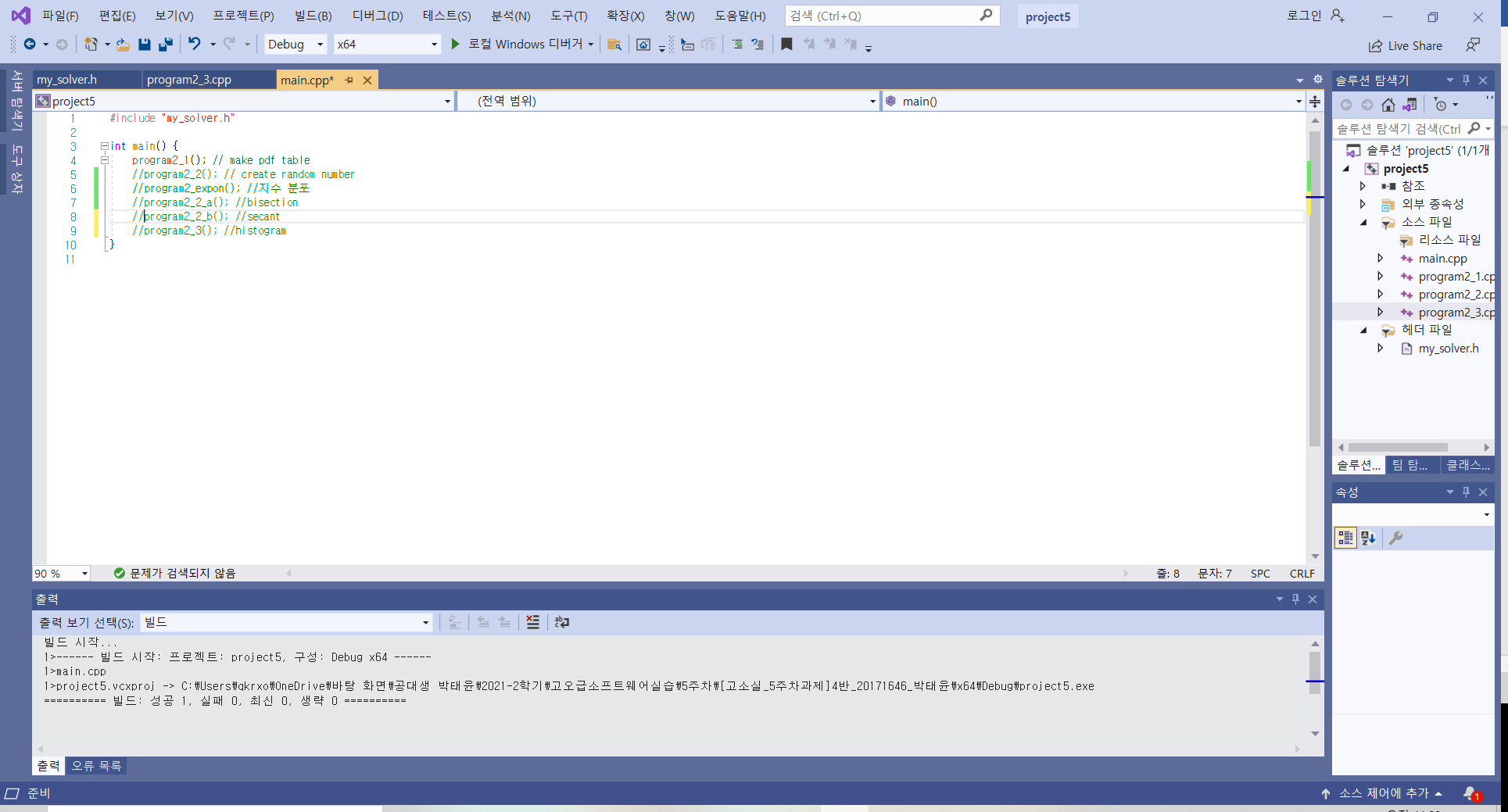
고급소프트웨어실습I

5주차 보고서

20171646 박태윤

- 프로그램 구동 방법 및 소개



현재 메인 함수는 다음과 같이 작성이 되어있습니다.

5줄 주석 처리가 되어있는데, pdf를 만드는 program2\_1()을 항상 먼저 동작시키고, 나머지 부분에 대해서 주석 처리를 뺀 후에 동작을 시킬 수 있는데 이는 다음과 같습니다.

- program2\_2() : 실습 2번

텍스트이(가) 표시된 사진

자동 생성된 설명

각 구간에 따른 넓이가 먼저 출력이 되고 이후 생성하고자 하는 난수의 개수를 입력받을 수 있습니다.

- program2\_2\_a() : 과제 2번(bisection을 이용한 난수 생성)

텍스트이(가) 표시된 사진

자동 생성된 설명

생성하고자 하는 난수를 입력 받는 과정은 2\_2에서와 같지만 마지막에 실행 시간이 출력이 됩니다.

- program2\_2\_b() : 과제 2번(secant를 이용한 난수 생성)

텍스트이(가) 표시된 사진

자동 생성된 설명

동작 방식은 program2\_2\_a와 같습니다.

- program2\_expon() : 과제 1번(지수 분포 함수, 람다 입력 및 평균 + 분산 비교)

텍스트이(가) 표시된 사진

자동 생성된 설명

프로그램 2\_expon은 난수 생성 개수를 입력 받은 뒤 지수 분포 함수의 람다값을 입력받습니다. 총 3번을 입력받을 수 있으며

텍스트이(가) 표시된 사진

자동 생성된 설명

람다 값을 입력하면 theory줄에는 이론적인 (평균, 분산)값, calculate에는 inverse방법을 이용하여 계산한 (평균, 분산)값이 출력이 됩니다.

- program2\_3() : 과제 2번(histogram.txt 생성)

현재 만들어진 random\_event\_table.txt, pdf\_table.txt를 통해 히스토그램 데이터를 구축하는 프로그램입니다. program2\_2\_a, program2\_2\_b에 대해 사용할 수 있는데 가장 최근에 만들어진 random\_event\_table.txt, pdf\_table.txt를 통해 히스토그램 데이터를 만듭니다.

- program2\_2\_c() : 과제 2번(뉴턴 랩슨 방법)

뉴턴 랩슨 방법을 이용하여 역함수 값을 계산하는 프로그램입니다. 동작 방식은 program2\_2\_a, program2\_2\_b와 같습니다.

**같은 이름의 텍스트 파일(random\_event\_table.txt 등)에 내용을 입력을 하기 때문에 전체 실행이 아닌 한 프로그램 마다 주석을 한 줄씩 해제시키면서 동작시키는걸 추천드립니다.**

**- 실습 1**

CurveSampling프로그램을 이용해 그래프를 그려 샘플링을 한 뒤 이 데이터를 이용해 정규화 된 pdf를 만들고 이를 이용해 각 구간마다 적분을 하여 구간마다 넓이를 출력합니다. 총 넓이는 1과 매우 근사해야 합니다. 이에 대한 코드는 다음과 같이 작성하였습니다.

void program2\_1()

{

FILE\* fp\_r, \*fp\_w;

\_\_int64 start, freq, end;

float resultTime = 0;

fp\_r = fopen("sampling\_table.txt", "r");

if (fp\_r == NULL) {

printf("input file not found....\n");

exit(0);

}

fp\_w = fopen("pdf\_table.txt", "w");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double sample, x;

fscanf(fp\_r, "%lf %lf", &sample, &x);

double\* arr\_x = (double\*)malloc(sizeof(double) \* sample);

double\* arr\_y = (double\*)malloc(sizeof(double) \* sample);

for (int i = 0; i < sample; i++) {

fscanf(fp\_r, "%lf %lf", &arr\_x[i], &arr\_y[i]);

}

//x정규화 [0, 1]

for (int i = 0; i < sample; i++) {

arr\_x[i] = i \* (1.0 / (sample - 1));

}

//y정규화

double sum\_height = 0;

for (int i = 0; i < sample; i++) {

sum\_height += arr\_y[i];

}

sum\_height = sum\_height \* 2.0 - (arr\_y[0] + arr\_y[(int)sample - 1]);

for (int i = 0; i < sample; i++) {

arr\_y[i] = arr\_y[i] / (sum\_height \* (1.0 / (sample - 1)) \* 0.5);

