C/C++ Day 5

윤광진, 광주과학기술원 02/09/2018

1 Matrix 클래스

매트릭스 클래스를 구현해보자. 이 매트릭스 클래스는 임의의 크기의 행렬의 실수배, 덧셈, 뺄셈, 곱셈, 전치 등을 할수 있다. 즉 임의 크기 행렬 A 와 B, $k \in \mathbb{R}$ 대해서 kA, $A+B$, $A-B$, $A \cdot B$, A^T 를 할 수 있다. 행렬 클래스의 구현은 Listing 2에 있다. 그리고 이 클래스의 사용법을 다룬 **`main.cpp`** 는 Listing 3에 있다. Listing 3의 9-42줄은 저장된 파일로부터 매트릭스 원소를 입력 받는 방법이 예시되어 있다. 저장된 데이터 파일은 Listing 1와 같다.

Listing 1: 데이터 파일 (A.txt)

```
3 3
1 0 0
0 1 0
0 0 1
```

Listing 1의 첫번째 줄에서 첫번째 숫자는 입력받으려는 행렬의 행의 수이고 두번째 숫자는 열의 수다. 그리고 이어지는 두번째 줄부터 행렬의 원소들이 쓰여져 있다. 3×3 짜리 단위 행렬임을 볼수 있다. 같은 행에서는 공백으로 열을 구분하고 행은 개행(newline)으로 구분해서 쓴다. Listing 3의 63-71번줄은 임의 크기의 행렬을 만들어서 행렬의 원소를 직접 입력하는 방법이 나와있다. 그 밑으로 뺄셈, 전치, 곱셈 등을 하는 예시가 나와있다.

Listing 2: Matrix.h

```
1  #ifndef _MATRIX_HEADER_H_
2  #define _MATRIX_HEADER_H_
3
4  #include <vector>
5
6  template <class T>
7  class Matrix
8  {
9  private:
10     int r, c;
11     std::vector<std::vector<T>> data;
12
13 public:
14     Matrix() { r = 0; c = 0; };
15     Matrix(int row, int col);
16     Matrix(int row, int col, T* rawdata);
```

```
17      ~Matrix();
18
19      int cols() { return c; }
20      int rows() { return r; }
21      T& elem(int row, int col) { return data[row][col]; };
22
23      void transpose();
24      Matrix<T> transposeCopyto();
25      Matrix<T> operator+(Matrix<T> &B);
26      Matrix<T> operator-(Matrix<T> &B);
27      Matrix<T> operator*(Matrix<T> &B);
28      Matrix<T> operator*(T k);
29
30      friend Matrix<T> operator*(T k, Matrix<T> &B) { return B * k; };
31  };
32
33  template <class T>
34  Matrix<T>::Matrix(int row, int col)
35  {
36      r = row;
37      c = col;
38      for (int i = 0; i < r; ++i)
39          data.push_back(std::vector<T>(c));
40  }
41
42  template <class T>
43  Matrix<T>::Matrix(int row, int col, T* rawdata)
44  {
45      r = row;
46      c = col;
47      int k = 0;
48      for (int i = 0; i < r; ++i)
49      {
50          std::vector<T> temp;
51          for (int j = 0; j < c; ++j)
52          {
53              temp.push_back(rawdata[k]);
54              ++k;
55          }
56          data.push_back(temp);
57      }
```

```
58 }
59
60 template <class T>
61 Matrix<T>::~Matrix()
62 {
63     data.clear();
64 }
65
66 template <class T>
67 void Matrix<T>::transpose()
68 {
69     std::vector<std::vector<T>> buff;
70     int r2 = c;
71     int c2 = r;
72     for (int i = 0; i < r2; ++i)
73         buff.push_back(std::vector<T>(c2));
74
75     for (int i = 0; i < r; ++i)
76         for (int j = 0; j < c; ++j)
77             buff[j][i] = data[i][j];
78
79     r = r2;
80     c = c2;
81     data.swap(buff);
82 }
83
84 template <class T>
85 Matrix<T> Matrix<T>::transposeCopyto()
86 {
87     Matrix<T> trans(c, r);
88
89     for (int i = 0; i < r; ++i)
90         for (int j = 0; j < c; ++j)
91             trans.elem(j, i) = data[i][j];
92
93     return trans;
94 }
95
96 template <class T>
97 Matrix<T> Matrix<T>::operator+(Matrix<T> &B)
98 {
```

```
99      _ASSERT((c == B.cols()) && (r == B.rows()));
100
101     Matrix<T> C(r, c);
102
103     for (int i = 0; i < r; ++i)
104         for (int j = 0; j < c; ++j)
105             C.elem(i, j) = data[i][j] + (B.elem(i, j));
106
107     return C;
108 }
109 template <class T>
110 Matrix<T> Matrix<T>::operator-(Matrix<T> &B)
111 {
112     _ASSERT((c == B.cols()) && (r == B.rows()));
113
114     Matrix<T> C(r, c);
115
116     for (int i = 0; i < r; ++i)
117         for (int j = 0; j < c; ++j)
118             C.elem(i, j) = data[i][j] - (B.elem(i, j));
119
120     return C;
121 }
122 template <class T>
123 Matrix<T> Matrix<T>::operator*(Matrix<T> &B)
124 {
125     _ASSERT(c == B.rows());
126
127     int r2 = r;
128     int c2 = B.cols();
129     int K = c;
130     Matrix<T> C(r2, c2);
131
132     for (int i = 0; i < r2; ++i)
133     {
134         for (int j = 0; j < c2; ++j)
135         {
136             T sum = 0;
137             for (int k = 0; k < K; ++k)
138             {
139                 sum += data[i][k] * B.elem(k, j);
```

```
140                 }
141                 C.elem(i, j) = sum;
142             }
143         }
144 
145         return C;
146 }
147 template <class T>
148 Matrix<T> Matrix<T>::operator*(T k)
149 {
150         Matrix<T> C(r, c);
151 
152         for (int i = 0; i < r; ++i)
153             for (int j = 0; j < c; ++j)
154                 C.elem(i, j) = data[i][j] * k;
155 
156         return C;
157 }
158 
159 #endif
```

