调研学习并给出矩阵的LU分解方法

背景及意义

矩阵的LU分解主要用来求解线性方程组或者计算行列式。

假设要求一个线性方程组，我们可以将其转换为AX=b的形式。其中A为方程组前的系数组成的矩阵，X，Y为两个向量。按照我们的高斯消元法，我们需要先将A转化为上三角矩阵，然后再进行求解，这样我们就可以得到X的值。

而LU分解是将矩阵A分解为L（下三角矩阵），U（上三角矩阵）的乘积。这样AX=Y即可变为LUX=Y，我们将UX看为Y，则有LY=b，我们可以求得Y，然后UX=Y，我们再求得X。

简单来说，LU分解将矩阵的分解和方程的求解过程分离。我们的目的是求解啊，这样做还要先拆再求不是多此一举？

其实不是这样，在不少应用场景中，当需要求解Ax = b的时候，左边的矩阵A很多时候是不变的，而右边的b会随着输入而变化。做LU分解的时候，只会用到A，所以可以预先准备好L和U，当有求解b的需求时，直接拿来用就可以。

将A分解为LU的时间复杂度为O(n^3)，而求解LUx = b只要O(n^2)。

分解方法

Doolittle分解

Doolittle分解可直接通过矩阵乘法导出计算过程。设$\boldsymbol{A}=\boldsymbol{LU}$，即

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a22 & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \cdots & \cdots & \cdots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \cdots & \cdots \\ & & & u_{nn} \end{bmatrix}$$

式（1.13）

第一步，应$l_{11}=1$,故$u_{1j} = a_{1j}(j=1,2,\ldots,n)$,且$a_{i1} = l_{i1}u_{11}$，则$l_{i1} = a_{i1}/u_{11}$，$(i=2,3,\ldots,n)$，由此计算出$U$的第$1$行及$L$的第$1$列元素。

第i步，由

$$a_{ij}=\sum_{k=1}^{i}l_{ik}u_{kj}=\sum_{k=1}^{i-1}l_{ik}u_{kj}+u_{ij},j=i,i+1,\ldots,n$$

式（1.14）

得

$$u_{ij}=a_{ij}-\sum_{k=1}^{i-1}l_{ik}u_{kj},j=i,i+1,\ldots,n$$

式（1.15）

又由

$$a_{ji}=\sum_{k=1}^{i}l_{jk}u_{ki}=\sum_{k=1}^{i-1}l_{jk}u_{ki}+l_{ji}u_{ii},j=i+1,i+2,\ldots,n$$

式（1.16）

得

$$l_{ji}=(a_{ji}-\sum_{k=1}^{i-1}l_{jk}u_{ki})/u_{ii},j=i+1,i+2,\ldots,n$$

式（1.17）

最后化简：

对$i=1,2,\ldots,n$

$$\begin{cases} a_{ij}\leftarrow u_{ij}=a_{ij}-\sum_{k=1}^{i-1}l_{ik}u_{kj},j=i,i+1,\ldots,n \\ a_{ji}\leftarrow l_{ji}=(a_{ji}-\sum_{k=1}^{i-1}l_{jk}u_{ki})/u_{ii},j=i+1,i+2,\ldots,n \end{cases}$$