}

double result[4];

for (int i = 0; i < 4; i++) {

double cal = 0.0;

for (int j = 0; j < sample; j++) {

if (i \* 0.25 <= arr\_x[j] && arr\_x[j] <= (i + 1) \* 0.25)

cal += arr\_y[j];

if (i \* 0.25 < arr\_x[j] && arr\_x[j] < (i + 1) \* 0.25)

cal += arr\_y[j];

}

cal = cal \* (1.0 / (sample - 1)) \* 0.5;

result[i] = cal;

}

double sum\_of\_res = 0.0;

for (int i = 0; i < 4; i++)

sum\_of\_res += result[i];

printf("\*\*\* Integrating the pdf from 0.0 to 1.0 = %lf\n", sum\_of\_res);

printf("\*\*\* Integrating the pdf from 0.0 to 0.25 = %lf\n", result[0]);

printf("\*\*\* Integrating the pdf from 0.25 to 0.5 = %lf\n", result[1]);

printf("\*\*\* Integrating the pdf from 0.5to 0.75 = %lf\n", result[2]);

printf("\*\*\* Integrating the pdf from 0.75 to 1.0 = %lf\n", result[3]);

fprintf(fp\_w, "%d %lf\n", (int)sample, (1.0 / (sample - 1)));

for (int i = 0; i < sample; i++) {

fprintf(fp\_w, "%lf %lf\n", arr\_x[i], arr\_y[i]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

free(arr\_x);

free(arr\_y);

if (fp\_r != NULL) fclose(fp\_r);

if (fp\_w != NULL) fclose(fp\_w);

}

생성이 된 sampling\_table.txt를 읽어 첫 줄에 있는 sampling개수와 간격은 각각 sample, x에 저장을 합니다. 이후 실제 값에 대한 내용은 각각 arr\_x, arr\_y배열에 저장을 합니다. 이후 정규화 과정을 거치는데, 배열의 인덱스는 0부터 시작을 하기 때문에 arr\_x에 대한 정규화는 인덱스 \* 1/(샘플링 개수 – 1)로 해주었습니다. 이후 arr\_y에 대한 정규화 과정을 거치는데, 이는 현재 arr\_y값에 전체 적분을 한 값을 나누는 것으로 할 수 있다. 사다리꼴 적분 방법을 이용하였는데, 사다리꼴의 윗변과 아랫변을 더한 뒤 높이를 곱하고 2로 나누는 식으로 할 수 있습니다. 여기에서 첫번째 사다리꼴은 arr\_y[0] + arr\_y[1], 두번째 사다리꼴은 arr\_y[1] + arr\_y[2], …. , 마지막 사디리꼴은 arr\_y[sample-2] + arr\_y[sample-1]값이 윗변 + 아랫변의 합이기 때문에 총 윗변과 아랫변을 더해준 값은 전체 arr\_y의 값을 더한 뒤 두 배를 하고 한번 씩만 더해주는 arr\_y[0]과 arr\_y[sample-1]을 빼주면 됩니다. 높이의 값은 정규화가 되었기 때문에 1 / (샘플링 개수 – 1)로 해주면 됩니다. 이렇게 해서 arr\_y에 대한 정규화를 할 수 있습니다. 이후 각 구간 별로 적분한 값을 구해줘야 하는데 이는 arr\_x에 대해 범위를 설정해 가면서 해당 범위 안에 대응이 되는 arr\_y를 모두 더해주고 거기에 높이인 1 / (샘플링 개수 – 1)을 더해준 뒤 2로 나눠주는, 앞에서 arr\_y정규화에 사용된 사다리꼴 적분 방식과 동일합니다. 이 각 구간에 대한 적분값의 정보를 result라는 배열에 저장을 해 출력을 시킵니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

실제로도, (0.0 ~ 1.0)에 대한 적분 값은 이론과 같이 1이 나오는 것을 확인할 수 있습니다.

**- 실습2**

실습1에서 만든 pdf를 이용해 cdf를 만들고 (0, 1)의 난수 값에 대해 cdf에서 이에 대응하는 값을 bisection방법을 이용하여 찾는 것이 실습의 목표입니다. 코드는 다음과 같이 작성하였습니다.

double cal\_remain(double mid, int idx, double arr\_x[], double arr\_y[], double cdf[]) {

return (arr\_y[idx] + (arr\_y[idx + 1] - arr\_y[idx]) / (arr\_x[idx + 1] - arr\_x[idx]) \* (mid - arr\_x[idx]) \* 0.5) \* (mid - arr\_x[idx]);

}

void program2\_2()

{

FILE\* fp\_r, \* fp\_w;

//FILE\* fp\_w2;

fp\_r = fopen("pdf\_table.txt", "r");

fp\_w = fopen("random\_event\_table.txt", "w");

//fp\_w2 = fopen("histogram.txt", "w");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double num, x;

fscanf(fp\_r, "%lf %lf", &num, &x);

double\* arr\_x = (double\*)malloc(sizeof(double) \* num);

double\* arr\_y = (double\*)malloc(sizeof(double) \* num);

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

fscanf(fp\_r, "%lf %lf", &arr\_x[i], &arr\_y[i]);

double\* cdf = (double\*)malloc(sizeof(double) \* num);

double sum = 0.0;

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

cdf[i] = 0.0;

cdf[0] = arr\_y[0] \* x \* 0.5;

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

cdf[i] = x \* (arr\_y[i - 1] + arr\_y[i]) \* 0.5 + cdf[i - 1];

}

int rd\_num;

printf("input random number : ");

scanf("%d", &rd\_num);

fprintf(fp\_w, "%d\n", rd\_num);

unsigned int iseed = (unsigned int)time(NULL);

srand(iseed);

for (int i = 0; i < rd\_num; i++) {

double U = (rand() % 10000) / 10000.0;

double start = arr\_x[0];

double end = arr\_x[(int)num - 1];

double last = 0.0;

int count = 0;

while (1) {

int idx = 0;

double mid = (start + end) \* 0.5;

for (int j = 0; j < num; j++) {

if (mid >= arr\_x[j] && mid <= arr\_x[j + 1]) {

idx = j;

break;

}

}

double res = cdf[idx] + cal\_remain(mid, idx, arr\_x, arr\_y, cdf);

if (res - U < 0) {

start = mid;

}

else {

end = mid;

}

if (fabs(res - U) < DELTA || count >= Nmax || fabs(mid - last) < EPSILON) {

fprintf(fp\_w, "%.15lf\n", mid);

break;

}

last = mid;

count++;

}

}

free(arr\_x);

free(arr\_y);

free(cdf);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (fp\_r != NULL) fclose(fp\_r);

if (fp\_w != NULL) fclose(fp\_w);

//if (fp\_w2 != NULL) fclose(fp\_w2);