Listing 3: Template

```
1  #include "Matrix.h"
2  #include <iostream>
3  #include <fstream>
4  #include <string.h>
5  
6  using namespace std;
7  int main()
8  {
9          ifstream fin("A.txt");
10         if (!fin.is_open())
11             return -1;
12 
13         int row, col;
14         fin >> row;
15         fin >> col;
16 
17         double *raw = new double[row*col];
18         int idx = 0;
19         while (fin >> raw[idx])
```

```
20          ++idx;
21
22      _ASSERT(idx == row * col);
23
24      ifstream fin2("B.txt");
25      if (!fin2.is_open())
26          return -1;
27
28      int row2, col2;
29      fin2 >> row2;
30      fin2 >> col2;
31      double *raw2 = new double[row2*col2];
32      idx = 0;
33      while (fin2 >> raw2[idx])
34          ++idx;
35
36      _ASSERT(idx == row2 * col2);
37
38      Matrix<double> mat(row, col, raw);
39      Matrix<double> mat2(row2, col2, raw2);
40
41      delete[] raw;
42      delete[] raw2;
43
44      for (int i = 0; i < mat.rows(); ++i)
45      {
46          for (int j = 0; j < mat.cols(); ++j)
47          {
48              cout << mat.elem(i, j) <<", ";
49          }
50          cout << endl;
51      }
52
53      for (int i = 0; i < mat2.rows(); ++i)
54      {
55          for (int j = 0; j < mat2.cols(); ++j)
56          {
57              cout << mat2.elem(i, j) << ", ";
58          }
59          cout << endl;
60      }
```

```
61
62      Matrix<double> matrix(3, 2);
63      for (int i = 0; i < 3; ++i)
64      {
65          for (int j = 0; j < 2; ++j)
66          {
67              matrix.elem(i, j) = i + j;
68              cout << matrix.elem(i, j) << ", ";
69          }
70          cout << endl;
71      }
72
73      Matrix<double> mat3 = mat - mat2;
74
75      for (int i = 0; i < mat3.rows(); ++i)
76      {
77          for (int j = 0; j < mat3.cols(); ++j)
78          {
79              cout << mat3.elem(i, j) << ", ";
80          }
81          cout << endl;
82      }
83
84      mat2.transpose();
85      for (int i = 0; i < mat2.rows(); ++i)
86      {
87          for (int j = 0; j < mat2.cols(); ++j)
88          {
89              cout << mat2.elem(i, j) << ", ";
90          }
91          cout << endl;
92      }
93
94      Matrix<double> mat4 = mat * mat2 ;
95      for (int i = 0; i < mat4.rows(); ++i)
96      {
97          for (int j = 0; j < mat4.cols(); ++j)
98          {
99              cout << mat4.elem(i, j) << ", ";
100         }
101         cout << endl;
```

```
102     }
103
104     Matrix<double> mat5 = 10 * mat4 * 2;
105     for (int i = 0; i < mat5.rows(); ++i)
106     {
107         for (int j = 0; j < mat5.cols(); ++j)
108         {
109             cout << mat5.elem(i, j) << ", ";
110         }
111         cout << endl;
112     }
113     return 0;
114 }
```

