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);
一様乱数：  std::uniform_int_distribution<> dist (0, 99);
```

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;
```

行列:固有値・固有ベクトルの計算

- 正方行列Aの固有値と固有ベクトルを計算するには次のようにする

```
int N = 3;
Eigen::MatrixXd A(N, N);
…
Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> eigensolver (A);
Eigen::VectorXd evals = eigensolver.eigenvalues();
Eigen::MatrixXd evecs = eigensolver.eigenvectors();
```

- 固有値は小さい順にソートされたベクトルとして得られる

$$\mathrm{evals}(0) \leq \mathrm{evals}(1) \leq \cdots \leq \mathrm{evals}(N)$$

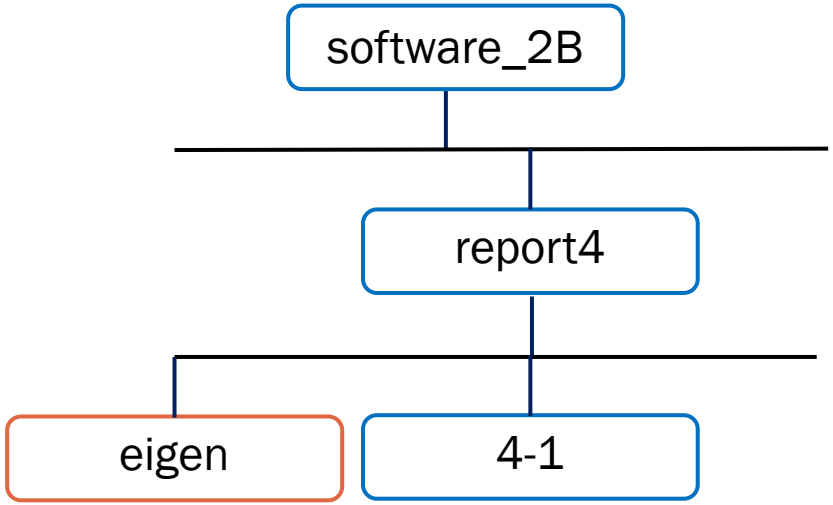
- ソートされた固有ベクトルに対応する単位固有ベクトルが列ベクトルとして格納された行列として得られる

```
Eigen::VectorXd min_evec = evec.col(0); // 最小固有値に対する単位固有ベクトル
```

eigenのインストール

- 演習課題を行うために各自のディレクトリにeigenをインストールする
- classroomからファイルをダウンロードして、以下のディレクトリ構成になるように展開する

```
% cd
% software_2B/report4
% unzip ファイルをダウンロードした場所/eigen.zip
```