}

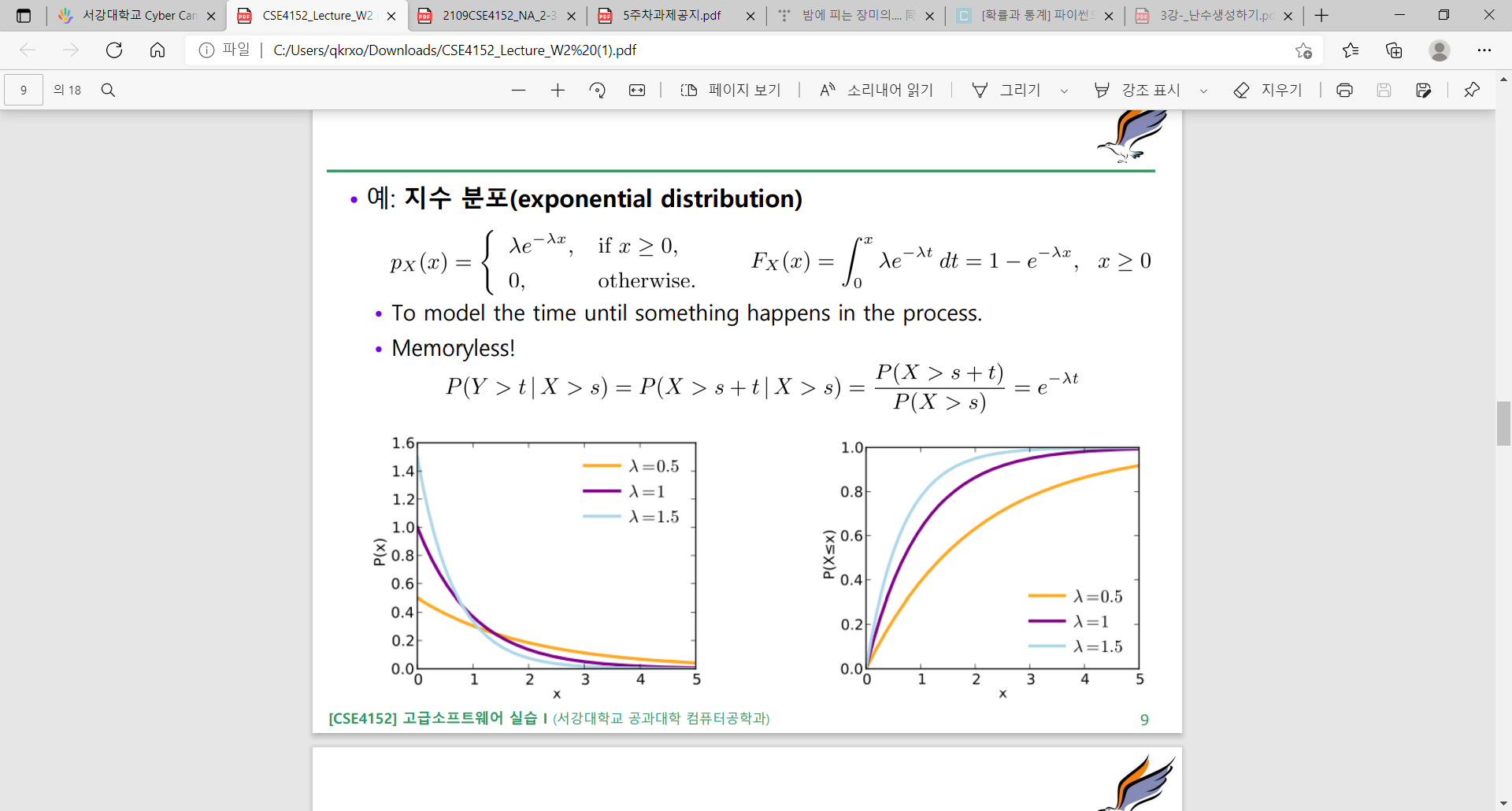
실습1에서 만든 pdf\_table.txt를 읽어 첫 줄에 적혀져 있는 개수와 간격을 각각 num과 x에 대입합니다. 이 후 pdf\_table의 데이터 값은 각각 arr\_x, arr\_y에 집어넣습니다. 이후 cdf라는 배열을 만드는데, cdf는 누적 분포 함수이기 때문에 pdf의 적분 값을 넣어줘야 합니다. ‘누적’ 분포이므로 cdf[i]에는 이전 적분 값인 cdf[i-1]에 현재 보고 있는 칸의 사다리꼴 적분 x \* (arr\_y[i-1] + arr\_y[i]) \* 0.5을 더해줍니다. 이 후 생성하고자 하는 난수의 개수를 입력을 rd\_num으로 받고, random\_event\_table.txt에 써줍니다. 난수를 시드 넘버 iseed, 실질적인 난수 값은 U로 만들었는데, U = (rand() % 10000) / 10000.0으로 0.0000 ~ 0. 9999사이의 값을 나타냅니다. 이후 bisection방법을 이용한 역함수 값을 계산하는데, bisection의 시작점 start는 pdf의 시작점 arr\_x[0], 끝점 end는 pdf의 마지막 지점 arr\_x[num-1]로 지정을 하고 시작을 합니다. bisection은 근에 계속 수렴해가는 중앙값(start와 end의 평균)으로 계산을 하는데 이를 나타내는게 mid변수입니다. mid변수를 계산을 하고 이 지점까지의 pdf적분 값을 계산합니다. 이전에 cdf를 구했기 때문에 mid가 cdf에서 몇 번째 인덱스에서 나타나는지를 먼저 구한 다음에 해당 cdf값을 더하고 이후 남아있는 부분은 자료 8page에 나와있는 공식을 이용하여 계산을 했습니다. 이는 함수 cal\_remain으로 구했으며 적분 값을 모두 계산한 뒤 이 결과를 res에 저장을 하고 res가 생성한 난수 U보다 작으면 start = mid, res가 U보다 크면 end = mid를 하여 계속해서 근 U와 mid를 근접하게 계산을 하였습니다. 지난 주차 실습에서와 마찬가지로 근과 계산한 근사치가 DELTA보다 작거나 최대 계산 횟수 Nmax를 넘었거나 계속해서 계산해서 이동하는 mid가 EPSILON보다 작은, 더 이상 의미있는 전진을 하지 않는 경우에는 현재 mid값을 random\_event\_table에 저장을 하고 계산을 종료하였습니다. 계산 횟수는 count변수가 나타냅니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 첫 줄에 생성한 난수의 개수, 이후 난수에 대응하는 계산한 X값을 random\_event\_table에 저장하였습니다.

**- 과제1**



지수 분포에 대한 식입니다. 이 분포에서 이론적인 평균값은(l은 람다를 의미합니다) E[x] = , 분산은 V[x] = 를 나타내는데, 실습 2번과 유사한 방식으로 충분한 개수의 난수를 생성하여 계산한 역함수 값에 대한 평균과 분산이 해당 이론적인 값과 유사한지를 봅니다. 이 때, 지수 분포는 식이 주어져 있고 역함수를 구할 수 있기 때문에 bisection을 이용하지 않고 실제 역함수에 생성한 난수를 대입을 하였습니다. 지수 분포의 cdf인 의 역함수는 x = -(1 - ) / l로 구할 수 있습니다. 과제 1에 대한 코드는 다음과 같이 작성하였습니다.

double inverse\_func(double y, double L) {

return -(log(1 - y) / L);

}

// HOMEWORK

void program2\_expon()

{

int rd\_num;

double lambda;

printf("input random number : ");

scanf("%d", &rd\_num);

for (int k = 0; k < 3; k++) {

printf("input lambda : ");

scanf("%lf", &lambda);

double\* arr = (double\*)malloc(sizeof(double) \* rd\_num);

unsigned int iseed = (unsigned int)time(NULL);

double sum = 0.0;

srand(iseed);

for (int i = 0; i < rd\_num; i++) {

double U = (rand() % 10000) / 10000.0;

//printf("%lf\n", U);

arr[i] = inverse\_func(U, lambda);

sum += arr[i];

}

double avg = sum / (double)rd\_num;

double var = 0.0;

for (int i = 0; i < rd\_num; i++) {

arr[i] = arr[i] - avg;

}

double pyun = 0.0;

for (int i = 0; i < rd\_num; i++) {

pyun += pow(arr[i], 2);

}

var = pyun / (double)rd\_num;

printf("theory : %lf %lf\n", 1.0 / lambda, 1.0 / pow(lambda, 2));

printf("calculate : %lf %lf\n\n", avg, var);

free(arr);

}

}

각기 다른 람다 값에 대해 평균과 분산을 비교하는데, 이 과정은 총 3번 반복을 하였습니다. 그 전에 생성하고자 하는 난수의 개수를 먼저 rd\_num에 입력을 받고 난수 U를 실습2에서와 같이 생성을 합니다. 이 U가 에 해당하고, 앞에서 구한 역함수 값을 계산해주는 inverse\_func(U, lambda)에 대입을 하여 이에 대응하는 x값을 arr[i]에 저장을 해주었습니다. 저장을 해주면서 계산한 값은 sum에 누적하였고, 최종적으로 rd\_num만큼 이 과정을 반복한 뒤에는 평균(avg)는 sum을 난수 개수인 rd\_num으로 나눈 값으로 계산을 하였습니다. 이후 편차의 제곱의 평균인 분산을 계산하여 var에 대입을 하였고, 이론적인 평균과 분산 값은 theory : %lf %lf로, 난수를 생성하여 inversion방법을 통해 계산한 평균과 분산 값은 calculate : %lf %lf로 콘솔창에 출력을 하였습니다. 다음은 난수 생성 개수를 다르게 하여 나타낸 결과들입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

난수를 50개 생성하여 이에 대한 평균과 분산 값을 비교했을 시에는 이론적인 평균, 분산 값과 다소 큰 차이를 보이는 것을 확인할 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

난수를 100개를 생성했을 시에는 50개를 생성했을 때보단 좀 더 적은 차이를 보이지만, 프로그램을 여러 번 돌리다 보면 실제 생성된 난수 U값에 의해 차이를 많이 나타내는 경우가 종종 있음을 확인할 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

난수를 1000개 생성한 결과입니다. 이전 50, 100개를 생성했을 때보다 훨씬 적은 차이를 보이는 것을 확인할 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

난수를 10000개 생성했을 때의 결과입니다. 이론적인 값과 유사하지만, 정확도가 1000개를 생성했을 때보다 매우 뛰어나다고 하기에는 어려운 값을 얻은 것을 확인할 수 있습니다.

