ソフトウェア演習２B

ライブラリの使用

標準テンプレートライブラリ（STL）

- 標準テンプレートライブラリと呼ばれる便利なライブラリが標準で用意されている
- データ構造に関するライブラリ（クラス）
  - stringクラス・・・文字列を扱うクラス（#include<string>）
  - vectorクラス・・・配列を拡張したようなコンテナクラス（#include<vector>）
- アルゴリズムに関するライブラリ（クラス）
  - sortクラス・・・データのソートを行うクラス　（#include<algorithm>）
  - stackクラス・・・スタック構造とスタック構造によるデータ処理を扱うクラス（#include<stack>）
  - queueクラス・・・キュー構造とキュー構造によるデータ処理を扱うクラス（#include<queue>）
  - 乱数に関するライブラリ・・・疑似乱数を発生させるライブラリ（#include<random>）

疑似乱数を生成する関数

- 高品質な疑似乱数霊を生成可能なメルセンヌ・ツイスタを使用可能
- 乱数発生の流れ

1. 乱数の種(seed)をもとに乱数生成器を初期化する

```
std::uint32_t seed = (std::uint32_t) atoi (argv[1]);
std::mt19937 method (seed);
```

2. どのような分布の乱数を発生されるかを決める

```
正規乱数： std::normal_distribution<> dist (0.0, 1.0);
                                            平均  標準偏差
一様乱数： std::uniform_int_distribution<> dist (0, 99);
                                                 値の範囲
```

3. 乱数を生成する

```
double val = dist (method);
```

ソート

- 配列やvectorクラスなどのコンテナデータをソートすることができる
- 配列データのソート方法：

```
int nData = 100;
int data[nData];
for (auto& val: data) val = …;
```

昇順ソート：　std::sort (data, data+nData);
降順ソート：　std::sort (data, data+nData, std::greater<double>());

ソート

- コンテナクラスデータのソート方法：

```
int nData = 100;
vector<double> data;
for (int n = 0; n < nData; n++) data.push_back(…);
```

昇順ソート：　std::sort (data.begin(), data.end());
降順ソート：　std::sort (data.rbegin(), data.rend());

行列・ベクトル計算ライブラリ：eigen

- 行列とベクトルの演算を実装したライブラリ
- 直感的に行列、ベクトルの演算を行うことができる
- 基本的な演算から連立方程式の解計算や固有値計算なども実装されている
- 使用する場合には以下のようにファイルをインクルードする

```
#include<Eigen/Dense>
```

ベクトル:宣言と初期化

- ベクトルの宣言

```
Eigen::VectorXd a(3);
```

- ベクトルの初期化
  - 値をすべて 0 に初期化

```
Eigen::VectorXd a = Eigen::VectorXd::Zero(3);
```

  - 値をすべて 1 に初期化

```
Eigen::VectorXd a = Eigen::VectorXd::Ones(3);
```

ベクトル:値へのアクセス

- 値の代入

```
for (int n = 0; n < 3; n++) a(n) = n;
a << 1, 2, 3;
a = Eigen::VectorXd::Zero(3);
```

- 値の取得

```
double val = a(0);
```

ベクトル:基本演算

- ベクトル同士の加減算

```
c = a + b;
```

- ベクトルのスカラー倍

```
double c = 2.0;
b = c * a;
```

- ベクトルの内積

```
double c = a.dot (b);
```

- ベクトル同士の積（ベクトル積）

```
c = a * b.transpose();
```

- ベクトルの外積（３次元ベクトルの場合）

```
c = a .cross (b);
```

行列:宣言と初期化

- 行列の宣言

```
Eigen::MatrixXd A(3, 3);
```

- 行列の初期化
  - 値をすべて 0 に初期化

```
Eigen::MatrixXd a = Eigen::MatrixXd::Zero(3, 3);
```

