**『GPS수신기위치계산문제』**

**수치 컴퓨팅 실험2: 특정확률 사건의 생성**

고급소프트웨어실습 3반

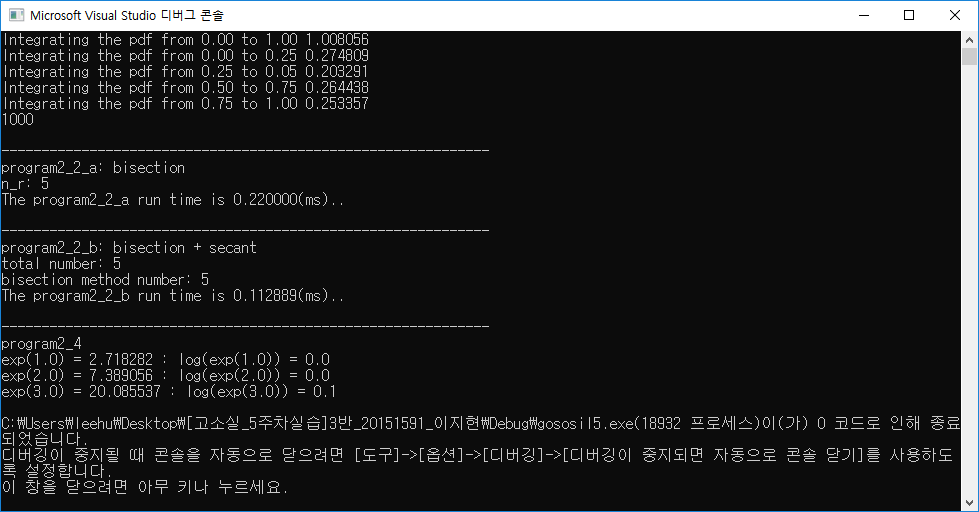
20151591 이지현

**숙제 2-1 i) 지수 분포의 인자 λ (> 0)에 대해 확률 변수 X의 기대값이 E[X] = 1/λ 이고 분산이 Var[X] = 1/λ^2이라는 사실을 고려하여, 세 가지 서로 다른 λ 값을 설정하라.**

**숙제 2-1 ii) 자신이 정한 λ에 해당하는 지수 분포 각각에 대하여, inversion 방법을 사용하여 자신이 정한 충분히 큰 개수만큼 이 지수 분포를 따르는 난수를 생성하라.**

**숙제 2-1 iii) 자신이 생성한 난수들을 사용하여 평균값과 분산값을 구한 후 이론적인 값들과 얼마나 일치하는지 분석하라.**

|  |
| --- |
| program2\_3.cpp에 내장된 program2\_4() function |
| void program2\_4()  {  printf("\n-------------------------------------------------------------\n");  printf("program2\_4\n");  double Lambda[3] = { 1, 2, 3 };  double U;  double X1, X2;  int n = 1000;  for (int i = 0; i < 3; i++) {  U = (double)rand() / RAND\_MAX; //uniform random numbers  X1 = -1 / Lambda[i] \* log(1 - U); //역변환법  X2 = exp(Lambda[i]);// R function rexp(n,lambda)  printf("exp(%.1f) = %f : log(exp(%.1f)) = %.1f\n", Lambda[i], X2, Lambda[i], X1);  }  } |



