1121 CS233 B 線性代數 HW2 函式說明

1113341 張維家

- 矩陣乘法

一、 想法與程式碼截圖

為了方便呼叫矩陣乘法函式時可以直接寫成 C=A*B ，設計一個 Matrix 的 class，裡面包含一個名為 element 的 4*4 double 陣列 data member，用以儲存矩陣之各元素值；以及一些 member function，其中之一即為 operator*：

```
Matrix operator*(Matrix right)
{
    Matrix product;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            for (int k = 0; k < 4; k++)
                product.element[i][j] += this->element[i][k] * right.element[k][j];

    return product;
}
```

二、 說明

假設有 3 個 n*n 的矩陣 A、B、C，且 C=AB，則矩陣乘法定義為「C 的第 i 列第 j 行為 A 的第 i 列各元素及 B 的第 j 行各元素依序（1~n）相乘之總和」，等同於以下公式：

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

在程式中此函式的呼叫指令是 C=A*B（三者皆為 Matrix 之物件），對應到前面程式碼 C 為 product、A 為*this、B 為 right。

- 求行列式值

一、 想法與程式碼截圖

求行列式值有兩種做法：

1. 透過餘因子展開（cofactor expansion）降階計算
2. 透過基本列運算求出上三角矩陣，將列交換次數乘以對角線值相乘得到行列式，即：

$$\det(T) = (-1)^{count} * u_{11} * u_{22} * \ldots * u_{nn} = (-1)^{count} \prod_{i=1}^{n} u_{ii}$$

這邊因為便利性和效率，使用第二種方式運算：

```cpp
double determinant(Matrix T)
{
    int count = 0;
    Matrix U = upper_triangle(T, count);
    double det = 1;
    for (int i = 0; i < 4; i++)
        det *= U.element[i][i];
    if (isnan(det)) return 0;
    return ((count & 1) ? -det : det);
}
```

為了讓求行列式的邏輯更直覺，將上三角轉換獨立寫成函式，否則這個 function 內的指令會太長，不方便閱讀和說明。

在 C++中，如果有 0.0/0.0 或無限大/無限大等情況，所得出來的值為 nan(ind)，表示該值並非一個合法數字(not a number, nan)，圖中倒數第二個指令之 isnan 是定義於<cmath>標頭檔的 function，它能協助判斷是否為 nan，若為 nan，則將行列值強制設為 0，避免掉一些會導致程式結果有誤之狀況。

上三角矩陣函式：

```cpp
Matrix upper_triangle(Matrix T, int& count)
{
    count = 0;
    Matrix U = T;
    for (int i = 0; i < 4; i++)
    {
        if (U.element[i][i] == 0)
        {
            for (int tempI = i + 1; tempI < 4; tempI++)
            {
                if (U.element[tempI][i] != 0) /*exchange two row*/
                {
                    for (int j = 0; j < 4; j++)
                    {
                        int temp = U.element[i][j];
                        U.element[i][j] = U.element[tempI][j];
                        U.element[tempI][j] = temp;
                    }
                }
            }
            count++;
        }

        for (int tempI = i + 1; tempI < 4; tempI++)
        {
            double sub = U.element[tempI][i] / U.element[i][i];
            for (int j = 0; j < 4; j++)
                U.element[tempI][j] -= sub * U.element[i][j];
        }
    }

    return U;
}
```

二、 說明

1. 求行列式值

先將原本的矩陣 T 轉換成上三角矩陣 U（過程中記錄列交換次數 count）後，再取(-1)的 count 次方，將其乘上 U 的對角項值，即為行列式值，如前所述之公式：

$$\det(T) = (-1)^{count} * u_{11} * u_{22} * \dots * u_{nn} = (-1)^{count} \prod_{i=1}^{n} u_{ii}$$

2. 上三角矩陣轉換

若一 n 階方陣可逆，則其行列式不等於 0，根據上述行列式公式可知，它的上三角矩陣斜對角項 t_{ii} 皆必不為零，又因為上三角矩陣的定義（0, ∀ i>j），因此對角項就是每列的非零首項（pivot），並且越靠下的列，pivot position 越靠右。

在 upper_triangle 這個函式中，先從第一列判斷對角項(t_{ii})是否為 0，若為 0，就從該列的下一列（i+1）開始找同行（i）非零的項。因為在越下方的非零首項應該越靠右，因此若第 i 列第 i 行之值為零，且第 k （i+1~n）列之第 i 行值不為零，則第 k 列應該要是較上方的列，兩列互換。舉例來說，若 t_{11} 為 0，那就依序找 t_{21}、t_{31}…，若 t_{21} 不為零，代表 t_2 應該在 t_1 上方，故需要將 t_1 和 t_2 兩列互換。同時記錄列互換次數 count，以便後面計算行列式時可以直接使用$(-1)^{count}$。

