2024 年ソフトウェア演習2B

第４　回課題

B243392　　ALCANDER IMAWAN　　2024 年 7 月 08 日

Q1

プログラム

main.cpp

```
1: #include <iostream>
2: #include <random> //to use std::mt19937
3: #include <cmath> //for std::pow and std::sqrt
4: #include <vector>
5: #include <cstdlib> //for std::atof and std::atoi
6: #include <fstream>
7: #include <iomanip>
8:
9: double calculateMean (const std::vector<double>& arr, int n){
10:         double sum = 0.0;
11:         for(int i= 0; i < n; i++){
12:                 sum += arr[i];
13:         }
14:         return sum / n;
15: }
16:
17: double calculateVariance(const std::vector<double>& arr, int n){
18:         double sum = 0.0;
19:         double mean = calculateMean(arr, n);
20:
21:         for (int i= 0; i < n; i++){
```

```
22:             sum += std::pow(arr[i] - mean, 2);
23:         }
24:         return sum / n;
25: }
26:
27: int main (int argc, char *argv[]){
28:         double meanDifferenceSquared, varianceDifferenceSquared;
29:         if(argc != 5){
30:             std::cerr << "Usage: " << argv[0] << " <seed> <number of data> <mean>
<variance>" << std::endl;
31:             return 1;
32:         }
33:
34:         //parse input
35:         std::uint32_t seed = std::atoi(argv[1]);
36:         int numData = std::atoi(argv[2]);
37:         double mean = std::atof(argv[3]);
38:         double variance = std::atof(argv[4]);
39:
40:         double stdev = std::sqrt(variance);
41:
42:         std::cout << "Calculating sample mean and sample variance" <<std::endl;
43:
44:         std::vector<double> arr;
45:         //
46:         std::mt19937 mt(seed); //instantiate 32-bit Mersenne Twister with a seed from
std::random_device to ensure different results every run
47:
48:         //create a reusable random number generator that generates uniform numbers
between -100 and 100
49:         std::normal_distribution<double> randoms{mean, stdev};
50:         //put random numbers in arr
51:         for(int count = 0; count < numData; ++count){
```

```
52:             arr.push_back(randoms(mt));
53:         }
54:
55:         std::ofstream outfile("plot_data.txt");
56:         outfile << "N¥tMeanDifferenceSquared¥tVarianceDifferenceSquared¥n";
57:
58:         // Calculate squared differences
59:         for(int i = 100; i <= numData; i += 100){
60:             meanDifferenceSquared = std::pow(calculateMean(arr, i) - mean, 2);
61:             varianceDifferenceSquared = std::pow(calculateVariance(arr, i) - variance, 2);
62:
63:             outfile << i << "¥t" << meanDifferenceSquared << "¥t" <<
varianceDifferenceSquared << "¥n";
64:         }
65:
66:         outfile.close();
67:         std::cout << "Data saved to plot_data.txt" << std::endl;
68:         return 0;
69: }
```

解説

指定された乱数シード、データ数、平均、および分散を使用して正規分布に従うデータセットを生成し、そのデータセットのサンプル平均とサンプル分散を計算する。また、指定された平均および分散との平方差を計算し、結果をファイルに保存する。

まず、ヘッダファイルを行くルードする。ヘッダファイルの説明はコメントアウトにした。次は calculateMean 関数。配列内のデータの平均値を計算し、ループを使用して全ての要素の合計を求め、その合計をデータ数で割る。もう一つの関数は calculateVariance 関数である。配列内のデータの分散を計算する。まず平均値を計算し、その後各データポイントと平均の差の二乗の合計を求め、それをデータ数で割る。

Main 関数は最初に、コマンドライン引数が正しい形式で提供されているかを確認し、シード、データ数、平均、および分散をパースする。また、Mersenne Twister エンジンを使用して正規分布に従う乱数を生成します。指定されたシードとパラメータに基づいて乱数を生成する。そして、指定されたデータ数だけの乱数を生成し、ベクターに格納する。最後に、

plot_data_exponent.txt というファイルにサンプル数に対する平均および分散の平方差を出力する。ループを用いて、データの一部集合ごとにこれらの値を計算し、ファイルに書き込む。