**숙제 2-2 i) 여러분이 실습시간에 작성한 코드 (프로그램 2-2)를 잘 가다듬어 문제없이 그리고 효율적으로 수행되도록 수정하라 (프로그램 2-2(a)).**

|  |
| --- |
| program2\_3.cpp |
| #include "my\_solver.h"  #include <time.h>  #include <math.h>  #include <stdlib.h>  // global variables  const double DELTA = 0.000000000001;  const int Nmax = 100;  const double EPSILON = 0.00000000001;  double \_F2(double X, double \*data\_x, double \*data\_y, int n) {  int i;  for (i = 0; i < n - 1; i++)  if (data\_x[i] <= X && X < data\_x[i + 1]) break;  double y\_base = data\_y[i];  double y\_diff = data\_y[i + 1] - data\_y[i];  double r\_base = data\_x[i + 1] - data\_x[i];  double r\_0 = (X - data\_x[i]);  double r\_1 = (data\_x[i + 1] - X);  return y\_base + y\_diff \* r\_0 / r\_base;  }  // HOMEWORK  void program2\_2\_a()  {  \_\_int64 start, freq, end;  float resultTime = 0;  FILE \*fp\_r, \*fp\_w;  int n\_r;  fp\_r = fopen("pdf\_table.txt", "r");  fp\_w = fopen("random\_event\_table\_2.txt", "w");  scanf("%d", &n\_r);  fprintf(fp\_w, "%d\n", n\_r);  srand(time(NULL));  double \*rand\_U = new double[n\_r];  for (int i = 0; i < n\_r; i++)  rand\_U[i] = rand() / RAND\_MAX;  int n;  double fre;  fscanf(fp\_r, "%d %lf", &n, &fre);  double \*x = new double[n];  double \*y\_tmp = new double[n];  double \*y = new double[n];  double integral\_result = 0;  for (int i = 0; i<n; i++){  fscanf(fp\_r, "%lf %lf", &x[i], &y\_tmp[i]);  }  double fre\_half = fre / 2.0;  y[0] = 0;  y[1] = fre\_half \* (y\_tmp[0] + y\_tmp[1]);  for (int i = 2; i < n; i++){  y[i] = y[i - 1] + fre\_half \* (y\_tmp[i - 1] + y\_tmp[i]);  }    CHECK\_TIME\_START;  // something to do...  int i = 0;  while (n\_r--) {  double U = rand\_U[i++];  double x0 = INT\_MAX;  double a0 = 0, b0 = 1, temp;  double x1 = (a0 + b0) / 2;  for (;;) {  if (fabs(\_F2(x1, x, y, 100) - U) < DELTA)  break;  if (fabs(x1 - x0) < EPSILON)  break;  double value\_a = \_F2(a0, x, y, 100) - U;  double value\_b = \_F2(b0, x, y, 100) - U;  temp = (a0 + b0) / 2.0;  double value\_ab = \_F2(temp, x, y, 100) - U;  if (value\_a \* value\_ab > 0)  a0 = temp;  else if (value\_b \* value\_ab > 0)  b0 = temp;  else {  if (fabs(value\_a) < fabs(value\_b))  b0 = temp;  else  a0 = temp;  }  x0 = x1;  x1 = (a0 + b0) / 2;  }  fprintf(fp\_w, "%.15lf / %.15lf\n", U, x1);  }  CHECK\_TIME\_END(resultTime);  delete[] x;  delete[] y\_tmp;  delete[] y;  delete[] rand\_U;  printf("The program2\_2\_a run time is %f(ms)..\n", resultTime\*1000.0);  } |

