Assignment 1

1) mean

1. 한 개의 주사위를 굴렸을 때 -> 

2. n 개의 주사위를 굴렸을 때 -> 

2) variance

1. 한 개의 주사위를 굴렸을 때



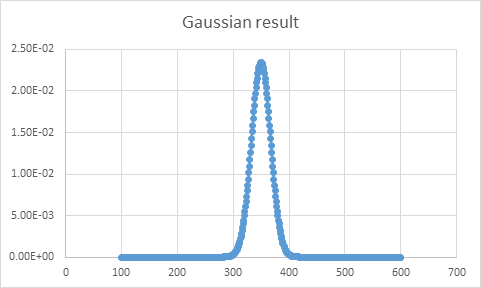
2. n 개의 주사위를 굴렸을 때



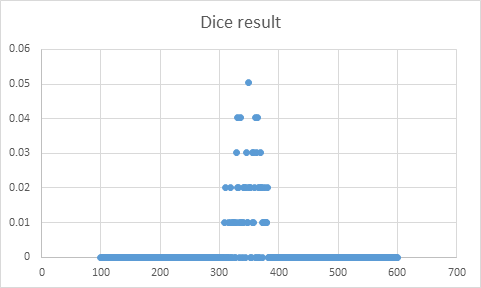
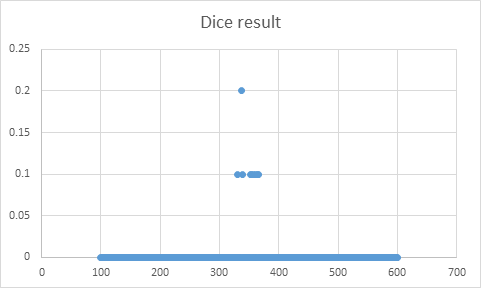
3) 100개의 주사위를 N번 던졌을 때 주사위 눈금 합의 분포와 Gaussian 분포 비교

\* , , 

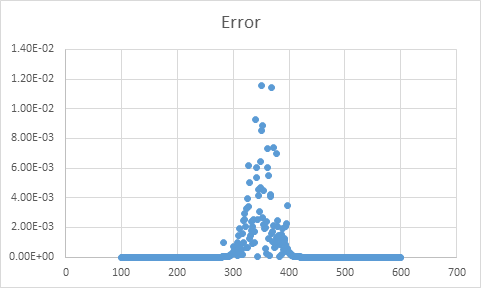
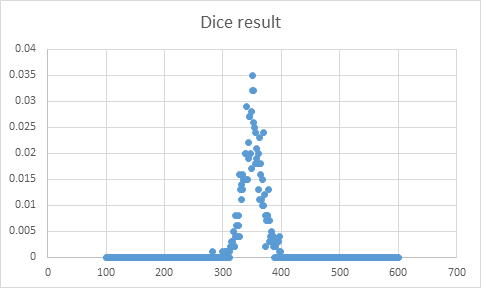
Error = ABS ( Dice result – Gaussian result )



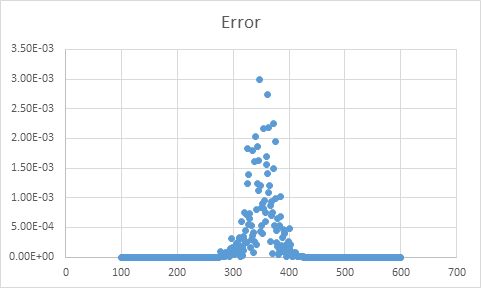
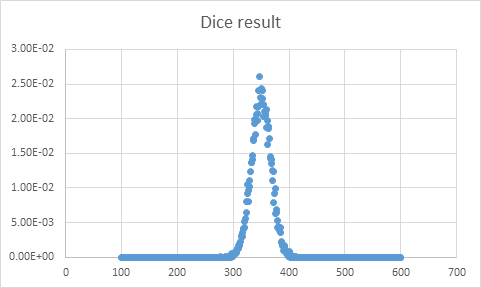
1. 던진 횟수 N = 10 2. 던진 횟수 N = 100