  - 値をすべて 1 に初期化

```
Eigen::MatrixXd a = Eigen::MatrixXd::Ones(3, 3);
```

  - 単位行列に初期化

```
Eigen::MatrixXd a = Eigen::MatrixXd::Identity(3, 3);
```

行列:値へのアクセス

- 値の代入

```
for (int n = 0; n < 3; n++)
  for (m = 0; m < 3; m++) A(n, m) = 0.0;

Eigen::VectorXd a = Eigen::VectorXd::Zero(3);
A.col(0) = a;
A.row(2) = a;
```

- 値の取得

```
std::cout << A(0, 0) << std::endl;
```

行列:基本演算

- 行列同士の加減算

```
C = A + B;
```

- 行列のスカラー倍

```
double c = 2.0;
B = c * A;
```

- 行列同士の積

```
C = A * B;
```

- 行列とベクトルの積

```
c = A * b;
```

- 逆行列の計算

```
A_inv = A.inservse();
```

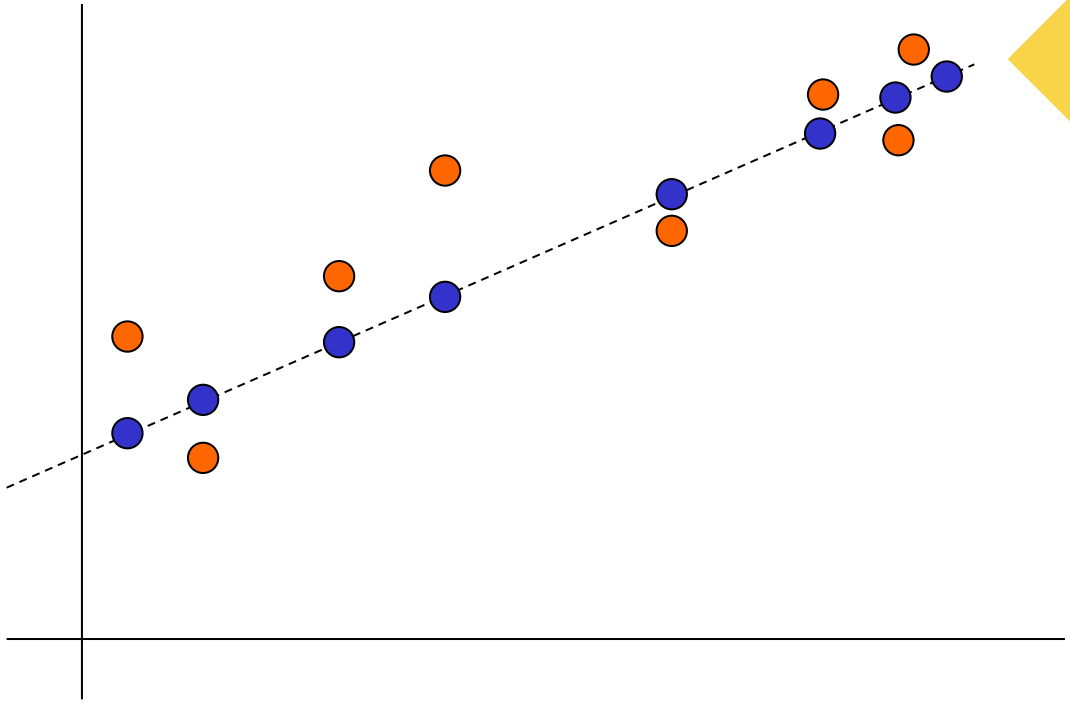
Eigenを使用するプログラムのコンパイル

- pkg-config というプログラムを使ってライブラリのオプションを指定する

```
% cd report4/4-2
% g++ 4-2.cpp –o program4-2 `pkg-config –cflags eigen3`
% ./program4-2
```

最小二乗法による直線パラメータの推定

- データ点列 $(x_\alpha, y_\alpha), \alpha = 1, \ldots, N$ に直線 $ax + by + c = 0$ を当てはめたい。