**숙제 2-2(ii) 주어진 확률밀도함수에 대하여 자신의 코드가 올바르게 난수를 생성하는지 통계적으로 확인하라. 이를 위하여 아래에서 설명하는 내용의 코드(프로그램 2-3)을 작성한 후 이를 활용하라.**

|  |
| --- |
| program2\_3.cpp |
| void program2\_3()  {  FILE \*fp\_r\_pdf, \*fp\_r\_rand, \*fp\_w;  fp\_r\_pdf = fopen("pdf\_table.txt", "r");  fp\_r\_rand = fopen("random\_event\_table\_2\_a.txt", "r");  fp\_w = fopen("histogram.txt", "w");  int n, N;  double h;  fscanf(fp\_r\_pdf, "%d %lf", &n, &h);  fscanf(fp\_r\_rand, "%d", &N);  double \*x = new double[n];  double \*fx = new double[n];  double \*rand = new double[N];  int \*histo = new int[n];  for (int i = 0; i < n; i++) {  fscanf(fp\_r\_pdf, "%lf %lf", &x[i], &fx[i]);  histo[i] = 0;  }  for (int i = 0; i < N; i++) {  fscanf(fp\_r\_rand, "%lf", &rand[i]);  for (int k = 0; k < n; k++) {  if (x[k] <= rand[i] && rand[i] < x[k + 1]) {  histo[k]++;  break;  }  }  }  for (int i = 0; i < n - 1; i++) {  fprintf(fp\_w, "%lf ~ %lf : %d\n", x[i], x[i + 1], histo[i]);  }  delete[] x;  delete[] fx;  delete[] rand;  delete[] histo;  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0.000000 ~ 0.010101 | 1 | 0.333333 ~ 0.343434 | 3 | 0.666667 ~ 0.676768 | 2 |
| 0.010101 ~ 0.020202 | 1 | 0.343434 ~ 0.353535 | 1 | 0.676768 ~ 0.686869 | 0 |
| 0.020202 ~ 0.030303 | 1 | 0.353535 ~ 0.363636 | 2 | 0.686869 ~ 0.696970 | 1 |
| 0.030303 ~ 0.040404 | 0 | 0.363636 ~ 0.373737 | 1 | 0.696970 ~ 0.707071 | 0 |
| 0.040404 ~ 0.050505 | 0 | 0.373737 ~ 0.383838 | 3 | 0.707071 ~ 0.717172 | 1 |
| 0.050505 ~ 0.060606 | 0 | 0.383838 ~ 0.393939 | 1 | 0.717172 ~ 0.727273 | 1 |
| 0.060606 ~ 0.070707 | 1 | 0.393939 ~ 0.404040 | 0 | 0.727273 ~ 0.737374 | 1 |
| 0.070707 ~ 0.080808 | 0 | 0.404040 ~ 0.414141 | 1 | 0.737374 ~ 0.747475 | 0 |
| 0.080808 ~ 0.090909 | 0 | 0.414141 ~ 0.424242 | 0 | 0.747475 ~ 0.757576 | 1 |
| 0.090909 ~ 0.101010 | 0 | 0.424242 ~ 0.434343 | 2 | 0.757576 ~ 0.767677 | 2 |
| 0.101010 ~ 0.111111 | 1 | 0.434343 ~ 0.444444 | 6 | 0.767677 ~ 0.777778 | 0 |
| 0.111111 ~ 0.121212 | 1 | 0.444444 ~ 0.454545 | 1 | 0.777778 ~ 0.787879 | 2 |
| 0.121212 ~ 0.131313 | 1 | 0.454545 ~ 0.464646 | 3 | 0.787879 ~ 0.797980 | 0 |
| 0.131313 ~ 0.141414 | 1 | 0.464646 ~ 0.474748 | 1 | 0.797980 ~ 0.808081 | 0 |
| 0.141414 ~ 0.151515 | 0 | 0.474748 ~ 0.484849 | 1 | 0.808081 ~ 0.818182 | 0 |
| 0.151515 ~ 0.161616 | 0 | 0.484849 ~ 0.494950 | 1 | 0.818182 ~ 0.828283 | 0 |
| 0.161616 ~ 0.171717 | 1 | 0.494950 ~ 0.505051 | 1 | 0.828283 ~ 0.838384 | 1 |
| 0.171717 ~ 0.181818 | 0 | 0.505051 ~ 0.515152 | 5 | 0.838384 ~ 0.848485 | 0 |
| 0.181818 ~ 0.191919 | 0 | 0.515152 ~ 0.525253 | 1 | 0.848485 ~ 0.858586 | 1 |
| 0.191919 ~ 0.202020 | 0 | 0.525253 ~ 0.535354 | 1 | 0.858586 ~ 0.868687 | 0 |
| 0.202020 ~ 0.212121 | 0 | 0.535354 ~ 0.545455 | 2 | 0.868687 ~ 0.878788 | 0 |
| 0.212121 ~ 0.222222 | 1 | 0.545455 ~ 0.555556 | 2 | 0.878788 ~ 0.888889 | 1 |
| 0.222222 ~ 0.232323 | 1 | 0.555556 ~ 0.565657 | 1 | 0.888889 ~ 0.898990 | 1 |
| 0.232323 ~ 0.242424 | 1 | 0.565657 ~ 0.575758 | 2 | 0.898990 ~ 0.909091 | 0 |
| 0.242424 ~ 0.252525 | 1 | 0.575758 ~ 0.585859 | 3 | 0.909091 ~ 0.919192 | 0 |
| 0.252525 ~ 0.262626 | 0 | 0.585859 ~ 0.595960 | 3 | 0.919192 ~ 0.929293 | 0 |
| 0.262626 ~ 0.272727 | 1 | 0.595960 ~ 0.606061 | 4 | 0.929293 ~ 0.939394 | 0 |
| 0.272727 ~ 0.282828 | 2 | 0.606061 ~ 0.616162 | 1 | 0.939394 ~ 0.949495 | 1 |
| 0.282828 ~ 0.292929 | 0 | 0.616162 ~ 0.626263 | 1 | 0.949495 ~ 0.959596 | 1 |
| 0.292929 ~ 0.303030 | 2 | 0.626263 ~ 0.636364 | 2 | 0.959596 ~ 0.969697 | 0 |
| 0.303030 ~ 0.313131 | 1 | 0.636364 ~ 0.646465 | 2 | 0.969697 ~ 0.979798 | 0 |
| 0.313131 ~ 0.323232 | 1 | 0.646465 ~ 0.656566 | 3 | 0.979798 ~ 0.989899 | 1 |
| 0.323232 ~ 0.333333 | 1 | 0.656566 ~ 0.666667 | 0 | 0.989899 ~ 1.000000 | 0 |