經過列互換後，現在 t_{ii} 不為 0，下一步需要將 i+1~n 列的第 i 行藉由基本列運算設成 0，以 i=1 舉例，t_{11} 不為 0，則 t_{21}、t_{31}、…、t_{n1} 皆須為 0。作法是將需要設為 0 的項（假設為第 k 列的第 i 項，k 為(i+1)~n）除以 t_{ii}，得到一個基數（基數是用來表示列運算中「$cR_i+R_k \rightarrow R_k$」中的 c），將第 i 列的每一項分別乘以基數，再減去第 k 列的對應項。若以 U 表示上三角矩陣、T 為原始矩陣、b 為基數、i 表示基準點位置、k 表示目前運算到第 k 列，j 表示第 j 行，上述文字可表示成：

$$b = t_{ki} / u_{ii}, \ \forall k = (i+1) \sim n$$
$$u_{kj} = t_{kj} - (t_{ij} * b), \ \forall j = 0 \sim n$$

- 求反矩陣

一、 想法與程式碼截圖

設有一 n 階方陣 A，求其反矩陣可以先將矩陣擴增為$[A|I_n]$，再用高斯 - 喬登消去法將矩陣化成$[I_n|A^{-1}]$，A^{-1} 即為 A 的反矩陣：

```
void inverse(Matrix T, Matrix& inverseT, double det)
{
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            inverseT.element[i][j] = (i == j ? 1 : 0);

    for (int i = 0; i < 4; i++)
    {

        if (T.element[i][i] == 0)
        {
            for (int j = 0; j < 4; j++)
            {
                T.element[i][j] += T.element[i + 1][j];
                inverseT.element[i][j] += inverseT.element[i + 1][j];
            }

            elimination(T, inverseT, i);
        }
        else
        {
            elimination(T, inverseT, i);
        }
    }
}
```

這邊也是將重複的指令另外寫成 function，elimination 這個函數主要是來實現高斯 - 喬登消去法。

高斯 - 喬登消去法：

```
void elimination(Matrix& T, Matrix& inverseT, int i)
{
    double base = T.element[i][i];
    for (int j = 0; j < 4; j++) /*set 1 (pivot)*/
        T.element[i][j] /= base, inverseT.element[i][j] /= base;
    for (int row = 0; row < 4; row++)
    {
        if (row == i) continue; /*set zero for not pivot items*/
        double base = T.element[row][i];
        for (int col = 0; col < 4; col++)
        {
            T.element[row][col] -= base * T.element[i][col];
            inverseT.element[row][col] -= base * inverseT.element[i][col];
        }
    }
}
```

二、 說明

在高斯 - 喬登消去法中，若要求一 n 階方陣之反矩陣，需要先設定一個 n 階單位矩陣，在本作業中，方陣皆為四維，因此在 inverse 這個函數裡面，直接讓它產生一個四階單位矩陣，即對角項皆為 1，其餘為 0。在這邊由於稍後會直接用它做運算生成反矩陣，因此命名為 inverseT。

如同前面求行列式值部分提到的上三角矩陣，單位矩陣 I_n 也是一個 pivot 在越靠下越靠右的 (化簡列階梯型) 矩陣，且 rank 為 n。因此也要先判斷若 T_{ii} 為 0，要先透過列運算讓 T_{ii} 不為 0，才能做後續列運算使得 T 轉換 I_n。與上三角矩陣不同之處在於，這邊採用將 i+1 列加到 i 列的方式，是因為如果用列互換，指令會變多，

非但不會增加效率，也相對不直觀，如下所示：

```
for (int i = 0; i < 4; i++)
{
    if (T.element[i][i] == 0)
    {
        for (int j = 0; j < 4; j++)
        {
            double tmp = T.element[i][j];
            T.element[i][j] = T.element[i + 1][j];
            T.element[i + 1][j] = tmp;

            tmp = inverseT.element[i][j];
            inverseT.element[i][j] = inverseT.element[i + 1][j];
            inverseT.element[i + 1][j] = tmp;
        }

        elimination(T, inverseT, i);
    }
    else
    {
        elimination(T, inverseT, i);
    }
}
```

因為只要確保 T_{ii} 不為 0，後續就能藉由列運算將 T_{ii} 轉換成 1、其他項（無論是 T_{ij} 或 T_{ji}，$j \neq i$）化為 0，因此列運算過程及步驟並非唯一。

將 T 轉換成 I_n 分成兩部分：第一部份是對角項 T_{ii}，應該為 1 的位置，取 T_{ii} 為基數，將該整列除以基數，此時 T_{ii} 為 1，其他項透過第二部分進行消去；第二部分則是取 T_{ki} 為基數（k 表當前操作對象所在第 k 列），由於 T_{ii} 為 1，T_{ki} 與 T_{ii} 屬同一行不同列，才符合單位矩陣定義，因此 T_{ki} 在經過這輪消去後應為 0，可以得出：

$$T_{ki} = T_{ki} - 1 * T_{ki}$$

T_{ki} 即為第 i 列要對第 k 列作列運算時的基數（列運算「$cR_i+R_k \rightarrow R_k$」中的 c），此時再將 T_i 各項乘上基數後的值減去 T_k 各項，即可完成一輪列運算：將每一列（除了第 i 列）的第 i 行消去成 0。經過多輪列運算後，就能得到單位矩陣 I_n 與 T 的反矩陣。

需要注意的是，高斯 - 喬登消去法是藉由對擴增矩陣$[T|I_n]$做列運算得出$[I_n|T^{-1}]$，所以在 elimination 函式中可以觀察到，每個列運算都是一起對 T 和 inverseT 操作。