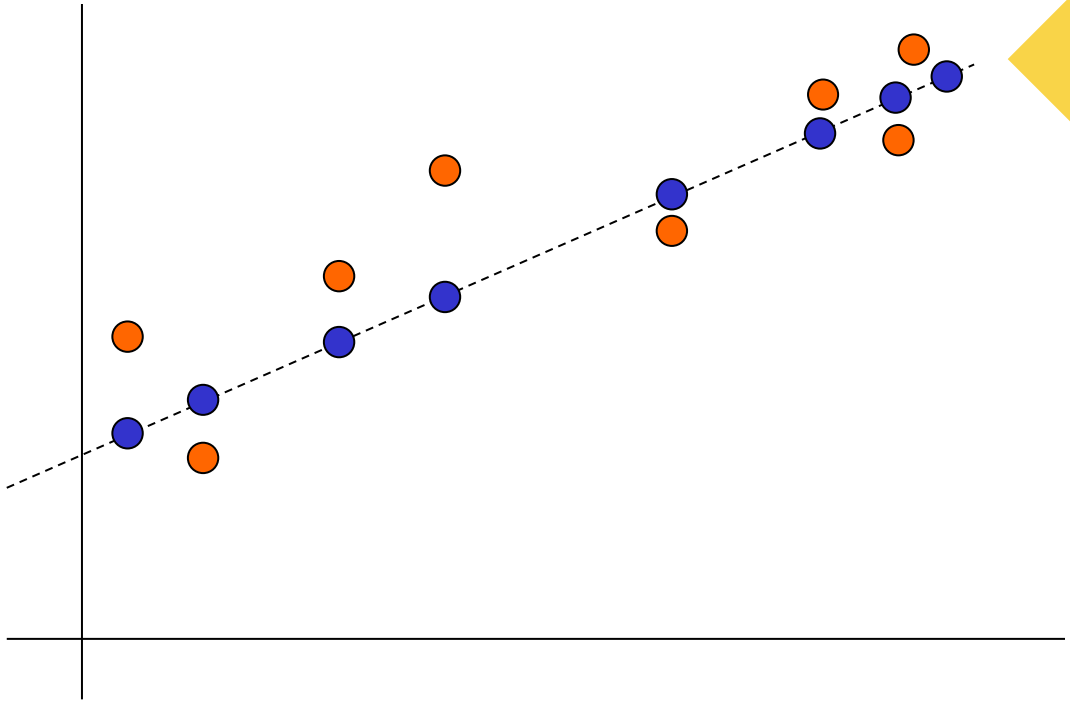
Eigenを使用するプログラムのコンパイル

- g++の-Iオプションを使ってライブラリファイルの場所を指定する

```
% cd report4/4-2
% g++ 4-2.cpp –I../eigen –o program4-2
% ./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、$ax_\alpha + by_\alpha + c \neq 0, \alpha = 1, \ldots, N$

最小化する関数の定義

- 直線 $ax + by + c = 0$ に対して、パラメータベクトルとデータベクトルを以下のように定義する

$$\boldsymbol{u} = (A, B, C)^\top, \qquad \boldsymbol{x} = (x, y, 1)^\top$$

- 最小化すべき関数は次のように定義することができる

$$J = \sum_{\alpha=1}^{N} (\boldsymbol{u}^\top \boldsymbol{x}_\alpha)^2$$

関数の変形

- 定義した関数は2次関数であるから、それを最小化する解は関数をパラメータで微分したものを0とおいて、それを解いたものである
- 関数は次のように書き直すことができる

$$J = \sum_{\alpha=1}^{N} (\boldsymbol{u}^\top \boldsymbol{x}_\alpha)^2 = \sum_{\alpha=1}^{N} (\boldsymbol{u}^\top \boldsymbol{x}_\alpha)(\boldsymbol{x}_\alpha^\top \boldsymbol{u})$$

$$= \sum_{\alpha=1}^{N} \boldsymbol{u}^\top (\boldsymbol{x}_\alpha \boldsymbol{x}_\alpha^\top \boldsymbol{u})$$

$$= \boldsymbol{u}^\top (\sum_{\alpha=1}^{N} \boldsymbol{x}_\alpha \boldsymbol{x}_\alpha^\top) \boldsymbol{u}$$

$$= \boldsymbol{u}^\top \boldsymbol{M} \boldsymbol{u}, \quad \boldsymbol{M} = \sum_{\alpha=1}^{N} \boldsymbol{x}_\alpha \boldsymbol{x}_\alpha^\top$$

関数の最小化

- 関数 Jをパラメータ $\boldsymbol{u}$ で微分して$\mathbf{0}$ とおくと、次のようになる

$$\frac{\partial J}{\partial \boldsymbol{u}} = 2\boldsymbol{M}\boldsymbol{u} = \mathbf{0}$$

- 上式を満たすベクトル $\boldsymbol{u}$ は、行列 $\boldsymbol{M}$ の固有値 0 に対応する固有ベクトルである

- $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{u}$ を最小化するパラメータ $\boldsymbol{u}$ は、行列 $\boldsymbol{M}$ の最小固有値に対する単位固有ベクトルとして得られる