난수를 1000개 정도 생성했을 시에, 이론적인 값과 어느 정도 유사한 평균과 분산을 얻을 수 있음을 확인하였습니다.

**-과제 2**

(program2\_2\_a)

난수를 생성하여 bisection방법을 이용해 역함수 값을 구하는 프로그램입니다. 이에 대한 코드는 다음과 같습니다.

double cal\_remain2(double mid, int idx, double arr\_x[], double arr\_y[], double cdf[]) {

return (arr\_y[idx] + (arr\_y[idx + 1] - arr\_y[idx]) / (arr\_x[idx + 1] - arr\_x[idx]) \* (mid - arr\_x[idx]) \* 0.5) \* (mid - arr\_x[idx]);

}

void program2\_3() {

FILE\* fp\_r, \* fp\_r2;

FILE\* fp\_w;

fp\_r = fopen("pdf\_table.txt", "r");

fp\_r2 = fopen("random\_event\_table.txt", "r");

fp\_w = fopen("histogram.txt", "w");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double num, x;

fscanf(fp\_r, "%lf %lf", &num, &x);

double\* arr\_x = (double\*)malloc(sizeof(double) \* num);

double\* arr\_y = (double\*)malloc(sizeof(double) \* num);

int arrange[100];

for (int i = 0; i < 100; i++)

arrange[i] = 0;

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

fscanf(fp\_r, "%lf %lf", &arr\_x[i], &arr\_y[i]);

double sum = 0.0;

int num\_of\_nan = 0;

fscanf(fp\_r2, "%d", &num\_of\_nan);

double\* ran\_event = (double\*)malloc(sizeof(double) \* num\_of\_nan);

for (int i = 0; i < num\_of\_nan; i++)

fscanf(fp\_r2, "%lf", &ran\_event[i]);

//printf("sss\n");

for (int i = 0; i < num\_of\_nan; i++) {

int mok = ran\_event[i] \* 100;

//printf("%lf\n", ran\_event[i]);

arrange[mok]++;

}

for (int i = 0; i < 100; i++)

fprintf(fp\_w, "%d\n", arrange[i]);

free(arr\_x);

free(arr\_y);

free(ran\_event);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (fp\_r != NULL) fclose(fp\_r);

if (fp\_r2 != NULL) fclose(fp\_r2);

if (fp\_w != NULL) fclose(fp\_w);

}

// HOMEWORK

void program2\_2\_a()

{

\_\_int64 start, freq, end;

float resultTime = 0;

FILE\* fp\_r, \* fp\_w;

fp\_r = fopen("pdf\_table.txt", "r");

fp\_w = fopen("random\_event\_table.txt", "w");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double num, x;

fscanf(fp\_r, "%lf %lf", &num, &x);

double\* arr\_x = (double\*)malloc(sizeof(double) \* num);

double\* arr\_y = (double\*)malloc(sizeof(double) \* num);

int arrange[100];

for (int i = 0; i < 100; i++)

arrange[i] = 0;

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

fscanf(fp\_r, "%lf %lf", &arr\_x[i], &arr\_y[i]);

double\* cdf = (double\*)malloc(sizeof(double) \* num);

double sum = 0.0;

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

cdf[i] = 0.0;

cdf[0] = arr\_y[0] \* x \* 0.5;

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

cdf[i] = x \* (arr\_y[i - 1] + arr\_y[i]) \* 0.5 + cdf[i - 1];

}

//for (int i = 0; i < num; i++)

//printf("%lf\n", cdf[i]);

int rd\_num;

printf("input random number : ");

scanf("%d", &rd\_num);

fprintf(fp\_w, "%d\n", rd\_num);

CHECK\_TIME\_START;

unsigned int iseed = (unsigned int)time(NULL);

srand(iseed);

for (int i = 0; i < rd\_num; i++) {

double U = (rand() % 10000) / 10000.0;

double start2 = arr\_x[0];

double end2 = arr\_x[(int)num - 1];

double last = 0.0;

int count = 0;

while (1) {

int idx = 0;

double mid = (start2 + end2) \* 0.5;

for (int j = 0; j < num; j++) {

if (mid >= arr\_x[j] && mid <= arr\_x[j + 1]) {

idx = j;

break;

}

}

double res = cdf[idx] + cal\_remain2(mid, idx, arr\_x, arr\_y, cdf);

if (res - U < 0) {

start2 = mid;

}

else {

end2 = mid;

}

if (fabs(res - U) < DELTA || count >= Nmax || fabs(mid - last) < EPSILON) {

int mok = mid \* 100;

arrange[mok]++;

fprintf(fp\_w, "%.15lf\n", mid);

break;

}

last = mid;

count++;

}

}

free(arr\_x);

free(arr\_y);

free(cdf);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (fp\_r != NULL) fclose(fp\_r);

if (fp\_w != NULL) fclose(fp\_w);

CHECK\_TIME\_END(resultTime);

printf("The program2\_2\_a run time is %f(ms)..\n", resultTime \* 1000.0);

}

CHECK\_TIME\_START, CHECK\_TIME\_END를 통해 수행 시간을 구합니다. start는 난수 시드를 정하기 직전부터 시작을 하였습니다. 실행 결과는 다음과 같습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

총 1000000개의 난수를 생성하였고, 이에 대한 bisection의 수행 시간은 총 8532.641411ms가 걸린 것을 확인할 수 있습니다.

(program2\_2\_b)

프로그램2\_2\_b는 앞선 프로그램 2\_2\_a와 달리 bisection방법이 아니라 secant방법을 이용하여 난수에 대응하는 역함수 값을 계산하였습니다. 이전 주차 실습에서 bisection방법보다는 secant방법을 이용하면 구하고자 하는 근에 조금 더 빠른 속도로 수렴하는 것을 확인하였고 이에 따라 같은 데이터 분포에 대해 같은 수의 난수를 생성하여 계산을 했을 시 프로그램 수행 속도가 좀 더 빠르게 나오는 것을 기대할 수 있습니다. 다만, secant방법은 초기 구간을 잘 설정을 해야하는데, 이를 위해 bisection방법을 딱 1번 수행하여 실제 근과 어느 정도 근접한 구간을 지정해 secant방법을 수행하였습니다. 이에 대한 코드는 다음과 같습니다.

double cal\_remain3(double mid, int idx, double arr\_x[], double arr\_y[], double cdf[], double u) {

return cdf[idx] + (arr\_y[idx] + (arr\_y[idx + 1] - arr\_y[idx]) / (arr\_x[idx + 1] - arr\_x[idx]) \* (mid - arr\_x[idx]) \* 0.5) \* (mid - arr\_x[idx]) - u;

}

void program2\_2\_b()

{

\_\_int64 start, freq, end;

float resultTime = 0;

FILE\* fp\_r, \* fp\_w;

fp\_r = fopen("pdf\_table.txt", "r");

fp\_w = fopen("random\_event\_table.txt", "w");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//float x;