분포를 확인하기 위해 히스토그램을 뽑아오고, 그에 대한 값을 그래프로 그려보았다. 처음 확률 밀도 함수를 뽑기 위해 사용하였던 프로그램에 그린 함수의 모양과 유사한 밀도가 나왔으며, 너무 한 부분에만 몰리지 않고 적당하게 분포되어 있음을 알 수 있다. 따라서 코드가 올바르게 난수를 생성하고 있음을 알 수 있다.

**숙제 2-2(iii) 비선형 방정식의 근을 구하는 방법 구현 시 앞에서 사용한 Bisection 방법을 Secant 방법으로 대치한 난수 생성 프로그램을 작성하라 (프로그램 2-2(b)). 물론 자신의 프로그램이 제대로 작동하는지 실험적으로 확인하라.**

|  |
| --- |
| program2\_3.cpp |
| double \_F2\_(double X, double \*data\_x, double \*data\_y, int n) {  double r1, r2;  double y;  for (int i = 0; i < n - 1; i++) {  if (data\_x[i] <= X && X <= data\_x[i + 1]) {  idx = i;  break;  }  }  r1 = X - data\_x[idx];  r2 = data\_x[idx + 1] - X;  y = (r2 \* data\_y[idx] + r1 \* data\_y[idx + 1]) / (r1 + r2);  return y;  }  double \_F2P\_(double X, double \*data\_x, double \*data\_y, int n) {  double s;  double r1, r2;  double y;  for (int i = 0; i < n - 1; i++) {  if (data\_x[i] <= X && X <= data\_x[i + 1]) {  idx = i;  break;  }  }  s = (X - data\_x[idx]) / (data\_x[idx + 1] - data\_x[idx]);  return (1 - s) \* data\_y[idx] + s \* data\_y[idx + 1];  }  void program2\_2\_b()  {  printf("\n-------------------------------------------------------------\n");  printf("program2\_2\_b: bisection + secant\n");  \_\_int64 start, freq, end;  float resultTime = 0;  double U;  FILE \*fp\_r, \*fp\_w;  int n\_r, bi\_n;  fp\_r = fopen("pdf\_table.txt", "r");  fp\_w = fopen("random\_event\_table\_2\_b.txt", "w");  printf("total number: ");  scanf("%d", &n\_r);  printf("bisection method number: ");  scanf("%d", &bi\_n);  //fprintf(fp\_w, "%d / %d\n", n\_r, bi\_n);  fprintf(fp\_w, "%d\n", n\_r);  srand(time(NULL));  int n;  double fre;  fscanf(fp\_r, "%d %lf", &n, &fre);  double \*x = new double[n];  double \*y\_tmp = new double[n];  double \*y = new double[n];  for (int i = 0; i<n; i++){  fscanf(fp\_r, "%lf %lf", &x[i], &y\_tmp[i]);  }  double fre\_half = fre / 2.0;  y[0] = 0;  y[1] = fre\_half \* (y\_tmp[0] + y\_tmp[1]);  for (int i = 2; i < n; i++){  y[i] = y[i - 1] + fre\_half \* (y\_tmp[i - 1] + y\_tmp[i]);  }  CHECK\_TIME\_START;  // something to do...  int i = 0;  while (n\_r--) {  U = (double)rand() / RAND\_MAX;  double x0 = INT\_MAX;  double a0 = 0, b0 = 1, temp;  double x1 = (a0 + b0) / 2;  for (int j = 0; j < bi\_n; j++) {  if (fabs(\_F2(x1, x, y, 100) - U) < DELTA)  break;  if (fabs(x1 - x0) < EPSILON)  break;  double value\_a = \_F2(a0, x, y, 100) - U;  double value\_b = \_F2(b0, x, y, 100) - U;  temp = (a0 + b0) / 2.0;  double value\_ab = \_F2(temp, x, y, 100) - U;  if (value\_a \* value\_ab > 0)  a0 = temp;  else if (value\_b \* value\_ab > 0)  b0 = temp;  else {  if (fabs(value\_a) < fabs(value\_b))  b0 = temp;  else  a0 = temp;  }  x0 = x1;  x1 = (a0 + b0) / 2;  }  x0 = x1;  x1 = x0;  for (int k = 0; k < 50; k++) {  if (fabs(\_F2(x1, x, y, 100) - U) < DELTA)  break;  if (fabs(x1 - x0) < EPSILON)  break;  x0 = x1;  x1 = x0 - (\_F2\_(x1, x, y, 100) - U) / \_F2P\_(x1, x, y, 100);  if (x1 > 1)  x1 = 1;  if (x1 < 0)  x1 = 0;  }  //fprintf(fp\_w, "%.15lf / %.15lf\n", U, x1);  fprintf(fp\_w, "%.15lf\n", x1);  }  CHECK\_TIME\_END(resultTime);  if (fp\_r != NULL) fclose(fp\_r);  if (fp\_w != NULL) fclose(fp\_w);  printf("The program2\_2\_b run time is %f(ms)..\n", resultTime\*1000.0); |