2 Estimaing π

이번 절에서는 π 값을 근사해 볼 것이다. 방법은 Monte Carlo method로 한다[1]. 컴퓨터는 반복연산을 매우 빠르고 정확하게 할 수 있기 때문에 이와 같은 방법에 강하다. 이 시뮬레이션을 시각화 하기 위해서 **OpenCV**라는 영상처리 라이브러리를 사용할 것이다. 우선 **OpenCV**라이브러리 설치법부터 시작한다.

2.1 OpenCV 설치 및 라이브러리 연동

OpenCV 라이브러리를 https://opencv.org/releases.html에서 받는다. 최신 버전인 3.4.0에서 **Win pack**으로 받으면 된다. 적당한 곳에 압축을 풀자. 이제 압축을 푼 폴더를 **<ocv>** 라고 하겠다. **<ocv>** 하위에 opencv 폴더가 보일 것이다. 이제 비주얼 스튜디오로 새 프로젝트를 생성하자. 그리고 프로젝트 속성으로 이동한다. 즉, **Project** → **Properties ...** 로 속성 창을 연다. 여기서 좌측의 **Debugging** 을 선택하자. 오른쪽에서 **Environment**를 찾고 그 가장 우측의 드롭다운버튼(아래화살표)을 눌르고 **Edit...**를 선택해 편집창을 연다. 그곳에 PATH=%PATH%;**<ocv>**\opencv \build \x64 \vc15 \bin 을 입력한다. 다시 속성창 좌측에서 **C/C++** → **General**을 선택하자. 그후 우측에서 **Additional Include Directoris**에서 가장 우측의 드롭다운버튼 (아래화살표)을 누른후 **Edit...**를 눌러 편집 창을 띄우자. 여기서 **<ocv>**\opencv \build \include 를 입력한다. 그리고 다시 속성창의 왼쪽에서 **Linker** → **General**을 선택한다. 그리고 우측의 **Additional Library Directories**에서 가장 우측의 드롭다운버튼을 누른후 **Edit ...**를 눌러 편집창을 띄우자. 여기에 code<ocv>\opencv \build \x64 \vc15 \lib 를 선택한다. 마지막으로 다시 속성창에서 **Linker** → **Input**을 선택한 후 **Additional Dependencies**에서 가장 우측의 드롭다운버튼을 누른후 **Edit ...**를 눌러 편집창을 띄우자. 그리고 이곳에 **opencv_340d.lib** 를 입력하자. 그리고 Listing 5를 타이핑해서 결과를 확인하자. **show**란 이름의 검은색 창이 뜨면 된다.