グラフは excel を用いて作成した。

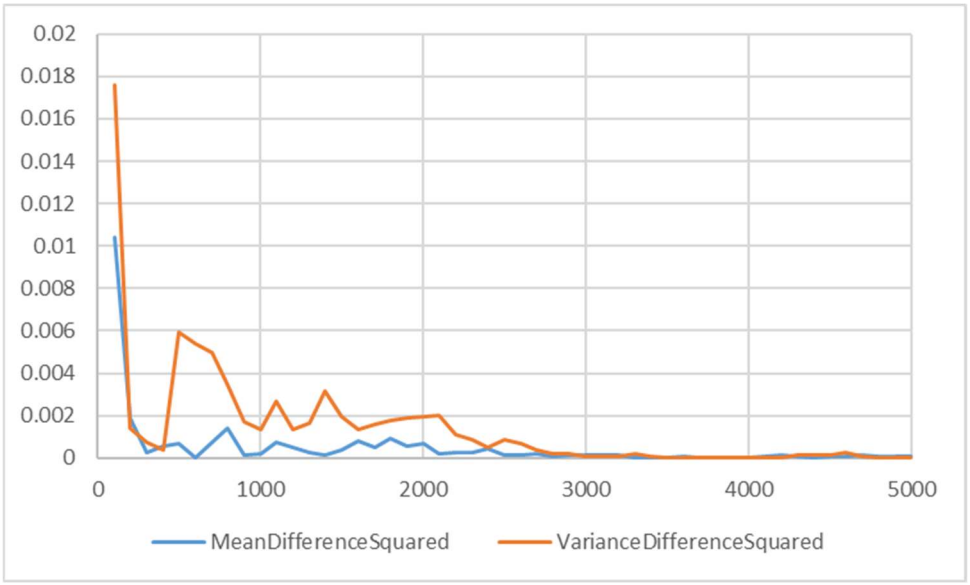
動作確認

Data saved to plot_data.txt

Plot_data.txt

N	MeanDifferenceSquared	VarianceDifferenceSquared
100	0.0103786	0.0175694
200	0.00187758	0.00142483
300	0.000270825	0.000769888
400	0.000538777	0.000384614
500	0.000665555	0.00591506
600	2.42007e-05	0.00537752
700	0.000742843	0.00497827
800	0.00142535	0.00344825
900	0.000155548	0.0017347
1000	0.000196631	0.00137239
1100	0.000737603	0.00269971
1200	0.000489377	0.00133032
1300	0.000264515	0.00165191
1400	0.000158869	0.00315386
1500	0.000349483	0.001978
1600	0.000794502	0.00131933
1700	0.000516864	0.00160336
1800	0.000912338	0.0017477
1900	0.000584689	0.00189066
2000	0.000660701	0.00194345
2100	0.000174878	0.00202137
2200	0.00026961	0.00112756
2300	0.000251723	0.000875984
2400	0.000418808	0.000487835
2500	0.000123877	0.000855268
2600	0.000158729	0.000673903

2700	0.00016915	0.000378366
2800	0.000103726	0.000206894
2900	0.000153929	0.000202593
3000	0.000123642	6.92101e-05
3100	0.000127404	8.05281e-05
3200	0.000154629	5.74922e-05
3300	1.1365e-05	0.000180523
3400	1.61586e-05	8.29125e-05
3500	2.83436e-05	2.35843e-05
3600	6.05951e-05	4.10438e-05
3700	1.6171e-05	4.78518e-06
3800	9.06973e-07	2.08673e-05
3900	4.54796e-06	1.99806e-05
4000	3.49748e-05	8.03822e-06
4100	6.9424e-05	1.92141e-06
4200	0.000117525	3.57647e-05
4300	6.30825e-05	0.000110551
4400	2.96631e-05	0.000141169
4500	6.78097e-05	0.000162772
4600	6.52549e-05	0.000273645
4700	0.000112284	8.87016e-05
4800	0.000100868	2.72363e-05
4900	8.54177e-05	2.98613e-06
5000	6.66216e-05	6.9164e-06