**숙제 2-2(iv) 충분히 큰 난수의 개수 *nr*에 대해 프로그램 2-2(a)와 프로그램 2-2(b)를 수행시킨 후, 각 방법이 난수를 생성하는데 걸린 시간을 비교하라.**

Newton 방식을 할 때는 초기값을 적절하게 구해주는 것이 중요한데, 이를 Bisection을 통해 구해주면 편하고 비교적 정확하게 구할 수 있다. 다만 우리는 처음 Bisection을 몇 회 실행할 것인지에 대해 의논을 해봐야 할 텐데 이는 실험값을 통해 구하도록 한다. 우리는 난수를 100개 생성할 때, 5의 배수를 단위로 0, 5, 10, 15, 20, 25, 50, 100번의 Bisection을 실행해보도록 한다. 그 결과는 다음과 같다.

|  |  |
| --- | --- |
|  | 시간 |
| 5 | 0.600421 |
| 10 | 0.898971 |
| 15 | 1.252462 |
| 20 | 1.546786 |
| 25 | 1.880052 |
| 50 | 2.602703 |
| 100 | 2.660388 |

우리는 실험 결과에 따라 bisection은 5번을 실행하고 Newton 방식을 통해 난수를 구하도록 하자. 그 때, bisection과 Newton 방식을 비교하도록 하자. 데이터의 개수는 100개를 사용하도록 한다. 그 때 3번의 반복된 실험의 결과를 비교하도록 하자.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bisection | |  | Newton | |
|  | 시간 |  | 시간 |
| 1회 | 2.772985 | 1회 | 0.595591 |
| 2회 | 2.724082 | 2회 | 0.614005 |
| 3회 | 2.653445 | 3회 | 0.600421 |
| 평균 | 2.716837 | 평균 | 0.603339 |

결과를 따져보면 두 값은 모두 높은 정확도를 보인다. 다만 걸린 시간의 경우 Newton 방식이 Bisection 방식에 비해 평균 4.5배 빠름을 알 수 있다. 따라서 동일한 방식을 사용할 수 있다면, Newton 방식을 사용하면 좋고, 또한 그 Newton 방식에 초기값 설정은 Bisection 방식으로 잡아준다면 제일 좋은 성능을 발휘할 수 있을 것이다.