//int num;

double num, x;

fscanf(fp\_r, "%lf %lf", &num, &x);

double\* arr\_x = (double\*)malloc(sizeof(double) \* num);

double\* arr\_y = (double\*)malloc(sizeof(double) \* num);

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

fscanf(fp\_r, "%lf %lf", &arr\_x[i], &arr\_y[i]);

double\* cdf = (double\*)malloc(sizeof(double) \* num);

double sum = 0.0;

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

cdf[i] = 0.0;

cdf[0] = arr\_y[0] \* x \* 0.5;

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

cdf[i] = x \* (arr\_y[i - 1] + arr\_y[i]) \* 0.5 + cdf[i - 1];

}

int rd\_num;

printf("input random number : ");

scanf("%d", &rd\_num);

fprintf(fp\_w, "%d\n", rd\_num);

CHECK\_TIME\_START;

unsigned int iseed = (unsigned int)time(NULL);

srand(iseed);

for (int i = 0; i < rd\_num; i++) {

double U = (rand() % 10000) / 10000.0;

double start2 = arr\_x[0];

double end2 = arr\_x[(int)num - 1];

double last = 0.0;

int count = 0;

double mid = 0.0;

//bisection

for(int l = 0; l < 1; l++){

int idx = 0;

mid = (start2 + end2) \* 0.5;

for (int j = 0; j < num; j++) {

if (mid >= arr\_x[j] && mid <= arr\_x[j + 1]) {

idx = j;

break;

}

}

double res = cdf[idx] + cal\_remain2(mid, idx, arr\_x, arr\_y, cdf);

if (res - U < 0) {

start2 = mid;

}

else {

end2 = mid;

}

last = mid;

}

double a = start2;

double b = end2;

double cal\_arr[Nmax + 5];

int idx\_a = 0;

int idx\_b = 0;

for (int j = 0; j < num; j++) {

if (a >= arr\_x[j] && a <= arr\_x[j + 1]) {

idx\_a = j;

break;

}

}

for (int j = 0; j < num; j++) {

if (b >= arr\_x[j] && b <= arr\_x[j + 1]) {

idx\_b = j;

break;

}

}

double f\_b = cal\_remain3(b, idx\_b, arr\_x, arr\_y, cdf, U);

double f\_a = cal\_remain3(a, idx\_a, arr\_x, arr\_y, cdf, U);

cal\_arr[0] = f\_a;

cal\_arr[1] = f\_b;

//secant

while(1){

int idx\_tmp = 0;

double tmp = b - cal\_arr[count+1] \* ((b - a) / (cal\_arr[count + 1] - cal\_arr[count]));

for (int j = 0; j < num; j++) {

if (tmp >= arr\_x[j] && tmp <= arr\_x[j + 1]) {

idx\_tmp = j;

break;

}

}

cal\_arr[count + 2] = cal\_remain3(tmp, idx\_tmp, arr\_x, arr\_y, cdf, U);

if (fabs(cal\_arr[count+2]) < DELTA || count >= Nmax || fabs(tmp - b) < EPSILON) {

//printf("%.15lf %lf %lf\n", cal\_arr[count+2], tmp, U);

fprintf(fp\_w, "%.15lf\n", tmp);

break;

}

a = b;

b = tmp;

count++;

}

}

free(arr\_x);

free(arr\_y);

free(cdf);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (fp\_r != NULL) fclose(fp\_r);

if (fp\_w != NULL) fclose(fp\_w);

CHECK\_TIME\_END(resultTime);

printf("The program2\_2\_b run time is %f(ms)..\n", resultTime \* 1000.0);

}

bisection을 이용하면, 초기에 그래프 첫 지점으로 지정해놓은 start2와 끝 점으로 지정해놓은 end2가 계속 실제 근에 수렴을 하고 이를 1번 수행하여 얻은 start2와 end2를 secant에서 a,b로 설정하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

시컨트 방법은 다음과 같은 형식으로 계산을 하는데, a는 여기서 , b는 을 나타냅니다. 이 후 그 지점까지의 적분 값을 계산해주는 cal\_remain3함수를 이용하여 각각 f\_a, f\_b에 대응을 시키고 이를 cal\_arr배열에 저장을 한 뒤 시컨트 방법을 수행시켰습니다. 효율성을 위해 cal\_arr배열에 저장한 값을 계산마다 계속 사용하였으며 역시 EPSILON, DELTA, Nmax조건에 걸릴 시에 계산을 종료하였습니다. 프로그램 수행 결과는 다음과 같습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

같은 데이터에 대해 같은 수의 난수에 대한 계산을 했을 시에 실제로 bisection방법보다 더 적은 시간이 걸린 것을 확인할 수 있습니다.

초기값을 바이섹션 방법을 2번 돌린 start2와 end2값으로 설정하여 시컨트 계산을 진행했는데, 만약 이 과정 없이 시컨트의 초기값을 단순하게 0.0, 1.0으로 설정한 뒤 계산을 진행하면 다음과 같은 결과를 얻게 됩니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

딱 한번만 바이섹션을 돌려 구하고자 하는 값이 0.0에 가까운지 1.0에 가까운지를 보고 0.0에 가깝다면 0.0 ~ 0.5를, 1.0에 가깝다면 0.5 ~ 1.0을 보는 느낌으로 초기값을 바이섹션을 통해 설정을 한 결과가 단순하게 0.0, 1.0으로 초기값을 설정하여 시컨트 계산을 진행한 것 보다 성능이 개선된 것을 확인할 수 있습니다.

(program2\_2\_c)

newton-raphson방법을 이용하여 값을 구하는 프로그램입니다. 이에 대한 코드는 다음과 같습니다.

void program2\_2\_c()

{

\_\_int64 start, freq, end;

float resultTime = 0;

FILE\* fp\_r, \* fp\_w;

fp\_r = fopen("pdf\_table.txt", "r");

fp\_w = fopen("random\_event\_table.txt", "w");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double num, x;

fscanf(fp\_r, "%lf %lf", &num, &x);

double\* arr\_x = (double\*)malloc(sizeof(double) \* num);

double\* arr\_y = (double\*)malloc(sizeof(double) \* num);

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

fscanf(fp\_r, "%lf %lf", &arr\_x[i], &arr\_y[i]);

double\* cdf = (double\*)malloc(sizeof(double) \* num);

double sum = 0.0;

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

cdf[i] = 0.0;

cdf[0] = arr\_y[0] \* x \* 0.5;

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

cdf[i] = x \* (arr\_y[i - 1] + arr\_y[i]) \* 0.5 + cdf[i - 1];

}

int rd\_num;

printf("input random number : ");

scanf("%d", &rd\_num);

fprintf(fp\_w, "%d\n", rd\_num);

CHECK\_TIME\_START;

unsigned int iseed = (unsigned int)time(NULL);

srand(iseed);

for (int i = 0; i < rd\_num; i++) {

double U = (rand() % 10000) / 10000.0;

double start2 = arr\_x[0];

double end2 = arr\_x[(int)num - 1];

double last = 0.0;

int count = 0;

double mid = 0.0;

int idx = num / 2.0;

//bisection

for (int l = 0; l < 2; l++) {

idx = 0;

mid = (start2 + end2) \* 0.5;

for (int j = 0; j < num; j++) {

if (mid >= arr\_x[j] && mid <= arr\_x[j + 1]) {

idx = j;

break;

}

}

double res = cdf[idx] + cal\_remain2(mid, idx, arr\_x, arr\_y, cdf);

if (res - U < 0) {

start2 = mid;

}

else {

end2 = mid;

}

last = mid;

}

double a = arr\_x[idx];

double cal\_arr[Nmax + 5];

int idx\_a = 0;

for (int j = 0; j < num; j++) {

if (a >= arr\_x[j] && a <= arr\_x[j + 1]) {

idx\_a = j;

break;

}

}

double f\_a = cal\_remain3(a, idx\_a, arr\_x, arr\_y, cdf, U);

cal\_arr[0] = f\_a;

//newton

while (1) {

//int idx\_tmp = 0;

double s = (a - arr\_x[idx\_a]) / (arr\_x[idx\_a + 1] - arr\_x[idx\_a]);

double tmp = a - cal\_arr[count] / ((1 - s) \* arr\_y[idx\_a] + s \* arr\_y[idx\_a + 1]);

for (int j = 0; j < num; j++) {

if (tmp >= arr\_x[j] && tmp <= arr\_x[j + 1]) {

idx\_a = j;

break;