0.02
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
1000
2000
3000
4000
5000
MeanDifferenceSquared
VarianceDifferenceSquared

Q2

プログラム

Q2.cpp

```
1: #include <iostream>
2: #include <Eigen/Dense>
3:
4: int main() {
5:         // Define vectors a and b
6:         Eigen::VectorXd a(3);
7:         Eigen::VectorXd b(3);
8:
9:         a << 1, 2, 3;
10:          b << 4, 5, 6;
11:
12:          // Define matrices A and B
13:          Eigen::MatrixXd A(3, 3);
14:          Eigen::MatrixXd B(3, 3);
15:
16:          A << 1, 2, 3,
17:                   4, 5, 6,
18:                   7, 8, 9;
19:
20:          B << 9, 8, 7,
21:                   6, 5, 4,
22:                   3, 2, 1;
23:
24:          // Output the vectors and matrices
25:          std::cout << "Vector a:¥n" << a << "¥n¥n";
26:          std::cout << "Vector b:¥n" << b << "¥n¥n";
27:          std::cout << "Matrix A:¥n" << A << "¥n¥n";
28:          std::cout << "Matrix B:¥n" << B << "¥n¥n";
```

```
29:
30:     // Vector addition: a + b
31:     Eigen::VectorXd vector_sum = a + b;
32:     std::cout << "a + b =¥n" << vector_sum << "¥n¥n";
33:
34:     // Outer product of vectors: a * b^T
35:     Eigen::MatrixXd outer_product = a * b.transpose();
36:     std::cout << "a * b^T =¥n" << outer_product << "¥n¥n";
37:
38:     // Dot product of vectors: a^T * b
39:     double dot_product = a.transpose() * b;
40:     std::cout << "a^T * b = " << dot_product << "¥n¥n";
41:
42:     // Matrix-vector multiplication: A * b
43:     Eigen::VectorXd matrix_vector_product = A * b;
44:     std::cout << "A * b =¥n" << matrix_vector_product << "¥n¥n";
45:
46:     // Matrix-matrix multiplication: A * B
47:     Eigen::MatrixXd matrix_matrix_product = A * B;
48:     std::cout << "A * B =¥n" << matrix_matrix_product << "¥n¥n";
49:
50:     return 0;
51: }
52:
```

解説

Eigen ライブラリを使用してベクトルおよび行列の基本的な演算を行う。具体的には、ベクトルと行列の定義と初期化、ベクトルの和、ベクトルの外積、内積、行列とベクトルの積、および行列同士の積を計算し、その結果を出力する。

まず、ベクトル a, b と行列 A, B 作成した。確認するため、ベクトル a、b および行列 A、B をコンソールに出力する。次に、順番に計算が行った。ベクトル a と b の和、ベクトル a と b の外積（a * b^T）、ベクトル a と b の内積（a^T * b）、行列 A とベクトル b の積、行列 A と B の積である。各計算を行い、その後結果を出力する。

動作確認

```
Vector a:
1
2
3

Vector b:
4
5
6

Matrix A:
1 2 3
4 5 6
7 8 9

Matrix B:
9 8 7
6 5 4
3 2 1

a + b =
5
7
9

a * b^T =
 4  5  6
 8 10 12
12 15 18

a^T * b = 32

A * b =
 32
 77
122

A * B =
 30  24  18
 84  69  54
138 114  90
```