式（1.18）

看似复杂的式子，其实就是将化为上三角的过程公式化得到U，将消去对角线下元素的过程中乘以的系数的相反数合成L，就实现了杜立特尔分解。

```
#include <iostream>
#include <vector>

using namespace std;

void doolittle(vector<vector<double>>& A, vector<vector<double>>& L,
vector<vector<double>>& U) {
    int n = A.size();

    for (int i = 0; i < n; i++) {
        // U矩阵的第一行就是A矩阵的第一行
        U[0][i] = A[0][i];

        // L矩阵的第一列是A矩阵第一列除以U矩阵的(1,1)元素
        L[i][0] = A[i][0] / U[0][0];
    }

    // 构造U矩阵和L矩阵
    for (int i = 1; i < n; i++) {
        for (int j = i; j < n; j++) {
            double sum = 0;
            for (int k = 0; k < i; k++) {
                sum += L[i][k] * U[k][j];
```

```cpp
            }

            U[i][j] = A[i][j] - sum;
        }

        for (int j = i + 1; j < n; j++) {
            double sum = 0;
            for (int k = 0; k < i; k++) {
                sum += L[j][k] * U[k][i];
            }

            L[j][i] = (A[j][i] - sum) / U[i][i];
        }
    }
}

int main() {
    // 示例矩阵
    vector<vector<double>> A = {{4, 3, 1},
                                {2, 1, 2},
                                {1, 3, 9}};

    int n = A.size();

    // 初始化L和U矩阵
    vector<vector<double>> L(n, vector<double>(n, 0));
    vector<vector<double>> U(n, vector<double>(n, 0));

    doolittle(A, L, U);

    // 打印L和U矩阵
    cout << "L matrix:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << L[i][j] << " ";
        }
        cout << endl;
    }

    cout << "U matrix:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << U[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Crout 分解

我们已经学会了上述Doolittle分解的方法

为了引出Crout分解，我们先提一下LDU分解

LDU分解其实很简单，只需要将LU分解简单变化即可。我们知道U是一个上三角矩阵，我们将其对角元元素提取出来形成一个对角矩阵D，则剩下的新的U和原来的L，其三者就组成了LDU分解。

矩阵的 *LDU* 分解：

$$\begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & 3 & 1 & \\ 3 & 4 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & & & \\ & 1 & & \\ & & 2 & \\ & & & 2 \end{pmatrix} \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{2} & 0 \\ & 1 & 1 & 1 \\ & & 1 & 1 \\ & & & 1 \end{pmatrix}.$$

知乎 @乔胤博

我们将左侧的两个矩阵的乘积合为一个矩阵，我们就得到了Crout分解。

$$\begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 2 & & & \\ 4 & 1 & & \\ 8 & 3 & 2 & \\ 6 & 4 & 2 & 2 \end{pmatrix} \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{2} & 0 \\ & 1 & 1 & 1 \\ & & 1 & 1 \\ & & & 1 \end{pmatrix}.$$

知乎 @乔胤博

用公式去表示整个分解过程为

矩阵$\boldsymbol{A} = [a_{ij}]_{n\times n}$的Crout分解：

$$\begin{cases} \text{对于}k = 1,2,\cdots,n\text{计算} \\ l_{ik} = a_{ik} - \sum\limits_{t=1}^{k-1} l_{it}u_{tk} \qquad (i = k, k+1, \cdots, n) \\ u_{kj} = \dfrac{(a_{kj} - \sum_{t=1}^{k-1} l_{kt}u_{yj})}{l_{kk}} \qquad (j = k+1, k+2, \cdots, n; k < n) \end{cases} \tag{6}$$

```
vector<vector<vector<double>>> crout(vector<vector<double>> A) {
    int n = A.size();
    vector<vector<double>> L(n, vector<double>(n, 0));
    vector<vector<double>> U(n, vector<double>(n, 0));

    // 初始化对角元素
    for (int i = 0; i < n; i++) {
        L[i][i] = 1;
    }

    // 计算L和U的元素
    for (int j = 0; j < n; j++) {
        for (int i = j; i < n; i++) {
```

```
            double sum = 0;
            for (int k = 0; k < j; k++) {
                sum += L[i][k] * U[k][j];
            }
            U[i][j] = A[i][j] - sum;
        }
        for (int i = j + 1; i < n; i++) {
            double sum = 0;
            for (int k = 0; k < j; k++) {
                sum += L[j][k] * U[k][i];
            }
            L[i][j] = (A[i][j] - sum) / U[j][j];
        }
    }

    // 返回L和U矩阵
    vector<vector<vector<double>>> result;
    result.push_back(L);
    result.push_back(U);
    return result;
}
```

给出方案计算可逆矩阵的逆。

方法1 伴随矩阵求逆

伴随矩阵是矩阵元素所对应的代数余子式，所构成的矩阵，转置后得到的新矩阵。

将该伴随矩阵除以行列式就可以得到矩阵的逆。

伴随矩阵法： $A=\begin{bmatrix}1 & 2\\ -1 & -3\end{bmatrix}$

① 计算余子式矩阵：

$M_{11}=|-3|=-3$　$M_{12}=|-1|=-1$　$M_{21}=|2|=2$

$M_{22}=|1|=1 \Rightarrow M=\begin{bmatrix}-3 & -1\\ 2 & 1\end{bmatrix}$

② 改变M的符号，生成代数余子式矩阵C

$C=\begin{bmatrix}-3 & -1\\ 2 & 1\end{bmatrix}+\begin{bmatrix}+ & -\\ - & +\end{bmatrix}=\begin{bmatrix}-3 & 1\\ -2 & 1\end{bmatrix}$

③ 转置C，得到伴随矩阵A^*。

$A^*=C^{-1}=\begin{bmatrix}-3 & -2\\ 1 & 1\end{bmatrix}$

④ 计算矩阵A的行列式 det(A)

$det(A)=\begin{vmatrix}1 & 2\\ -1 & -3\end{vmatrix}=-3-(-2)=-3+2=-1$

⑤ 用伴随矩阵A^*除以det(A)，得到矩阵A的逆A^{-1}。

$A^{-1}=A^*/det(A)=\begin{bmatrix}-3 & -2\\ 1 & 1\end{bmatrix}/-1=\begin{bmatrix}3 & 2\\ -1 & -1\end{bmatrix}$

CSDN @松下J27

这种方法十分复杂，运算量大，容易出错。

代码