}

}

cal\_arr[count + 1] = cal\_remain3(tmp, idx\_a, arr\_x, arr\_y, cdf, U);

if (fabs(cal\_arr[count + 1]) < DELTA || count >= Nmax || fabs(tmp - a) < EPSILON) {

fprintf(fp\_w, "%.15lf\n", tmp);

break;

}

a = tmp;

//idx\_a = idx\_tmp;

count++;

}

}

free(arr\_x);

free(arr\_y);

free(cdf);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (fp\_r != NULL) fclose(fp\_r);

if (fp\_w != NULL) fclose(fp\_w);

CHECK\_TIME\_END(resultTime);

printf("The program2\_2\_c run time is %f(ms)..\n", resultTime \* 1000.0);

}

뉴턴 방법은 앞서 구현한 secant와는 달리 초기값이 하나가 필요합니다. secant와 비슷하게 적절한 초기값 설정을 위해 bisection을 2번 동작시킨 뒤 이에 대응하는 start값으로 뉴턴 방법 계산의 초기값으로 설정하였습니다. 또한 뉴턴 방법에서는 함수의 미분 값이 필요한데, 이는 자료에 있는 선형 보간 방법을 따랐습니다.

실행 결과는 다음과 같습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

실행 결과에서 볼 수 있듯 뉴턴 방법은 시컨트와 바이섹션 방법보다 더 빠른 것을 확인할 수 있습니다.

초기값 설정에서 바이섹션 방법을 2번 돌리는 방식을 채택했는데, 바이섹션 과정에서 계속해서 계산이 되는 start와 end의 평균값 mid가 pdf에서 x값인 arr\_x에서 나타내어 지는 인덱스(arr\_x[??] < mid < arr\_x[?? + 1])를 idx 변수로 표현하였고 바이섹션 계산이 모두 끝나면 그 다음 뉴턴 랩슨을 진행하는데 이 때 초기값 a를 arr\_x[idx]로 설정하였습니다.

만약 초기값 설정에 있어서 바이섹션 방법을 쓰지 않고 단순하게 arr\_x의 중앙값을 쓰게 된다면 다음과 같은 결과를 얻게 됩니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

바이섹션을 2번 돌려 적절한 초기값을 설정하였기 때문에 앞선 경우가 훨씬 성능이 개선된 결과를 나타내는 것을 확인할 수 있습니다.

(program2\_3)

만들어진 random\_event\_table.txt, pdf\_table.txt를 이용해 histogram데이터를 구축하는 프로그램입니다. 이에 대한 코드는 다음과 같습니다.

void program2\_3() {

FILE\* fp\_r, \* fp\_r2;

FILE\* fp\_w;

fp\_r = fopen("pdf\_table.txt", "r");

fp\_r2 = fopen("random\_event\_table.txt", "r");

fp\_w = fopen("histogram.txt", "w");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double num, x;

fscanf(fp\_r, "%lf %lf", &num, &x);

double\* arr\_x = (double\*)malloc(sizeof(double) \* num);

double\* arr\_y = (double\*)malloc(sizeof(double) \* num);

int arrange[100];

for (int i = 0; i < 100; i++)

arrange[i] = 0;

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

fscanf(fp\_r, "%lf %lf", &arr\_x[i], &arr\_y[i]);

double sum = 0.0;

int num\_of\_nan = 0;

fscanf(fp\_r2, "%d", &num\_of\_nan);

double\* ran\_event = (double\*)malloc(sizeof(double) \* num\_of\_nan);

for (int i = 0; i < num\_of\_nan; i++)

fscanf(fp\_r2, "%lf", &ran\_event[i]);

//printf("sss\n");

for (int i = 0; i < num\_of\_nan; i++) {

int mok = ran\_event[i] \* 100;

//printf("%lf\n", ran\_event[i]);

arrange[mok]++;

}

for (int i = 0; i < 100; i++)

fprintf(fp\_w, "%d\n", arrange[i]);

free(arr\_x);

free(arr\_y);

free(ran\_event);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

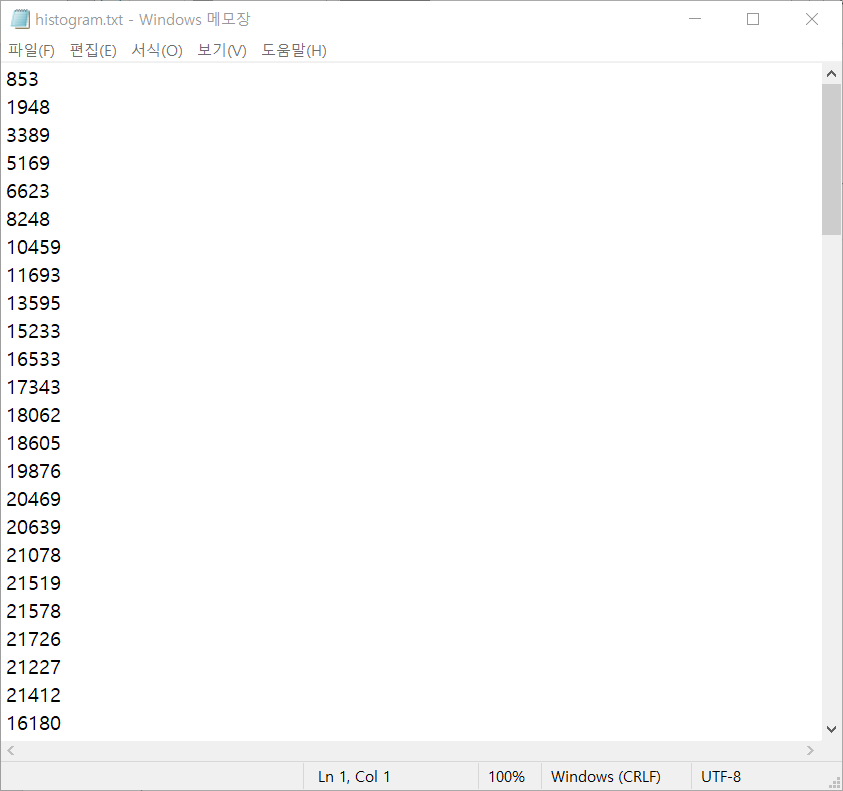
if (fp\_r != NULL) fclose(fp\_r);

if (fp\_r2 != NULL) fclose(fp\_r2);

if (fp\_w != NULL) fclose(fp\_w);