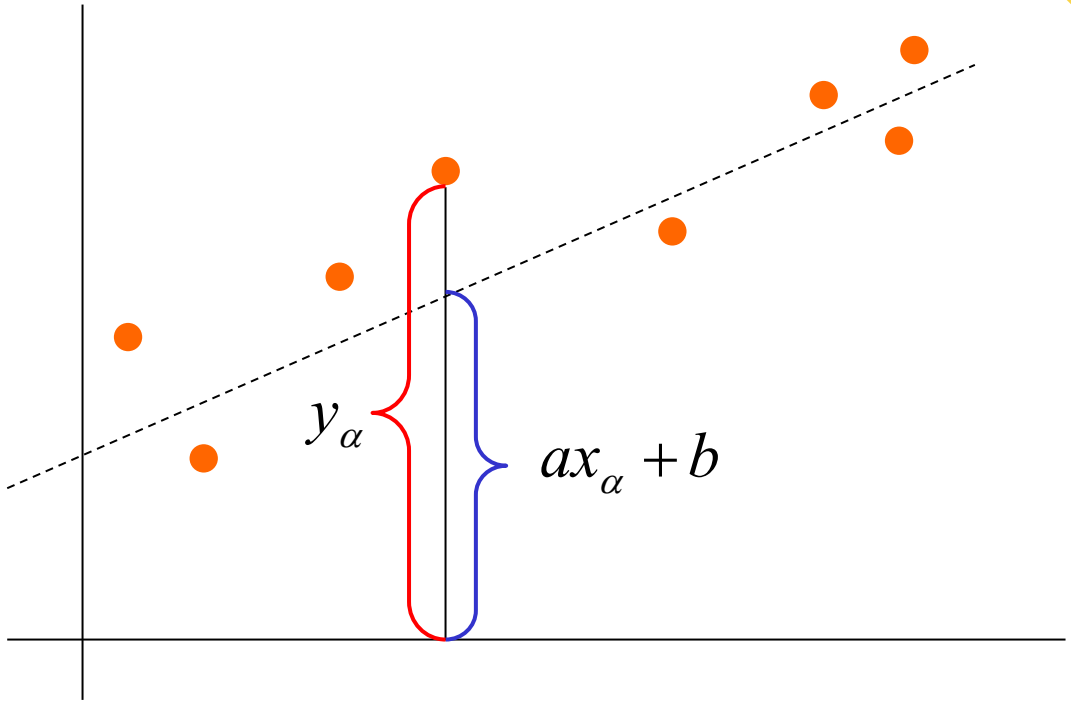


- データに誤差がない場合、$ax_\alpha + by_\alpha + c = 0, \alpha = 1, \ldots, N$
- データに誤差がある場合、$ax_\alpha + by_\alpha + c \neq 0, \alpha = 1, \ldots, N$

直線当てはめ

- できるだけ当てはめた直線とデータが近くなるようにパラメータを推定する。

- 次の目的関数を最小化するパラメータを推定する方法を最小２乗法と呼ぶ。



最小２乗法による直線当てはめ

- 観測データ$(x_\alpha, y_\alpha), \alpha = 1, \dots, N$ に直線$y = ax + b$ を当てはめる。
  - 目的関数

$$J = \frac{1}{2}\sum_{\alpha=1}^{N}\left(y_\alpha - (ax_\alpha + b)\right)^2$$

を各パラメータa, bで偏微分して0と置いた式を解けばよい。

$$\frac{\partial J}{\partial a} = \sum_{\alpha=1}^{N}(y_\alpha - ax_\alpha - b)(-x_\alpha) = a\sum_{\alpha=1}^{N}x_\alpha^2 + b\sum_{\alpha=1}^{N}x_\alpha - \sum_{\alpha=1}^{N}x_\alpha y_\alpha = 0$$

$$\frac{\partial J}{\partial b} = \sum_{\alpha=1}^{N}(y_\alpha - ax_\alpha - b)(-1) = a\sum_{\alpha=1}^{N}x\_\alpha + b\sum_{\alpha=1}^{N}1 - \sum_{\alpha=1}^{N}y_\alpha = 0$$

$$\begin{pmatrix} \sum x_\alpha^2 & \sum x_\alpha \\ \sum x_\alpha & \sum 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum x_\alpha y_\alpha \\ \sum y_\alpha \end{pmatrix}$$

正規方程式