Q3

プログラム

Q3.cpp

```
1: #include <iostream>
2: #include <random> // to use std::mt19937
3: #include <cmath> // for std::pow and std::sqrt
4: #include <vector>
5: #include <cstdlib> // for std::atof and std::atoi
6: #include <fstream>
7: #include <iomanip>
8:
9: // Function to calculate the variance of a vector
10: double calculateVariance(const std::vector<double>& arr) {
11:         double sum = 0.0;
12:         double mean = 0.0;
13:         for (const double& value : arr) {
14:                 sum += value;
15:         }
16:         mean = sum / arr.size();
17:
18:         double variance = 0.0;
19:         for (const double& value : arr) {
20:                 variance += std::pow(value - mean, 2);
21:         }
22:         return variance / arr.size();
23: }
24:
25: int main(int argc, char* argv[]) {
26:         if (argc != 3) {
```

```
27:             std::cerr << "Usage: " << argv[0] << " <seed> <number of data>" << std::endl;
28:             return 1;
29:         }
30:
31:         // Parse input
32:         std::uint32_t seed = std::atoi(argv[1]);
33:         int numData = std::atoi(argv[2]);
34:         double mean = 0.0;
35:         double variance = 1.0;
36:
37:         double stdev = std::sqrt(variance);
38:
39:         // Vector to store x and y coordinates
40:         std::vector<double> x_coords(numData);
41:         std::vector<double> y_coords(numData);
42:
43:         // Generate points on the line 3x - 2y + 4 = 0
44:         for (int alpha = 0; alpha < numData; ++alpha) {
45:             double x_alpha = -10 + (20.0 / (numData - 1)) * alpha;
46:             double y_alpha = (3 * x_alpha + 4) / 2;
47:             x_coords[alpha] = x_alpha;
48:             y_coords[alpha] = y_alpha;
49:         }
50:
51:         // Initialize random number generator
52:         std::mt19937 mt(seed);
53:         std::normal_distribution<double> noise_dist(mean, stdev);
54:
55:         // Add noise to the x and y coordinates
56:         std::vector<double> noisy_x_coords(numData);
57:         std::vector<double> noisy_y_coords(numData);
58:         std::vector<double> noise(numData);
59:
```

```
60:     for (int i = 0; i < numData; ++i) {
61:         double noise_val = noise_dist(mt);
62:         noisy_x_coords[i] = x_coords[i] + noise_val;
63:         noisy_y_coords[i] = y_coords[i] + noise_val;
64:         noise[i] = noise_val;
65:     }
66:
67:     // Calculate the variance of noisy x and y coordinates
68:     double variance_noisy_x = std::abs(calculateVariance(x_coords) -
calculateVariance(noisy_x_coords));
69:     double variance_noisy_y = std::abs(calculateVariance(y_coords) -
calculateVariance(noisy_y_coords));
70:
71:     std::cout << "Variance of noisy x coordinates: " << variance_noisy_x << std::endl;
72:     std::cout << "Variance of noisy y coordinates: " << variance_noisy_y << std::endl;
73:     std::cout << "Variance of noise: " << calculateVariance(noise) << std::endl;
74:
75:     // Output the data to a file
76:     std::ofstream outfile("noisy_data.txt");
77:     outfile << "x¥ty¥tx_noise¥tynoise¥n";
78:     for (int i = 0; i < numData; ++i) {
79:         outfile << x_coords[i] << "¥t" << y_coords[i] << "¥t" << noisy_x_coords[i] <<
"¥t" << noisy_y_coords[i] << "¥n";
80:     }
81:     outfile.close();
82:
83:     std::cout << "Data saved to noisy_data.txt" << std::endl;
84:
85:     return 0;
86: }
```

解説

指定されたシードとデータ数に基づいてノイズ付きのデータポイントを生成し、直線 3x−2y+4=0 に沿った点を生成する。その後、ノイズを追加してデータポイントを作成し、結果をファイルに保存する。

ヘッダファイルとコマンドライン引数をチェックし、ノイズ生成のためのパラメータを定義する。簡単にするため、平均が 0 にし、variance（分布）を 1 にする。次に、直線 3x−2y+4=0 に沿ったデータポイントを生成する。x 座標は一定の間隔で -10 から 10 まで変化し、対応する y 座標を計算する。また、Mersenne Twister エンジンを使用して正規分布に従うノイズを生成する。生成した x 座標と y 座標にノイズを追加し、新しい座標とノイズを格納する。

