Assignment9 보고서

2015004239 정성운

원본 : https://github.com/OdysseyJ/Numerical_Analysis/wiki/Assignment9

Assignment9

SeongWoon Jeong edited this page 5 minutes ago · 4 revisions

Numerical Analysis

Programming Assignment9

professor. 박종일

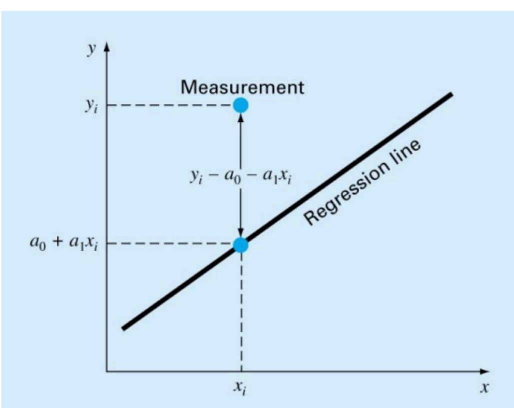
Linear Data Fitting.

1. 개요

- 이번 과제는 주어진 데이터를 Linear하게 Fitting하는 문제입니다. 데이터가 주어지면, 주어진 데이터를 최적으로 Fitting하는 best set of parameters를 구해내면 되는 문제입니다.

2. 과제 설명

- Linear Data Fitting



분포된 데이터들을 가장 잘 표현할 수 있는 Linear한 특정 값들을 구해내는 기법입니다.

- 실행
  - console을 이용해 실행, 원하는 데이터 파일 입력.

```
~/Documents/2019-2/Numerical_analysis/test  gcc -o Assignment9 Assignment9.c
Assignment9.c:14:12: warning: 'extern' variable has an initializer
      [-Wextern-initializer]
extern int err = 0;
           ^
Assignment9.c:15:14: warning: 'extern' variable has an initializer
      [-Wextern-initializer]
extern float det = 1.0;
             ^
2 warnings generated.
```

- 출력 형식
  - console에 출력.

```
2 warnings generated.
 ~/Documents/2019-2/Numerical_analysis/test  ./Assignment9
filename: fitdata1.dat
Linear Data Fitting
Solutions)
a[1]: 0.981888
a[2]: 0.002541
a[3]: -0.375173
a[4]: 0.001250
a[5]: 0.982163
a[6]: 1.157716
```

3. 사용언어 / 환경

C++/Mac OS

[실행 결과]

각각의 fitdata들에 대한 실행 결과 화면입니다.

[fitdata1.dat]

```
2 warnings generated.
~/Documents/2019-2/Numerical_analysis/test  ./Assignment9
filename: fitdata1.dat
Linear Data Fitting
Solutions)
a[1]: 0.981888
a[2]: 0.002541
a[3]: -0.375173
a[4]: 0.001250
a[5]: 0.982163
a[6]: 1.157716
```

[fitdata2.dat]

```
~/Documents/2019-2/Numerical_analysis/test  ./Assignment9
filename: fitdata2.dat
Linear Data Fitting
Solutions)
a[1]: 0.979907
a[2]: 0.000452
a[3]: -1.192227
a[4]: -0.001069
a[5]: 0.980346
a[6]: 0.491565
```

[fitdata3.dat]

```
~/Documents/2019-2/Numerical_analysis/test  ./Assignment9
filename: fitdata3.dat
Linear Data Fitting
Solutions)
a[1]: 0.980806
a[2]: 0.000545
a[3]: -0.944458
a[4]: -0.000717
a[5]: 0.979108
a[6]: 0.428937
```

[결과에 대한 분석]

[fitdata 양식]

$$(x_i, \; y_i, \; x_i', \; y_i'),$$

- 주어진 fitdata는, fitting 전의 데이터 (x,y), 그리고 fitting 후의 데이터 (x프라임) (y프라임)로 나눌 수 있습니다. x,y는 fitting전의 좌표값이며, fitting후에 나오는 x프라임과 y프라임은 각각 독립된 결과값입니다. 따라서 주어진 결과 a1, a2, a3 // a4, a5, a6는 서로 독립적인 값입니다.
- 따라서 우리는 J^T * J * (x,y) = J(x프라임)과, J^T * J * (x,y) = J(y프라임)값을 각각 독립적으로 계산하면 됩니다.

[데이터 fitting 방식]

- 데이터 fitting은 이전 x,y좌표와 fitting후의 x프라임,y프라임의 좌표를 통해 데이터를 fitting시키는 최적의 계수(여기서는 a)를 찾아내는 방식입니다.

$$x' = a_1 x + a_2 y + a_3$$
$$y' = a_4 x + a_5 y + a_6$$

- 최적의 계수를 찾기 위해서, 우리는 jacobian matrix를 이용합니다. (여기서는 c가 구하려는 최적의 계수임)

$$\boxed{\therefore \; \mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{c} = \mathbf{J}^{\mathrm{T}}\mathbf{y}}$$

