原 高斯消元(and牛顿迭代法)详解及模板

2018年07月26日 15:37:18 bestsort 阅读数 1113

高斯消元的最主要作用为求解线性方程组，说白了，就是解方程......
解方程还有一种为牛顿迭代法，我们每次枚举一个值X_0，代入方程看是否为根，不是的话则将X_0的值变为：
$X_0 = X_0 - F(X_0)/F'(X_0)$（其中$F'(x)$ 为 $F(x)$的导数）
其证明过程借用了高数中的基本知识泰勒级数，此处不再赘述。
有趣的是，在 雷神之锤III 中,卡马克用牛顿迭代法,令$X_0 = 0x5f3759df$后求出数字开根后的倒数的速度居然比库函数快4倍!而这个数字也成为了神奇

消元的过程就是模拟人手算解方程的过程

其算法过程就2个

1. 代入消元
2. 将算出的值代回，最终求得所有未知数的解

$$\begin{cases} x+2y+z=7 & A \\ 2x-y+3z=7 & B \\ 3x+y+2z=18 & C \end{cases}$$

通俗点，比如说对于这个方程组
第一个过程是将其中的一个变量消元化为 `x=num || y=num || z=num` 的形式，其中 `num` 为常数
第二个过程就是将这个已经求出来的 `x || y || z` 带回到原方程组中求出剩余两个未知数点值;

这里我们用矩阵模拟这个消元回代的过程
还是用上面的那个方程做例子

$$\begin{cases} x+2y+z=7 & A \\ 2x-y+3z=7 & B \\ 3x+y+2z=18 & C \end{cases}$$

转化为矩阵后为:

$$\begin{vmatrix} 1 & 2 & 1 & 7 \\ 2 & -1 & 3 & 7 \\ 3 & 1 & 2 & 18 \end{vmatrix}$$

其中最后一列为等号右边的值，设从上到下每行依次为 `R1,R2,R3` ,从左到右每一列依次为 `x,y,z` 的系数;

需要注意的是这里我们要把x最大的放在第一行，这里是为了在进行除法点时候减小误差
现在开始化简:

$$\begin{vmatrix} 3 & 1 & 2 & 18 \\ 0 & -5 & 5 & -15 \\ 1 & 2 & 1 & 7 \end{vmatrix}$$

3*R2-2*R1 ->

$$\begin{vmatrix} 3 & 1 & 2 & 18 \\ 0 & -5 & 5 & -15 \\ 0 & 5 & 1 & 3 \end{vmatrix}$$

3*R3-R1->

$$\begin{vmatrix} 3 & 1 & 2 & 18 \\ 0 & -5 & 5 & -15 \\ 0 & 0 & -6 & 12 \end{vmatrix}$$

R3*(-1)-R2->
到这里已经可消元完毕了，对应式子如下:

$$\begin{cases} 3x + y + 2z = 18 \\ -5y + 5z = -15 \\ -6z = 12 \end{cases}$$

然后就是回代求解了，这里就不细说了，解方程大家都会

以下模板来自kuangbin

```
#include<stdio.h>
#include<algorithm>
#include<iostream>
#include<string.h>
#include<math.h>
using namespace std;
const int MAXN=50;
int a[MAXN][MAXN];//增广矩阵
int x[MAXN];//解集
bool free_x[MAXN];//标记是否是不确定的变元
inline int gcd(int a,int b)
{
    int t;
    while(b!=0)
    {
        t=b;
        b=a%b;
        a=t;
    }
    return a;
}
inline int lcm(int a,int b)
{
    return a/gcd(a,b)*b;//先除后乘防溢出
}

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2表示有浮点数解，但无整数解，
//-1表示无解，0表示唯一解，大于0表示无穷解，并返回自由变元的个数)
//有equ个方程，var个变元。增广矩阵行数为equ,分别为0到equ-1,列数为var+1,分别为0到var.
int Gauss(int equ,int var)
{
    int i,j,k;
    int max_r;// 当前这列绝对值最大的行.
    int col;//当前处理的列
    int ta,tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;

    for(int i=0; i<=var; i++)
    {
        x[i]=0;
        free_x[i]=true;
    }

    //转换为阶梯阵.
    col=0; // 当前处理的列
    for(k = 0; k < equ && col < var; k++,col++)
    {
        // 枚举当前处理的行.
// 找到该col列元素绝对值最大的那行与第k行交换.(为了在除法时减小误差)
        max_r=k;
        for(i=k+1; i<equ; i++)
        {
            if(abs(a[i][col])>abs(a[max_r][col])) max_r=i;
        }
        if(max_r!=k)
        {
```

```
            for(j=k; j<var+1; j++) swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col]==0)
        {
            // 说明该col列第k行以下全是0了，则处理当前行的下一列.
            k--;
            continue;
        }
        for(i=k+1; i<equ; i++)
        {
            // 枚举要删去的行.
            if(a[i][col]!=0)
            {
                LCM = lcm(abs(a[i][col]),abs(a[k][col]));
                ta = LCM/abs(a[i][col]);
                tb = LCM/abs(a[k][col]);
                if(a[i][col]*a[k][col]<0)tb=-tb;     //异号的情况是相加
                for(j=col; j<var+1; j++)
                {
                    a[i][j] = a[i][j]*ta-a[k][j]*tb;
                }
            }
        }
    }

    // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
    for (i = k; i < equ; i++)
    {
        // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
        if (a[i][col] != 0) return -1;
    }
    // 2. 无穷解的情况: 在var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.
    // 且出现的行数即为自由变元的个数.
    if (k < var)
    {
        // 首先，自由变元有var - k个，即不确定的变元至少有var - k个.
        for (i = k - 1; i >= 0; i--)
        {
            // 第i行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第k行到第equ行.
            // 同样，第i行一定不会是(0, 0, ..., a), a != 0的情况，这样的无解的.
            free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过1个，则无法求解，它们仍然为不确定的变元.
            for (j = 0; j < var; j++)
            {
                if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
            }
            if (free_x_num > 1) continue; // 无法求解出确定的变元.
            // 说明就只有一个不确定的变元free_index，那么可以求解出该变元，且该变元是确定的.
            temp = a[i][var];
            for (j = 0; j < var; j++)
            {
                if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
            }
            x[free_index] = temp / a[i][free_index]; // 求出该变元.
            free_x[free_index] = 0; // 该变元是确定的.
        }
        return var - k; // 自由变元有var - k个.
    }
    // 3. 唯一解的情况: 在var * (var + 1)的增广阵中形成严格的上三角阵.
    // 计算出Xn-1, Xn-2 ... X0.
    for (i = var - 1; i >= 0; i--)
    {
        temp = a[i][var];
        for (j = i + 1; j < var; j++)
        {
            if (a[i][j] != 0) temp -= a[i][j] * x[j];     //--因为x[i]存的是temp/a[i][i]的值，即是a[i][i]=1时x[i]对应的值
        }
        if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
        x[i] = temp / a[i][i];
```

```
    return 0;
}
int main(void)
{
    freopen("in.txt", "r", stdin);
    freopen("out.txt","w",stdout);
    int i, j;
    int equ,var;
    while (scanf("%d %d", &equ, &var) != EOF)
    {
        memset(a, 0, sizeof(a));
        for (i = 0; i < equ; i++)
        {
            for (j = 0; j < var + 1; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }
        int free_num = Gauss(equ,var);
        if (free_num == -1) printf("无解!\n");
        else if (free_num == -2) printf("有浮点数解，无整数解!\n");
        else if (free_num > 0)
        {
            printf("无穷多解! 自由变元个数为%d\n", free_num);
            for (i = 0; i < var; i++)
            {
                if (free_x[i]) printf("x%d 是不确定的\n", i + 1);
                else printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        else
        {
            for (i = 0; i < var; i++)
            {
                printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        printf("\n");
    }
    return 0;
}
```

转载注明出处，谢谢合作