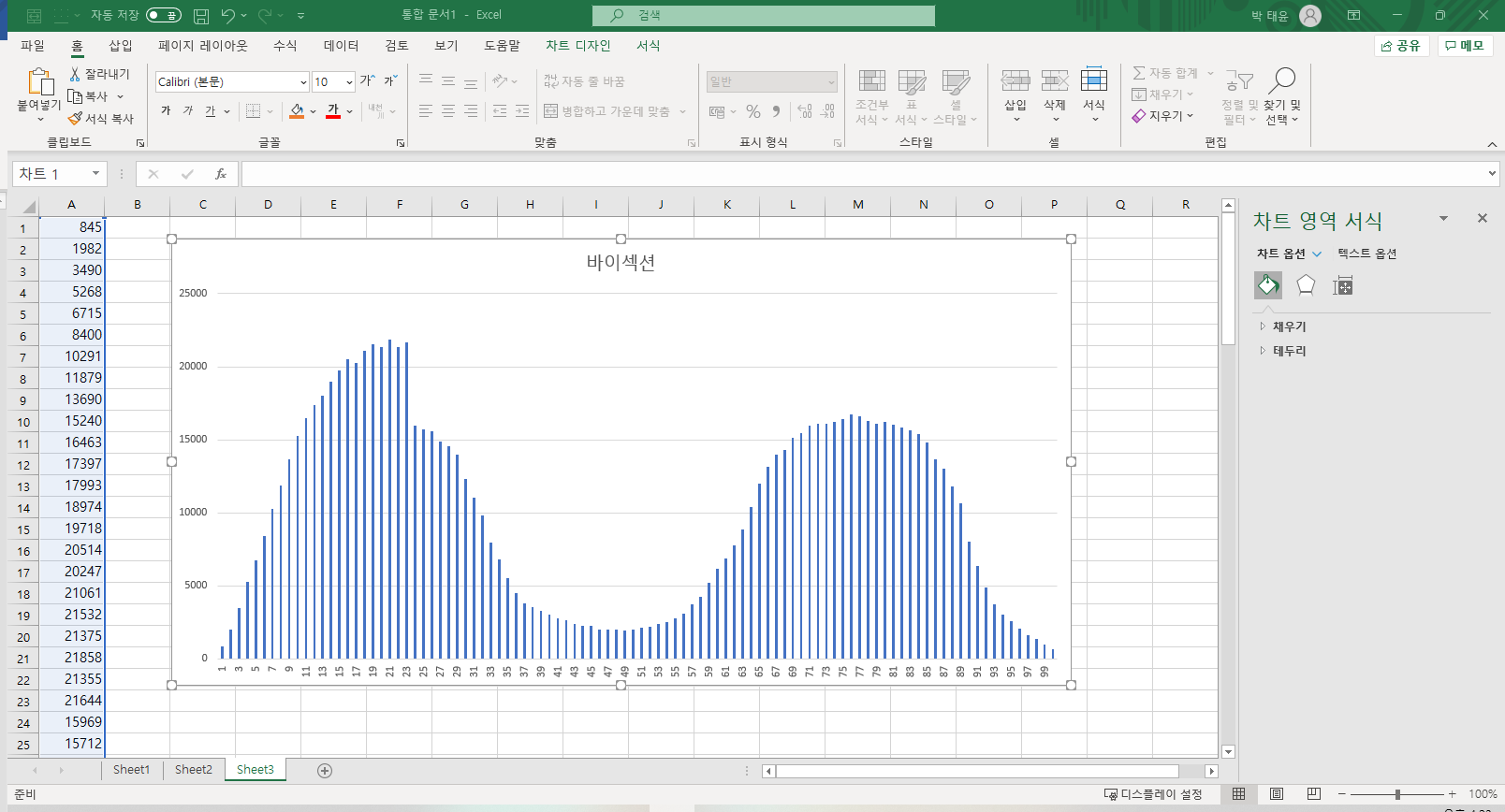
}

random\_event\_table.txt과 pdt\_table.txt에서 정보를 읽어온 다음 히스토그램 데이터를 만듭니다. arrange[100]이라는 배열을 선언해 이를 표현하였으며, random\_event\_table.txt의 값을 나타내는 ran\_event배열에 100을 곱해 integer변수인 mok에 저장을 한 다음 이를 arrange[mok]처럼 인덱스로 사용하여 하나씩 증가시키면서 범위별로 난수가 몇 번 나타났는지를 표현하였습니다. arrange배열의 사이즈가 100이기 때문에 구간의 간격은 0.01 즉, 0.00~0.01, 0.01~0.02 …. 0.99~1.00을 나타냅니다. 이 프로그램을 이용하여 histogram.txt를 만들면 다음과 같은 결과를 얻을 수 있습니다.

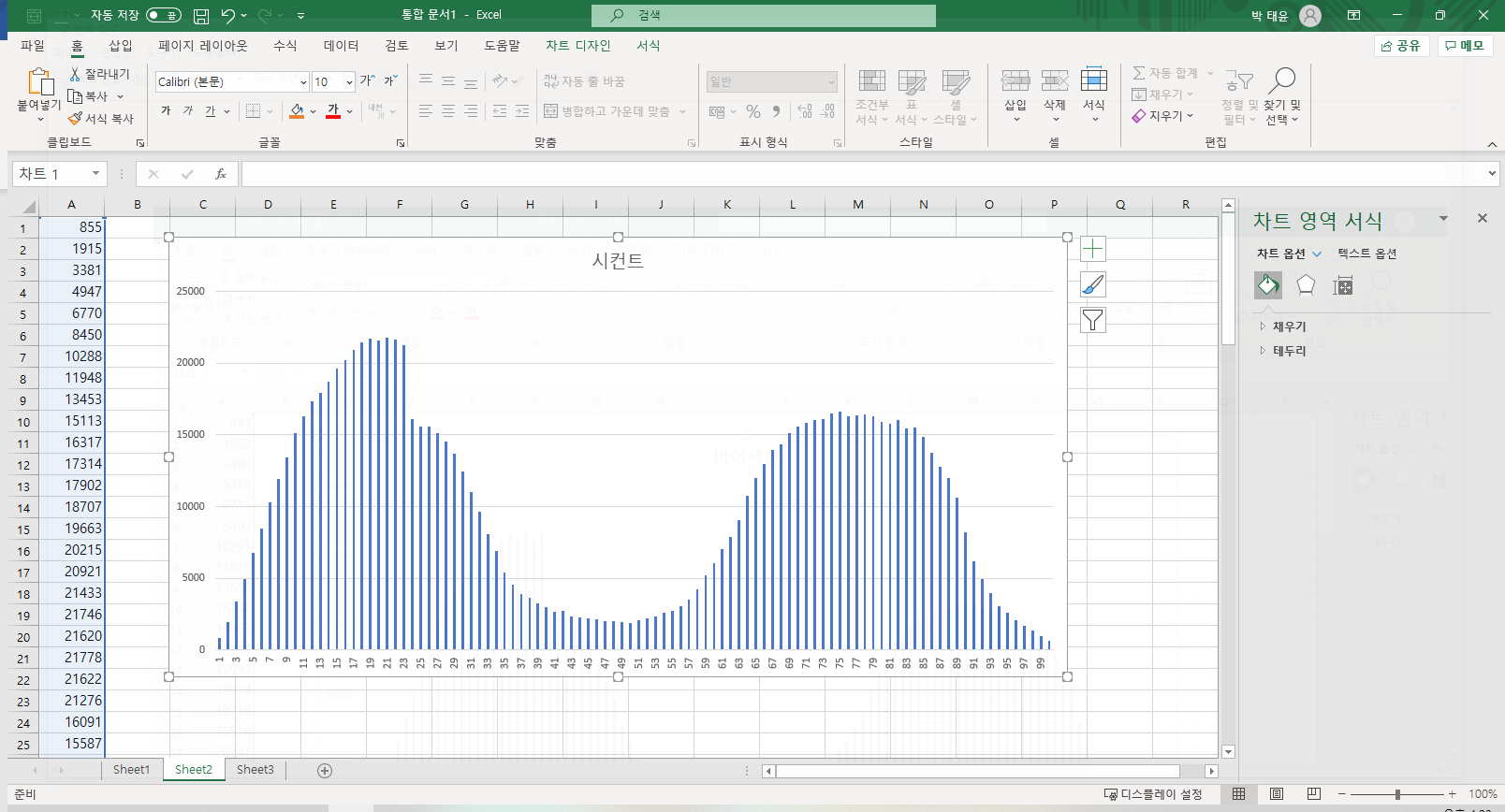


0.00 ~ 0.01, 0.01 ~ 0.02 …… 0.99 ~ 1.00에 해당하는 데이터의 분포 개수를 나타냅니다. 개수 카운트는 random\_event\_table.txt에서 읽은 값에 100을 곱한 결과를 인티져 변수 mok에 저장하여 100개 사이즈를 나타내는 배열 arrange[mok]를 arrange[mok]++하는 방식으로 구현하였습니다. 이 데이터를 가지고 엑셀 프로그램을 이용해 막대 그래프를 그리면 다음과 같은 결과를 얻을 수 있습니다. 3가지 프로그램(바이섹션, 시컨트, 뉴턴-랩슨)에 대해 엑셀로 히스토그램 그래프를 표현하였습니다.