```
#include <iostream>
#include <ctime>     //用于产生随机数据的种子

#define N 3    //测试矩阵维数定义
using namespace std;
//按第一行展开计算|A|
double getA(double arcs[N][N], int n)
{
    if (n == 1)
    {
        return arcs[0][0];
    }
    double ans = 0;
    double temp[N][N] = { 0.0 };
    int i, j, k;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - 1; j++)
        {
            for (k = 0; k < n - 1; k++)
```

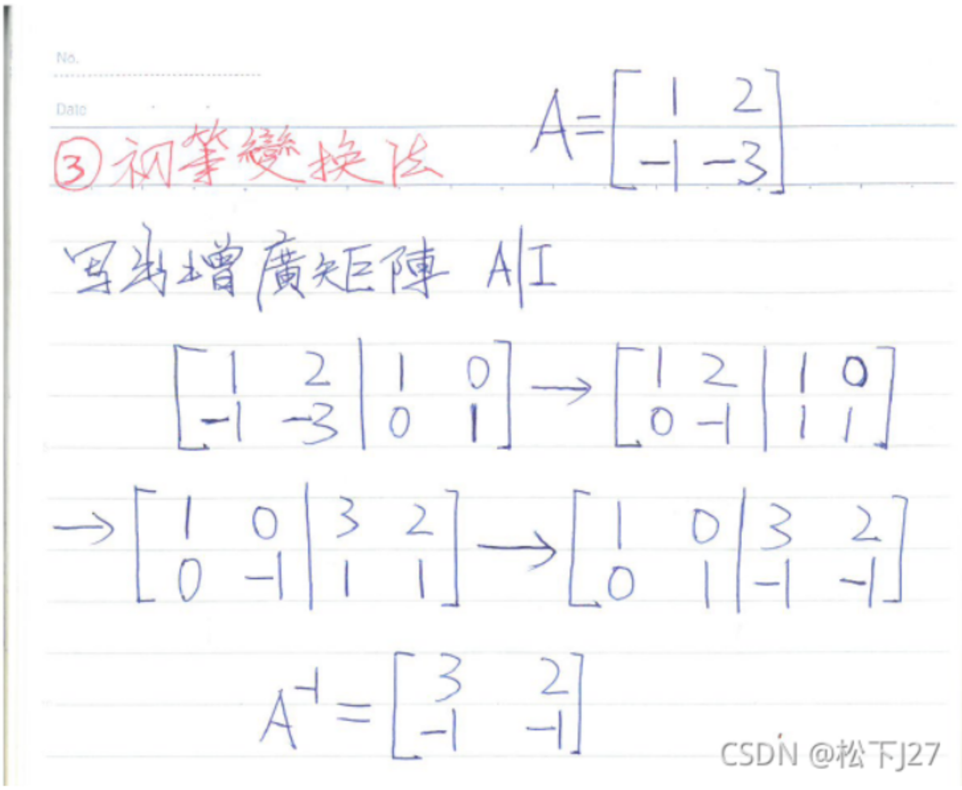
```
            {
                temp[j][k] = arcs[j + 1][(k >= i) ? k + 1 : k];

            }
        }
        double t = getA(temp, n - 1);
        if (i % 2 == 0)
        {
            ans += arcs[0][i] * t;
        }
        else
        {
            ans -= arcs[0][i] * t;
        }
    }
    return ans;
}

//计算每一行每一列的每个元素所对应的余子式，组成A*
void  getAStart(double arcs[N][N], int n, double ans[N][N])
{
    if (n == 1)
    {
        ans[0][0] = 1;
        return;
    }
    int i, j, k, t;
    double temp[N][N];
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            for (k = 0; k < n - 1; k++)
            {
                for (t = 0; t < n - 1; t++)
                {
                    temp[k][t] = arcs[k >= i ? k + 1 : k][t >= j ? t + 1 : t];
                }
            }


            ans[j][i] = getA(temp, n - 1);  //此处顺便进行了转置
            if ((i + j) % 2 == 1)
            {
                ans[j][i] = -ans[j][i];
            }
        }
    }
}

//得到给定矩阵src的逆矩阵保存到des中。
bool GetMatrixInverse(double src[N][N], int n, double des[N][N])
{
    double flag = getA(src, n);
    double t[N][N];
    if (0 == flag)
    {
        std::cout << "原矩阵行列式为0，无法求逆。请重新运行" << std::endl;
```

```
        return false;//如果算出矩阵的行列式为0，则不往下进行
    }
    else
    {
        getAStart(src, n, t);
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                des[i][j] = t[i][j] / flag;
            }