- 해당 식을 사용해 fitdata1, fitdata2, fitdata3의 최적의 계수를 각각 구해내면, 아래와 같은 결과를 얻을 수 있고, 이는 해당 파일의 데이터들을 가장 잘 fitting시키는 계수입니다.

```
2 warnings generated.
 ~/Documents/2019-2/Numerical_analysis/test  ./Assignment9
filename: fitdata1.dat
Linear Data Fitting
Solutions)
a[1]: 0.981888
a[2]: 0.002541
a[3]: -0.375173
a[4]: 0.001250
a[5]: 0.982163
a[6]: 1.157716
```

```
 ~/Documents/2019-2/Numerical_analysis/test  ./Assignment9
filename: fitdata2.dat
Linear Data Fitting
Solutions)
a[1]: 0.979907
a[2]: 0.000452
a[3]: -1.192227
a[4]: -0.001069
a[5]: 0.980346
a[6]: 0.491565
```

```
 ~/Documents/2019-2/Numerical_analysis/test  ./Assignment9
filename: fitdata3.dat
Linear Data Fitting
Solutions)
a[1]: 0.980806
a[2]: 0.000545
a[3]: -0.944458
a[4]: -0.000717
a[5]: 0.979108
a[6]: 0.428937
```

[전체 소스코드]

[Assignment9.c]

```c
// written by odysseyj
// HYU_2015004239 정성운
// https://github.com/OdysseyJ

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;}
#define NR_END 1
#define FREE_ARG char*
#define TINY 1.0e-20

extern int err = 0;
extern float det = 1.0;

void nrerror(char msg[]);
float *vector(long nl, long nh);
void free_vector(float *v, long nl, long nh);
void ludcmp(double a[][4], int n, int *indx, float *d);
void lubksb(double a[][4], int n, int *indx, double b[]);

int main(void) {
        double x[200], y[200], x1[200], y1[200];

    double J_tranJ[4][4], J[200][4], tranJ[4][200];

    double sol[4];
    char fileName[50];
    int indx[4];
    int count = 0;
    int i, j, k;
        double temp;
        float d;
```

```c
// 파일 읽기
printf("filename: ");
scanf("%s", fileName);
FILE *fp = fopen(fileName, "r");
while (fscanf(fp, "%lf", &temp) != EOF) x[count] = tem
// 파일 읽기 종료

// Jacovian Transpose, Jacovian matrix 채우기
for (i = 0; i < count; i++) {
    tranJ[1][i + 1] = J[i + 1][1] = x[i];
    tranJ[2][i + 1] = J[i + 1][2] = y[i];
    tranJ[3][i + 1] = J[i + 1][3] = 1;
}

// J^T * J  채우기
for (i = 1; i <= 3; i++){
    for (j = 1; j <= 3; j++){
        J_tranJ[i][j] = 0;
    }
}
for (i = 1; i <= 3; i++){
    for (j = 1; j <= 3; j++){
        for (k = 1; k <= count; k++){
            J_tranJ[i][j] += tranJ[i][k] * J[k][j];
        }
    }
}

// 우항 만들기.
    sol[1] = sol[2] = sol[3] = 0;
    for (i = 1; i <= count; i++) {
            sol[1] += x[i - 1] * x1[i - 1];
            sol[2] += y[i - 1] * x1[i - 1];
            sol[3] += x1[i - 1];
    }
```

```
    // J_tranJ 매트릭스 다시 분해하기. (LU backsubstitution 사용을
        ludcmp(J_tranJ, 3, indx, &d);

    // lu back substitution으로 정답 구하기
        lubksb(J_tranJ, 3, indx, sol);

    printf("Linear Data Fitting\n");
    printf("Solutions)\n");
    for (i = 1; i <= 3; i++) {
        printf("a[%d]: %lf\n", i, sol[i]);
    }

    // 우항 만들기.
        sol[1] = sol[2] = sol[3] = 0;
        for (j = 1; j <= count; j++) {
                sol[1] += x[j - 1] * y1[j - 1];
                sol[2] += y[j - 1] * y1[j - 1];
                sol[3] += y1[j - 1];
        }

    // J_tranJ 매트릭스 다시 분해하기.
        lubksb(J_tranJ, 3, indx, sol);

    // LU back substitution으로 정답 구하기.
    for (i = 1; i <= 3; i++) {
        printf("a[%d]: %lf\n", i + 3, sol[i]);
    }
        return 0;
}

void nrerror(char msg[]) {
```

```c
void nrerror(char msg[]) {
    printf("%s", msg);
    exit(1);
}

float *vector(long nl, long nh)
/* allocate a float vector with subscript range v[nl..nh]
{
    float *v;
    v = (float *)malloc((size_t)((nh - nl + 1 + NR_END) *
    if (!v) nrerror("allocation failure in vector()");
    return v - nl + NR_END;
}

void free_vector(float *v, long nl, long nh)
/* free a float vector allocated with vector() */
{
    free((FREE_ARG)(v + nl - NR_END));
}
```

```c
// LU_decomposition
void ludcmp(double a[][4], int n, int *indx, float *d)
{
    int i, imax, j, k;
    float big, dum, sum, temp;
    float *vv;
    vv = vector(1, n);
    *d = 1.0;
    for (i = 1; i <= n; i++) {
        big = 0.0;
        for (j = 1; j <= n; j++)
            if ((temp = fabs(a[i][j])) > big) big = temp;
        if (big == 0.0) {
            printf("Singular matrix in routine ludcmp\n");
            err++;
            return;
        }
        vv[i] = 1.0 / big;
    }
    for (j = 1; j <= n; j++) {
        for (i = 1; i<j; i++) {
            sum = a[i][j];
            for (k = 1; k<i; k++) sum -= a[i][k] * a[k][j]
            a[i][j] = sum;
        }
        big = 0.0;
        for (i = j; i <= n; i++) {
            sum = a[i][j];
            for (k = 1; k<j; k++)
                sum -= a[i][k] * a[k][j];
            a[i][j] = sum;
            if ((dum = vv[i] * fabs(sum)) >= big) {
                big = dum;
                imax = i;
            }
        }
        if (j != imax) {
            for (k = 1; k <= n; k++) {
                dum = a[imax][k];
                a[imax][k] = a[j][k];
                a[j][k] = dum;
            }
```

```c
        }
        if (j != imax) {
            for (k = 1; k <= n; k++) {
                dum = a[imax][k];
                a[imax][k] = a[j][k];
                a[j][k] = dum;
            }
            *d = -(*d);
            vv[imax] = vv[j];
        }
        indx[j] = imax;
        if (a[j][j] == 0.0) a[j][j] = TINY;
        if (j != n) {
            dum = 1.0 / (a[j][j]);
            for (i = j + 1; i <= n; i++) a[i][j] *= dum;
        }
    }
    free_vector(vv, 1, n);
}
```

```c
// LU_backsubstitution
void lubksb(double a[][4], int n, int *indx, double b[])
{
    int i, ii = 0, ip, j;
    float sum;
    for (i = 1; i <= n; i++) {
        ip = indx[i];
        sum = b[ip];
        b[ip] = b[i];
        if (ii)
            for (j = ii; j <= i - 1; j++) sum -= a[i][j] *
        else if (sum) ii = i;
        b[i] = sum;
    }
    for (i = n; i >= 1; i--) {
        sum = b[i];
        for (j = i + 1; j <= n; j++) sum -= a[i][j] * b[j]
        b[i] = sum / a[i][i];
    }
}
```