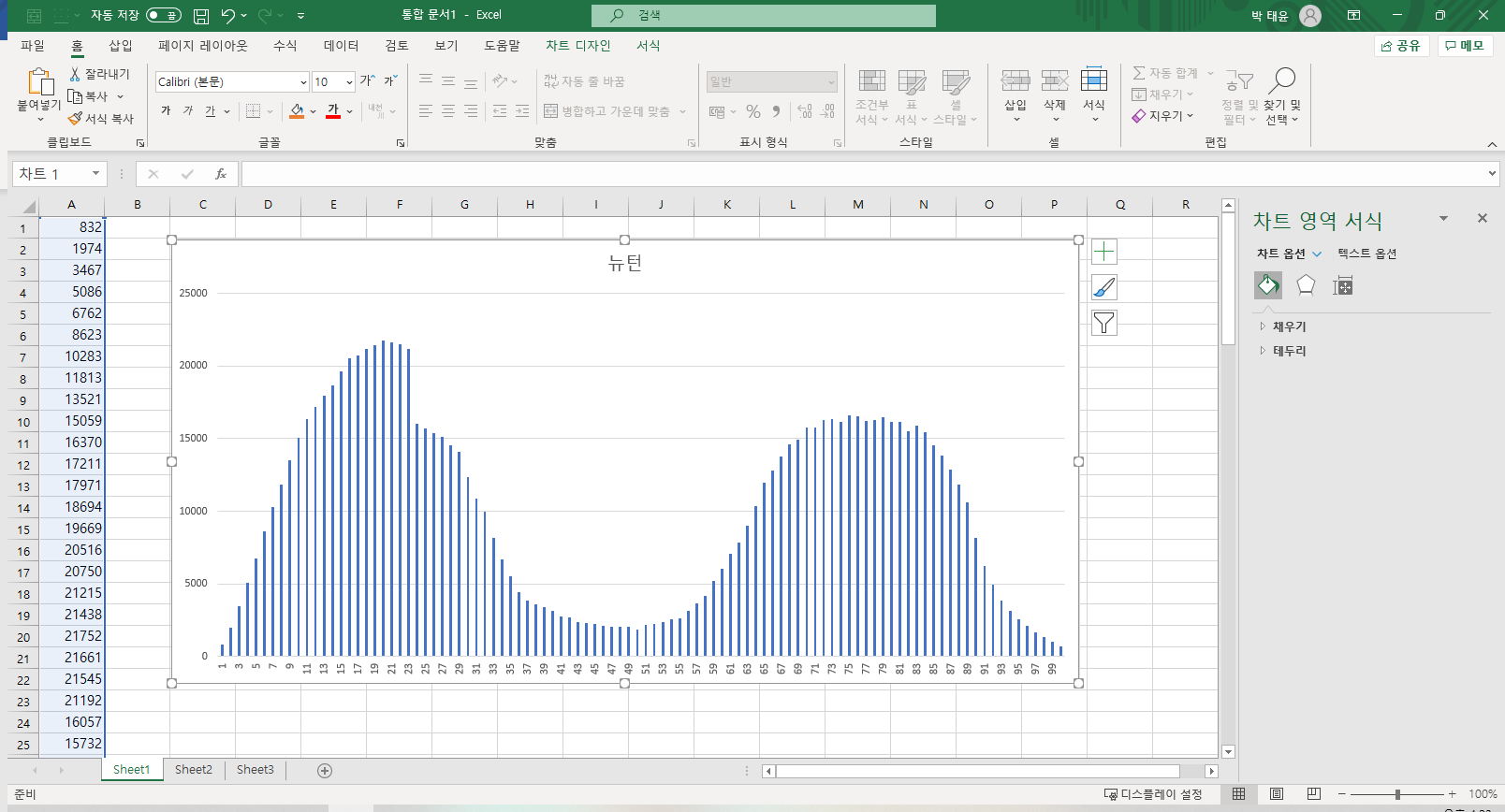
(바이섹션)



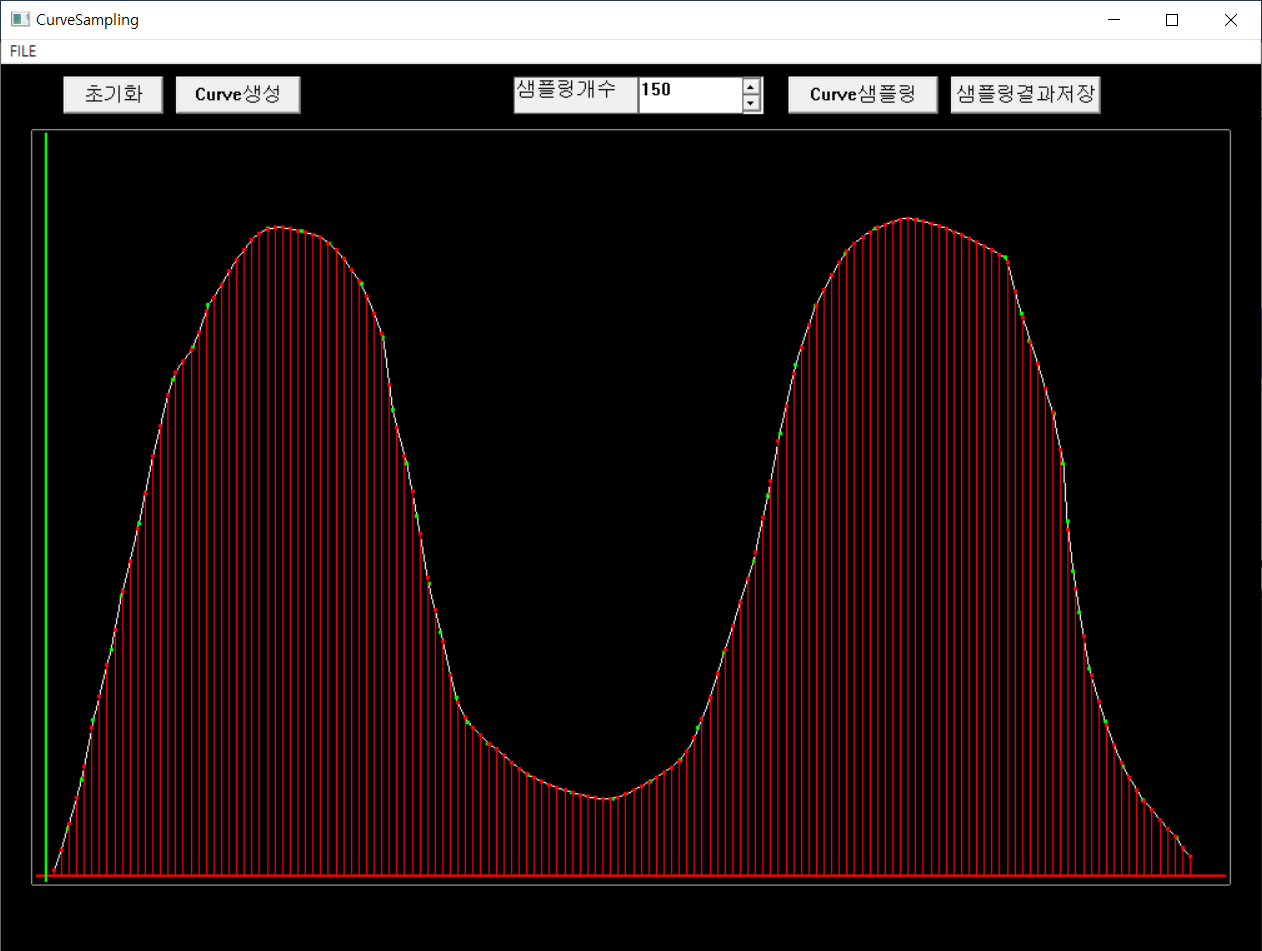
(시컨트)



(뉴턴-랩슨)



당시에 curveSampling프로그램을 이용하여 생성한 그래프는 다음과 같이 생겼습니다.



기존에 그렸던 그래프와, 난수에 대해 계산한 역함수 값을 구간별로 나누어 개수를 카운트하여 그린 막대그래프가 매우 유사한 모양을 나타냄을 확인할 수 있습니다. 프로그램을 이용하여 생성한 그래프가 큰 폭으로 증가하는 경우 누적 분포 함수 cdf또한 큰 폭으로 증가를 하기 때문에 난수를 생성하여 cdf에 대응하는 역함수 값을 계산하여 이에 대한 분포를 그래프로 표현했을 시에 프로그램을 이용하여 생성한 그래프와 유사한 모양을 나타내는 것을 확인할 수 있습니다.