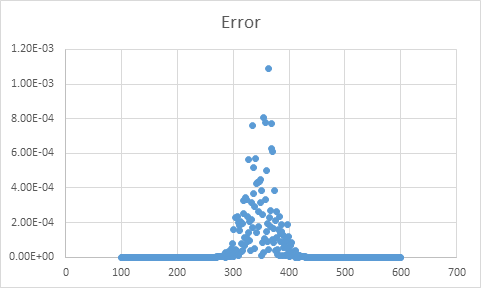
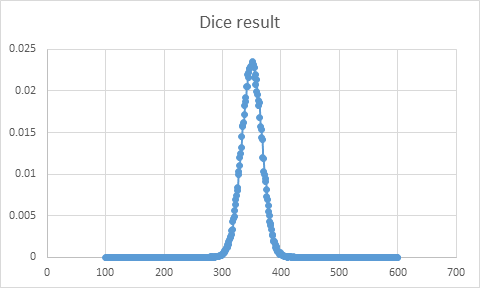
3. 던진 횟수 N = 1000 



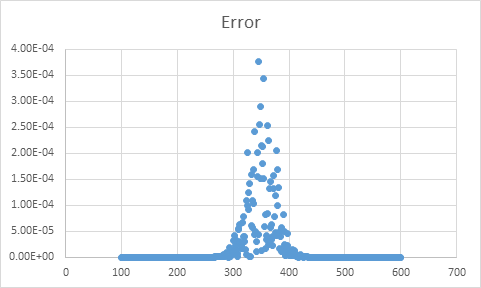
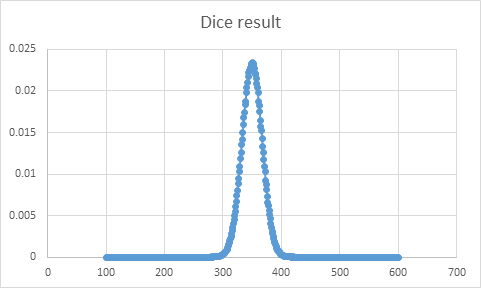
4. 던진 횟수 N = 10000 =  



5. 던진 횟수 N = 100000 =  



6. 던진 횟수 N =  



(dice result – gaussian result)

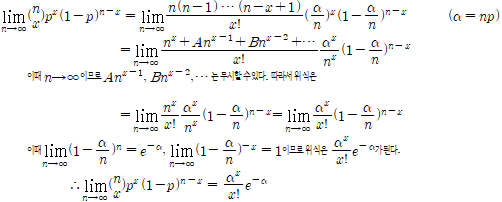
|  |  |
| --- | --- |
| 시행 횟수 (n) | Noise |
| 10 | 0.10508 |
| 100 | 0.0130305 |
| 1000 | 0.00110543 |
|  | 0.000109609 |
|  | 9.86253e-006 |
|  | 1.26977e-006 |

이때 N이 커질수록 Error그래프의 최대값이 작아지고 Gaussian pdf의 모양과 비슷해지는 것을 확인할 수 있다. 즉 시행 횟수를 늘릴수록 dice pdf가 gaussian pdf를 따라간다는 것을 알 수 있다. Noise 역시 시행 횟수가 10배 증가할수록 약 9배씩 줄어드는 것을 확인할 수 있다.

또한 위 그래프는 pdf(Probability Density Function)이므로 모든 범위의 pdf를 전부 더하면 1이 되는 것은 자명하다.

Assignment 2

1) Prove Poisson Limit



2) 이때 Poisson Distribution을 계산할 때 에서 x!를 C++로 구현하여야 하는데 C++에서는 int의 최대값이 2147483647이다.

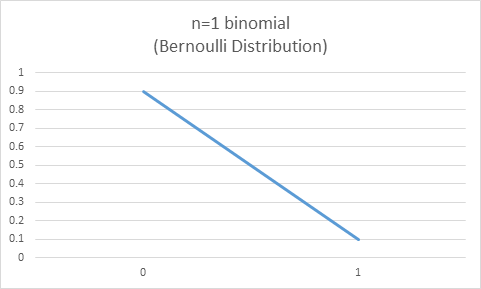
즉 14!=1278945280 < (int) < 15! = 19184179200이므로 factorial 안에 넣는 수가 14 이상만 되더라도 오류가 뜰 것이다. 실제로 x > 15일 때 값이 제대로 나오지 않거나 음수가 나오는 것을 확인할 수 있다.

|  |  |  |
| --- | --- | --- |
| x값 | 실제 x!값 | 컴파일된 x!값 |
| 14 | 1278945280 | 1278945280 |
| 15 | 19184179200 | 2004310016 |
| 16 | 306946867200 | 2004189184 |
| 17 | 5.2180967424e+12 | -288522240 |