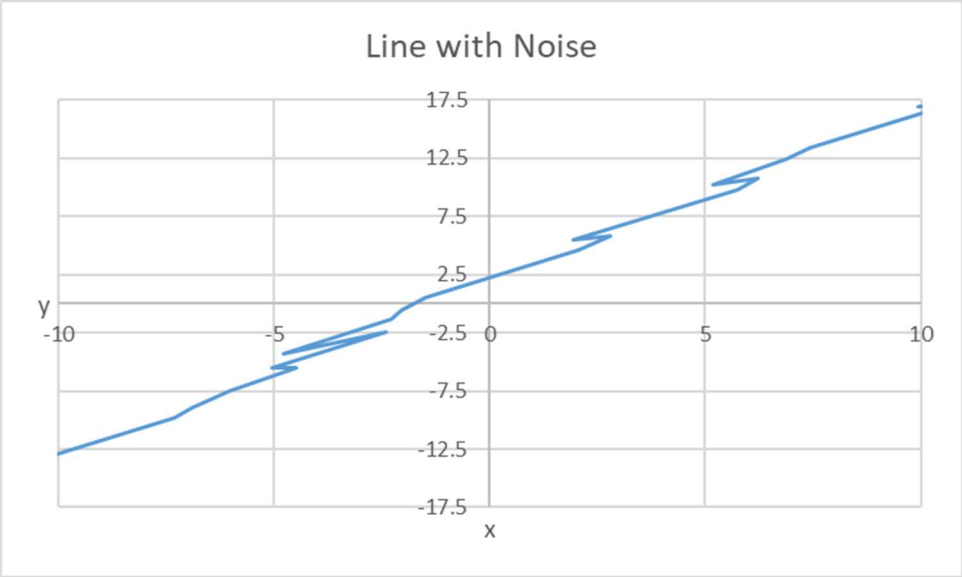
ノイズを付加した座標の分布(variance)を表示し、元の座標とノイズ付きの座標をファイル noisy_data.txt に出力する。

動作確認

x	y	x_noise	ynoise
-10	-13	-10.5696	-13.5696
-9	-11.5	-7.30417	-9.80417
-8	-10	-6.88913	-8.88913
-7	-8.5	-6.03241	-7.53241
-6	-7	-4.47864	-5.47864
-5	-5.5	-5.04825	-5.54825
-4	-4	-2.4047	-2.4047
-3	-2.5	-4.78309	-4.28309
-2	-1	-2.28645	-1.28645
-1	0.5	-2.04197	-0.541969
0	2	-1.47965	0.520355
1	3.5	2.07128	4.57128
2	5	2.80748	5.80748
3	6.5	1.94173	5.44173
4	8	5.73002	9.73002
5	9.5	6.23265	10.7327
6	11	5.1764	10.1764

7	12.5	6.8818	12.3818
8	14	7.4239	13.4239
9	15.5	10.6957	17.1957
10	17	9.92561	16.9256

```
Variance of noisy x coordinates: 0.331533
Variance of noisy y coordinates: 1.13134
Variance of noise: 1.26808
Data saved to noisy_data.txt
```



Q4

プログラム

Q4.cpp

```
1: #include <iostream>
2: #include <fstream>
3: #include <vector>
4: #include <Eigen/Dense>
5:
6: // Function to read data from a file
```

```
7: bool readData(const std::string& filename, std::vector<double>& x, std::vector<double>& y)
{
8:         std::ifstream infile(filename);
9:         if (!infile.is_open()) {
10:                 std::cerr << "Error opening file: " << filename << std::endl;
11:                 return false;
12:         }
13:         // Skip the header line
14:         std::string header;
15:         std::getline(infile, header);
16:         double a, b, x_val, y_val;
17:         while (infile >> a >> b >> x_val >> y_val) {
18:                 x.push_back(x_val);
19:                 y.push_back(y_val);
20:         }
21:         infile.close();
22:         return true;
23: }
24:
25: // Function to perform least squares fitting
26: void leastSquaresFit(const std::vector<double>& x, const std::vector<double>& y, double&
slope, double& intercept) {
27:         int n = x.size();
28:         Eigen::MatrixXd A(n, 2);
29:         Eigen::VectorXd b(n);
30:
31:         // Fill matrix A and vector b
32:         for (int i = 0; i < n; ++i) {
33:                 A(i, 0) = x[i];
34:                 A(i, 1) = 1.0;
35:                 b[i] = y[i];
36:         }
37:
```

```
38:     // Solve the normal equation A^T A w = A^T b
39:     Eigen::VectorXd w = (A.transpose() * A).ldlt().solve(A.transpose() * b);
40:
41:     // Extract slope and intercept
42:     slope = w[0];
43:     intercept = w[1];
44: }
45:
46: int main() {
47:     // Vectors to store the x and y coordinates
48:     std::vector<double> x, y;
49:
50:     // Read data from file
51:     if (!readData("noisy_data.txt", x, y)) {
52:         return 1;
53:     }
54:
55:     // Variables to store the slope and intercept
56:     double slope, intercept;
57:
58:     // Perform least squares fitting
59:     leastSquaresFit(x, y, slope, intercept);
60:
61:     // Output the results
62:     std::cout << "Slope: " << slope << std::endl;
63:     std::cout << "Intercept: " << intercept << std::endl;
64:
65:     return 0;
66: }
67:
```