Listing 4: OpenCV

```
1 #include <opencv2\opencv.hpp>
```

[1] https://en.wikipedia.org/wiki/Monte_Carlo_method

```
2
3  using namespace std;
4  using namespace cv;
5
6  void main()
7  {
8      int width = 500;
9      int height = 500;
10
11     Mat plain(Size(width, height), CV_8UC3);
12     plain.setTo(Scalar(0));
13     imshow("plain", plain);
14     waitKey();
15 }
```

2.2 Estimating π using the Monte Carlo Method

π를 근사하는 방법은 https://academo.org/demos/estimating-pi-monte-carlo/에 있는 방법을 이용할 것이다. 한변의 길이가 $2r$인 정사각형 R의 넓이는 $4r^2$이다. 그 정사각형 안을 꽉 채우는 원 C의 넓이는 πr^2이다. 그러면 이 둘의 넓이 비율을 $\frac{R}{C} = k$ 라하면 $k = \frac{\pi}{4}$가 된다. 여기서 우리는 k를 몬테 카를로 시뮬레이션으로 근사할 것이다. 방법은 간단하다. 점 하나를 무작위(Uniform distribution)로 찍어서 찍은 점이 원 안에 들어가면 N_{in}를 하나 증가 시킬것이다. N_T는 전체 시도횟수다. 즉, k를 $\frac{N_{in}}{N_T}$로 근사하는 것이다, $k \approx \frac{N_{in}}{N_T}$. 그러면 π는 $4k$를 함으로써 근사 될 수 있다.

그림 1이 실행화면이다. 코드는 Listing 5에 있다. N_T가 대략 35,000 이상일때 소숫점 둘째 자리 정도까지가 맞았다.

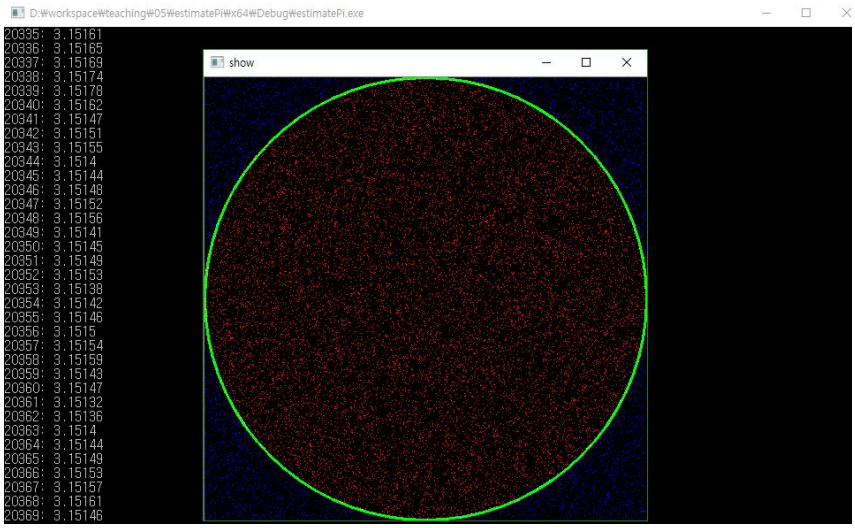

Figure 1: Caption

Listing 5: OpenCV

```cpp
#include <opencv2\opencv.hpp>
#include <random>

using namespace std;
using namespace cv;

void main()
{
    int radius = 250;
    long long max_try = 1000000000;
    long long ntry = 0;
    long long n_in = 0;
    Mat plain(Size(2 * radius + 1, 2 * radius + 1), CV_8UC3);
    plain.setTo(Scalar(0));
    circle(plain, Point(radius, radius), radius, Scalar(0, 255, 0), 2);

    default_random_engine gen;
    uniform_real_distribution<double> udis(-0.5, 0.5);
    while (ntry <= max_try)
    {
        ++ntry;
        double p1 = udis(gen);
        double p2 = udis(gen);
        int x = (p1 + 0.5) * 2. * radius;
        int y = (p2 + 0.5) * 2. * radius;

        double dist = sqrt((p1*p1) + (p2*p2));
        if (dist < 0.5) // in
        {
            plain.at<Vec3b>(x, y)[2] = 255;
            ++n_in;
        }
        else  // out
        {
            plain.at<Vec3b>(x, y)[0] = 255;
        }

        cout << ntry << ": " << 4.0*((double)n_in / (double)ntry) << endl;

        imshow("show", plain);

```

```
42            if (waitKey(1) == 27)
43                break;
44        }
45    }
```