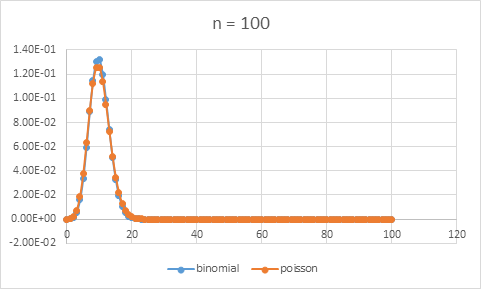
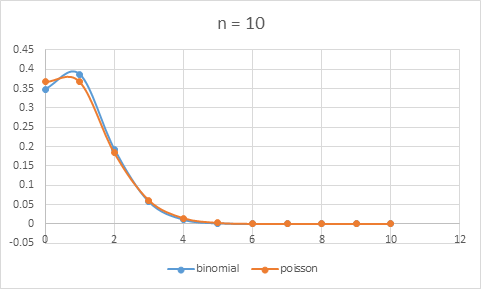
또한 double로 형식을 변경하면 int보다는 큰 값을 다룰 수 있지만 double 역시 최대값이 1.7E+308이기 때문에 x가 170을 넘어가게 되면 결과값으로 inf, 즉 오류를 출력하게 된다.

따라서 Poisson Distribution을 계산할 때 위에 주어진 식이 아닌 교수님께서 hint로 주신 코드를 이용하여 계산해보았다.

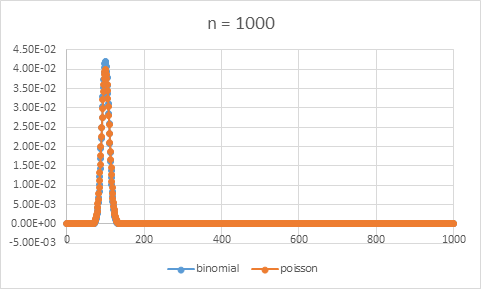
1. n = 1 (Bernoulli Distribution) 



2. n = 10 3. n = 100



4. n = 1000



(binomial - poisson)

|  |  |
| --- | --- |
| 시행 횟수 (n) | Noise |
| 10 | 0.000881347 |
| 100 | 0.000196186 |
| 1000 | 6.04194e-005 |

그래프를 보면 n이 커질수록, 즉 시행한 횟수가 늘어날수록 p의 확률, 즉 10%인 n/10에 가까워지는 것을 확인할 수 있다. 또한 noise 역시 줄어드는 것을 확인할 수 있다.

이때 poisson distribution은 현상이 정말 random한 현상인지 아닌지에 대하여 감지하는데 사용된다. 예를 들어 어떠한 현상의 poisson distribution가 poisson distribution의 그래프를 가지지 않는다면 이는 random하지 않고 의도적으로 발생했다는 것을 의미한다. 또한 poisson distribution를 time에 대해서 보면 event가 특정 time에 몰려서 나오게 되는 특징이 있다. 따라서 poisson distribution은 한정된 시간 및 공간에서 사건 발생 확률을 구하는데 유용하다. 즉 시간당 손님의 방문 수라든지, 월간 기계의 고장 횟수, 단위 길이당 균열의 발생 수 등과 같이 지정된 시간 또는 장소 등에서 어떤 사건이 발생할 확률을 예측할 수 있다.

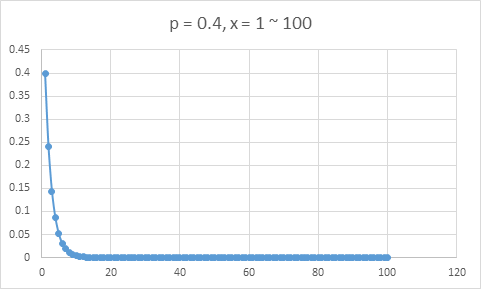
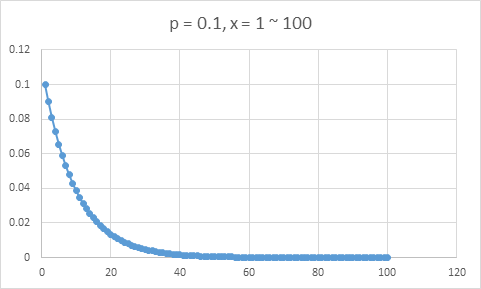
Additional Assignment 1