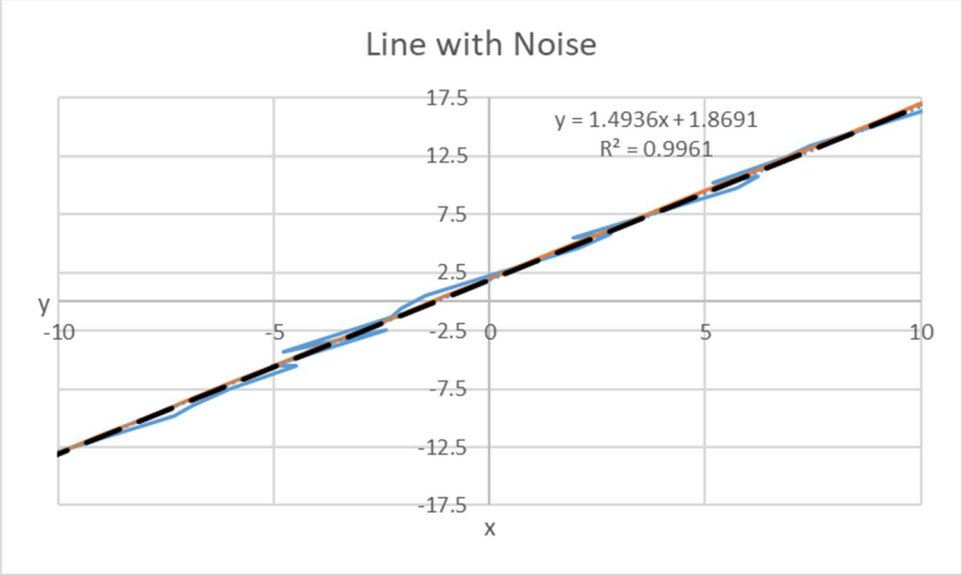
解説

ファイルからデータを読み込み、最小二乗法を用いて直線の傾きと切片を計算するプログラムである。readData 関数は、指定されたファイルからデータを読み込み、ファイルが開けなかった場合はエラーメッセージを表示し、false を返する。ファイルが正常に開けた場合は、データを読み込んで x および y ベクトルに格納し、true を返する。次に、leastSquaresFit 関数は、与えられた x および y のデータを用いて最小二乗法による直線フィッティングを行う。Eigen ライブラリを用いて、正規方程式 $A^TAw=A^Tb$を解くことで、傾き（slope）と切片（intercept）を求める。

メイン関数では、x および y ベクトルを宣言し、ファイルからデータを読み込む。読み込みが成功した場合は、最小二乗法によるフィッティングを行い、傾きと切片を計算する。グラフは excel で作成した。Excel に夜の結果と c++の結果は一致している。

動作確認

```
Slope: 1.49356
Intercept: 1.86913
```



自己チェック項目

4 乱数ライブラリの使い方を理解した.

3 行列· ベクトル演算ライブラリを使用することができる.

4 最小二乗法を理解して実装することができる.