double result = 0;

for (int i = r; i <= N; i++)

{

result += ProbabilityWithoutReturnToR(r, N, i) \* (long double)i;

}

return result;

}

long double MathExpectingWithoutReturnToRSqr(int r, int N)

{

long double result = 0;

for (int i = r; i <= N; i++)

{

result += ProbabilityWithoutReturnToR(r, N, i) \* (long double)i\* (long double)i;

}

return result;

}

#pragma endregion

#pragma region Dispersions

long double Dispersion(long double p, int n) {

return MathExpectingSqr(p, n) - pow(MathExpecting(p, n), 2);

}

long double DispersionRelative(long double p, int n) {

return MathExpectingRelativeSqr(p, n) - pow(MathExpectingRelative(p, n), 2);

}

long double DispersionWithoutReturn(int r, int N, int n) {

return MathExpectingWithoutReturnSqr(r, N,n) - pow(MathExpectingWithoutReturn(r, N, n), 2);

}

long double DispersionWithoutReturntoR(int r, int N) {

return MathExpectingWithoutReturnToRSqr(r, N) - pow(MathExpectingWithoutReturnToR(r, N), 2);

}

#pragma endregion

#pragma region Deviation

long double StdDeviation(long double p, int n)

{

return sqrtl(Dispersion(p, n));

}long double StdDeviationRelative(long double p, int n)

{

return sqrtl(DispersionRelative(p, n));

}

#pragma endregion

#pragma region Punkt1

void Number1Point1(int r, int g, int b, int n)

{

long double p = (long double)r / ((long double)r + (long double)g + (long double)b);

for (int i = 0; i <= n; i++)

{

cout << "P(" << i << ") = " << setprecision(10)<< Bernulli(p, n, i)<<endl;

}

}

void Number1Point3(int r, int g, int b, int n)

{

int k;

long double p = (long double)r / ((long double)r + (long double)g + (long double)b);

long double q = 1 - p;

long double fi;

while (1)

{

cout << "Enter k = ";

cin >> k;

long double forFi = (k - n \* p) / sqrtl(n \* q \* p);

cout << "ForFi = " << forFi << endl;

cout << "Enter fi = ";

cin >> fi;

cout << "P(" << k << ") = " << fi / sqrtl(n \* q \* p)<<endl<<endl;

}

}

void Number1Point4(int r, int g, int b, int n)

{

long double p = (long double)r / ((long double)r + (long double)g + (long double)b);

long double q = 1 - p;

long double F;

long double forF;

//long double forF = MathExpecting(p,n)/StdDeviation(p,n)ж

forF =r / StdDeviation(p, 25);

cout << "ForF = " << forF << endl;

cout << "Enter F = ";

cin >> F;

cout << "for n = 25" << ": " << 2\*F << endl << endl;

forF = r / StdDeviation(p, 50);

cout << "ForF = " << forF << endl;

cout << "Enter F = ";

cin >> F;

cout << "for n = 50" << ": " << 2 \* F << endl << endl;

forF = r / StdDeviation(p, 100);

cout << "ForF = " << forF << endl;

cout << "Enter F = ";

cin >> F;

cout << "for n = 100" << ": " << 2 \* F << endl << endl;

}

void Number1Point5(int r, int g, int b, int n)

{

long double p = (long double)r / ((long double)r + (long double)g + (long double)b);

long double q = 1 - p;

long double F;

long double forF;

forF = p / StdDeviationRelative(p, 100);

cout << "ForF = " << forF << endl;

cout << "Enter F = ";

cin >> F;

cout << "for n = 100" << ": " << 2 \* F << endl << endl;

forF = p / StdDeviationRelative(p, 200);

cout << "ForF = " << forF << endl;

cout << "Enter F = ";

cin >> F;

cout << "for n = 200" << ": " << 2 \* F << endl << endl;

forF = p / StdDeviationRelative(p, 400);

cout << "ForF = " << forF << endl;

cout << "Enter F = ";

cin >> F;

cout << "for n = 400" << ": " << 2 \* F << endl << endl;

}

void Number1Point6(int r, int g, int b, int n)

{

long double p = (long double)r / ((long double)r + (long double)g + (long double)b);

cout <<"Deviation: " <<StdDeviation(p, n)<<endl;

cout << "MathExpecting: " << MathExpecting(p, n) << endl;

}

void Number1Point7(int r, int g, int b, int n)

{

long double N1 = (long double)r + (long double)g + (long double)b;

long double phi;

long double i=19;

for(int i=19;i<=n;i++)

{

phi = (1024-11\* (long double)i)/sqrtl(231\* (long double)i);

cout << i << ": " << phi << endl;

}

}

#pragma endregion

#pragma region Punkt2

void Number2Point1(int r, int g, int b, int n)

{

std::cout << "P(k)=...\n";

for (int i = 0; i <= r; i++)

{

std::cout << i << "\t" <<setprecision(10)<< ProbabilityWithoutReturn(n,i,r+g+b,r)<<endl;

}

}

void Number2Point3(int r, int g, int b, int n)

{

cout << MathExpectingWithoutReturn(r, r + g + b, n) << endl;

}

void Number2Point4(int r, int g, int b, int n)

{

cout << DispersionWithoutReturn(r, r + g + b, n) << endl;

}

#pragma endregion

#pragma region Punkt3

void Number3Point1(int r, int g, int b, int n)

{

std::cout << "P(k)=...\n";

for (int i = r; i <= (r+g+b); i++)

{

std::cout << i << "\t" << setprecision(10) << ProbabilityWithoutReturnToR(r, r + g + b, i) << endl;

}

}

void Number3Point2(int r, int g, int b, int n)

{

std::cout << "M(k) = " << setprecision(10) << MathExpectingWithoutReturnToR(r, r + g + b) << endl;;

}

void Number3Point3(int r, int g, int b, int n)

{

cout << DispersionWithoutReturntoR(r, r + g + b) << endl;

}

#pragma endregion

int main()

{

#pragma region input

cout.setf(ios::fixed);

cout << "Enter R,G,B,n:"<<endl;

int R=0, G=0, B=0, n=1000;

cout << "R= ";

cin >> R;

cout << "G= ";

cin >> G;

cout << "B= ";

cin >> B;

cout << "n= ";

//cin >> n; //For punkt1

//n = G + B;//FOr punkt2

//Nothing for punkt3

cout << n << endl;

cout << "End of enter" << endl << endl;

#pragma endregion

Number1Point6(R, G, B, n);

return 0;

}