Geometric distribution(기하 분호)

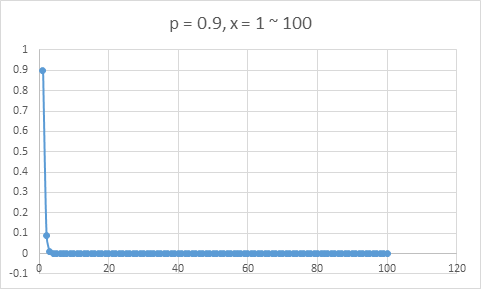
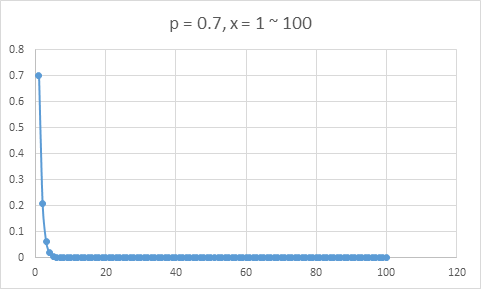
이는 성공할 확률이 p일 때 (0<=p<=1) x번 시행 후 첫 번째 성공을 얻을 확률이다.

-> 

1) p = 0.1, x = 1 ~ 100 2) p = 0.4, x = 1 ~ 100



3 ) p = 0.7, x = 1 ~ 100 4) p = 0.9, x = 1 ~ 100



이때 성공할 확률이 클수록 시행한 횟수를 늘렸을 때 확률이 급격히 낮아진다. 이는 직관적으로도 이해할 수 있는데, 성공할 확률이 70%나 90%인 경우는 시행한 횟수가 4회 이상만 되더라도 그 횟수만에 성공할 확률은 급격하게 낮아질 것이다.

Additional Assignment 2

Monte-Carlo Method ( 값 계산)

몬테카를로 방법이란 난수를 이용하여 함수의 값을 확률적으로 계산하는 알고리즘을 의미한다.

이를 이용하여 원주율 PI의 값을 계산해보았다.

1. 가로와 세로의 길이가 1인 정사각형을 그리고, 그 안에 사분원을 넣는다.
2. 정사각형 안쪽에 일정 개수의 점을 랜덤하게 찍는다.
3. 찍은 모든 점의 개수와 그 중에서 사분원 안쪽에 찍힌 점의 개수를 센다.
4. (사분원 안쪽에 찍힌 점의 개수)\*(모든 점의 개수)는 두 영역의 비율을 나타내며, 이 값에 4를 곱하면 PI값의 근사치를 추정할 수 있다.



이 과정을 C++ 코드로 구현하기 위해,

각 점의 x,y좌표를 0과 1 사이의 랜덤한 소수로 설정하고, 점 (0,0)으로부터 생성한 점까지의 거리가 1 이하인 경우 count를 증가시켰다.

이 과정을 반복문을 통해 설정한 횟수만큼 반복하게 하였고, 반복문이 종료된 후의 총 count값을 반복횟수로 나누고 4를 곱해서 PI 값을 추정해보았다.

|  |  |  |
| --- | --- | --- |
|  | 사분원 안에 찍힌 점의 개수 | PI |
| n = 100 | 85 | 3,4 |
| n = 1000 | 803 | 3.212 |
| n = 5000 | 3963 | 3.1704 |
| n = 10000 | 7939 | 3.1756 |
| n = 30000 | 23684 | 3.15787 |

실행 횟수가 적을 때는 이론적인 값과 비교적 차이가 큰 값이 도출되었으나, 실행 횟수를 급격히 증가시킬수록 어느 정도 이론적인 PI 값에 수렴하는 경향을 보였다.

x,y 좌표를 랜덤하게 설정하여 사분원 안에 찍히는 점의 개수가 컴파일할 때마다 달라지기 때문에, 몇 번의 실행 결과만 놓고 보면 n 값이 증가하는 만큼 더욱 이론적인 값에 수렴하지는 않는 것처럼 보일 수 있다. 하지만 x,y의 값을 더욱 균일하게 분포시키고, 실행 횟수를 매우 많게 한다면 평균적으로 점점 개선되는 경향을 보일 것이다.