[Line42]

```
// Jacovian Transpose, Jacovian matrix 채우기
for (i = 0; i < count; i++) {
    tranJ[1][i + 1] = J[i + 1][1] = x[i];
    tranJ[2][i + 1] = J[i + 1][2] = y[i];
    tranJ[3][i + 1] = J[i + 1][3] = 1;
}
```

Matrix form

$$\mathbf{J}^{\mathrm{T}}\mathbf{e} = \mathbf{0}$$

where $\mathbf{e} = [\mathbf{e}_1\ \mathbf{e}_2 \cdots \mathbf{e}_N]^{\mathrm{T}}$

$$\mathbf{J}^{\mathrm{T}} = \begin{bmatrix} \frac{\partial e_1}{\partial c_1} & \frac{\partial e_2}{\partial c_1} & \cdots & \frac{\partial e_N}{\partial c_1} \\ \frac{\partial e_1}{\partial c_2} & \ddots & & \frac{\partial e_N}{\partial c_2} \\ \vdots & & \ddots & \vdots \\ \frac{\partial e_1}{\partial c_M} & \frac{\partial e_2}{\partial c_M} & \cdots & \frac{\partial e_N}{\partial c_M} \end{bmatrix} = -\begin{bmatrix} f_1(x_1) & f_1(x_2) & \cdots & f_1(x_N) \\ f_2(x_1) & \ddots & & f_2(x_N) \\ \vdots & & \ddots & \vdots \\ f_M(x_1) & f_M(x_2) & \cdots & f_M(x_N) \end{bmatrix}$$

해당 방식으로 jacobvian matrix 채우는 코드입니다.

[Line64]

```
// 우항 만들기.
    sol[1] = sol[2] = sol[3] = 0;
    for (i = 1; i <= count; i++) {
            sol[1] += x[i - 1] * x1[i - 1];
            sol[2] += y[i - 1] * x1[i - 1];
            sol[3] += x1[i - 1];
    }
```

$$\boxed{\therefore\ \ \mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{c} = \mathbf{J}^{\mathrm{T}}\mathbf{y}}$$

해당 식에 대한 우항을 만드는 코드입니다.

[Line71]

```
        // J_tranJ 매트릭스 다시 분해하기. (LU backsubstitution
        ludcmp(J_tranJ, 3, indx, &d);

    // lu back substitution으로 정답 구하기
        lubksb(J_tranJ, 3, indx, sol);

    printf("Linear Data Fitting\n");
    printf("Solutions)\n");
    for (i = 1; i <= 3; i++) {
        printf("a[%d]: %lf\n", i, sol[i]);
    }
```

$$\boxed{\therefore \quad J^T J c = J^T y}$$

해당 식에서 좌항을 만드는 코드입니다. Ax=b의 꼴로 만들어서, backsubstitution을 이용해 답을 구합니다. sol에 답이 설정되어 있으므로 이를 출력합니다.