        }
    }

    return true;
}

int main()
{
    bool flag;//标志位，如果行列式为0，则结束程序
    int row = N;
    int col = N;
    double matrix_before[N][N]{};//{1,2,3,4,5,6,7,8,9};

    //随机数据，可替换
    srand((unsigned)time(0));
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            matrix_before[i][j] = rand() % 100 * 0.01;
        }
    }

    cout << "原矩阵：" << endl;

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            cout << *(*(matrix_before + i) + j) << " ";
        }
        cout << endl;
    }

    double matrix_after[N][N]{};
    flag = GetMatrixInverse(matrix_before, N, matrix_after);
    if (false == flag)
        return 0;

    cout << "逆矩阵：" << endl;

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cout << matrix_after[i][j] << " ";
```

```
        }
        cout << endl;
    }

    GetMatrixInverse(matrix_after, N, matrix_before);

    cout << "反算的原矩阵：" << endl;

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            cout << *(*(matrix_before + i) + j) << " ";
        }
        cout << endl;
    }

    return 0;
}
```

高斯消元法

已知矩阵A和对应维度的单位矩阵I，先写出增广矩阵A|I，然后对A进行高斯消元，此时I也在跟着变化。当A变为一个单位矩阵时，此时右侧的I就变为了逆矩阵。



代码

```
#include <iostream>
using namespace std;

const int N = 3; // 矩阵的维度

// 求矩阵A的逆矩阵
void inverse(double A[][N], double B[][N]) {
```

```
    // 初始化为单位矩阵
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            B[i][j] = (i == j) ? 1.0 : 0.0;
        }
    }

    // 高斯消元
    for (int k = 0; k < N; k++) {
        // 如果A[k][k]为0，则需要进行行交换，找到一个非零元素
        if (A[k][k] == 0) {
            for (int i = k + 1; i < N; i++) {
                if (A[i][k] != 0) {
                    // 交换第k行和第i行
                    for (int j = 0; j < N; j++) {
                        swap(A[k][j], A[i][j]);
                        swap(B[k][j], B[i][j]);
                    }
                    break;
                }
            }
        }
        // 如果无法找到非零元素，则矩阵A为奇异矩阵，无法求逆
        if (A[k][k] == 0) {
            cerr << "ERROR: singular matrix\n";
            return;
        }
        // 将第k行消成主元为1
        double f = A[k][k];
        for (int j = 0; j < N; j++) {
            A[k][j] /= f;
            B[k][j] /= f;
        }
        // 消元
        for (int i = 0; i < N; i++) {
            if (i == k) continue;
            f = A[i][k];
            for (int j = 0; j < N; j++) {
                A[i][j] -= f * A[k][j];
                B[i][j] -= f * B[k][j];
            }
        }
    }
}

int main() {
    double A[N][N] = {{2, 1, 3}, {0, 2, 1}, {1, 1, 1}}; // 原矩阵
    double B[N][N]; // 逆矩阵

    inverse(A, B);

    // 输出逆矩阵
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            cout << B[i][j] << " ";
        }
        cout << endl;
    }
```

```
    return 0;
}
```

LU分解法

我们已经将A分解为LU矩阵了，我们可以利用LU来实现矩阵的求逆。

具体做法为：

1. 对A进行LU分解，得到L和U。
2. 对于单位矩阵I，我们可以将其视为n个列向量的组合，即I=[e1, e2, ..., en]，其中ei是一个n维列向量，其第i个元素为1，其余元素为0。
3. 对于每个ei，我们可以使用LU分解的结果L和U来求解Bi= A^-1ei。具体来说，我们可以使用以下步骤来求解Bi：
   a. 解出下三角矩阵L的方程Ly=ei，得到y。
   b. 解出上三角矩阵U的方程Ux=y，得到x，即Bi。
4. 最终的逆矩阵A^-1就是由Bi组成的矩阵，即A^-1=[B1, B2, ..., Bn]。

用LU分解法求矩陣的逆

$$A^{-1}A = I = AA^{-1} \quad ① \qquad A = LU \quad ②$$

$$AA^{-1} = LUA^{-1} = I \quad ③$$

利用矩陣的乘法的定义:

$$\underset{A}{[L\ U]}\ \underset{A^{-1}}{[A_1^{-1}\ A_2^{-1} \cdots A_n^{-1}]} = \underset{I}{[I_1\ I_2 \cdots I_n]}$$

$$\begin{cases} [LU]A_1^{-1} = I_1 \Leftrightarrow Ax_1 = b_1 \\ [LU]A_2^{-1} = I_2 \Leftrightarrow Ax_2 = b_2 \\ \quad \vdots \\ [LU]A_n^{-1} = I_n \Leftrightarrow Ax_n = b_n \end{cases}$$