Linear Data Fitting

설명

N개의 data (x_i, y_i, x_i', y_i') 와 선형 관계 모델 $x' = a_1x + a_2y + a_3, y' = a_4x + a_5y + a_6$ 가 주어질 때,
최적의 파라미터 $a = (a_1, a_2, a_3, a_4, a_5, a_6)$ 을 찾아라.

이를 풀기 위해 두 번의 General Linear Least Square를 이용하자.

Model A: $y(z) = \Sigma c_j f_j(z_i)$ where $c_j = a_j \; j \in \{1, 2, 3\}$ and
$f_1(z_i) = x_i, f_2(z_i) = y_i, f_3(z_i) = 1, y_i = x'$

$$e_i = y_i - y(z_i) \tag{1}$$

$$J = \begin{pmatrix} \frac{\partial e_1}{\partial c_1} & \frac{\partial e_1}{\partial c_2} & \frac{\partial e_1}{\partial c_3} \\ \frac{\partial e_2}{\partial c_1} & \frac{\partial e_2}{\partial c_2} & \frac{\partial e_3}{\partial c_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial e_N}{\partial c_1} & \frac{\partial e_N}{\partial c_2} & \frac{\partial e_N}{\partial c_3} \end{pmatrix} = \begin{pmatrix} -f_1(z_1) & -f_2(z_1) & -f_3(z_1) \\ -f_1(z_2) & -f_2(z_2) & -f_3(z_2) \\ \vdots & \vdots & \vdots \\ -f_1(z_N) & -f_2(z_N) & -f_3(z_N) \end{pmatrix} \tag{2}$$

(1), (2) 식으로부터 $e = x' - (-Jc)$ 를 얻을 수 있고,

$J^Te = 0$ 로 부터 $J^Te = J^T(x' + Jc) = J^Tx' + J^TJc = 0$

즉, 다음과 같은 선형 시스템을 풀면 a_1, a_2, a_3을 구할 수 있다.

$$J^TJc = -J^Tx' \tag{3}$$

마찬가지 방법으로

Model B: $y(z) = \Sigma c_j f_j(z_i)$ where $c_j = a_j \; j \in \{4, 5, 6\}$ and
$f_1(z_i) = x_i, f_2(z_i) = y_i, f_3(z_i) = 1, y_i = y'$ 일때,

$$J^TJc = -J^Ty' \tag{4}$$

를 풀면 a_4, a_5, a_6도 구할 수 있다.

구현

```
result_t* a_123 = least_square(data, x, m, n);
result_t* a_456 = least_square(data, y, m, n);
printf("The result is: %lf, %lf, %lf, %lf, %lf, %lf\n", a_123[1], a_123[2], a_123[3],
a_456[1], a_456[2], a_456[3]);
free_dvector(a_123, 1, n);
free_dvector(a_456, 1, n);
```

호출부

```
static result_t* least_square(const result_t** data, const result_t* y, int m, int n){
    result_t* c = dvector(1, n);
    ...
    // 1. Calculating J_t
    // where J^T = -[
    //    [x_1, x_2, ..., x_m],
    //    [y_1, y_2, ..., y_m],
    //    [1  , 1  , ...,   1],
    // ]
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            J_t[i][j] = -f[i-1](data[j]);
        }
    }

    // 2. Calculating A = J^T * J
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            A[i][j] = 0;
            for(int k = 1; k <= m; ++k){
                A[i][j] += J_t[i][k] * J_t[j][k];
            }
        }
    }

    // 3. Calculating yy = -J^T * y
    for(int i = 1; i <= n; i++){
        c[i] = 0;
        for(int j = 1; j <= m; j++){
            c[i] += -J_t[i][j] * y[j];
        }
    }

    // 4. Solve Ac=yy (J^T * J * c = -J^T * y)
    /* 4-1. Perform the decomposition */
    ludcmp(A,n,indx,&determinant);
    /* 4-2. Back Substitution */
    lubksb(A,n,indx,c);

    ...

    return c;
}
```

least_square 구현

- J^T 계산
- $A = J^T J$ 계산
- $yy = -J^T y$ 계산

- $Ac = yy$ 풀기 (즉, $J^T Jc = -J^T y$ 풀기)

빌드 방법

> ./build.sh

실행 1

> ./main < ../fitdata/fitdata1.dat

해석 1

> The result is: 0.981888, 0.002541, -0.375174, 0.001250, 0.982163, 1.157715

실행 2

> ./main < ../fitdata/fitdata2.dat

해석 2

> The result is: 0.979907, 0.000452, -1.192227, -0.001069, 0.980346, 0.491568

실행 3

> ./main < ../fitdata/fitdata3.dat

해석 3

> The result is: 0.980806, 0.000545, -0.944459, -0.000717, 0.979108, 0.428936

Extra Experiments

> python3 generator.py **n** | ./main **n**

Number of Data	a_1	a_2	a_3	a_4	a_5	a_6
Original	1.5	2	-7	3.14	0	1.2345
10	0.738788	1.942012	-6.790573	2.378788	-0.057988	1.443927
100	1.455406	2.093235	-7.045538	3.095406	0.093235	1.188962
1000	1.521919	2.086630	-7.026462	3.161919	0.086630	1.208038
10000	1.509332	2.015298	-7.016249	3.149332	0.015298	1.218251
100000	1.515524	2.002524	-7.009885	3.155524	0.002524	1.224615

주어진 Data가 많아질수록 더 정확한 값으로(오차가 줄어드는 방향으로) fitting할 수 있음을 알 수 있다.
이는 많은 Data를 이용해 least square를 할 경우 큰 수의 법칙에 의해 노이즈의 평균이 0으로 가까이 이동하기 때문이다.