线性规划之单纯形法【超详解+图解】

1.作用

单纯形法是解决线性规划问题的一个有效的**算法**。线性规划就是在一组线性约束条件下，求解目标函数最优解的问题。

2.线性规划的一般形式

在约束条件下，寻找目标函数z的最大值。

$$\max z = x_1 + x_2$$
$$s.t. \begin{cases} 2x_1 + x_2 \leq 12 \\ x_1 + 2x_2 \leq 9 \\ x_1, x_2 \geq 0 \end{cases}$$

3.线性规划的可行域

满足线性规划问题约束条件的所有点组成的集合就是线性规划的可行域。若可行域有界（以下主要考虑有界可行域），线性规划问题的目标函数最优解必然在可行域的顶点上达到最优。

单纯形法就是通过设置不同的基向量，经过矩阵的线性变换，求得基可行解（可行域顶点），并判断该解是否最优，否则继续设置另一组基向量，重复执行以上步骤，直到找到最优解。所以，单纯形法的求解过程是一个循环迭代的过程。

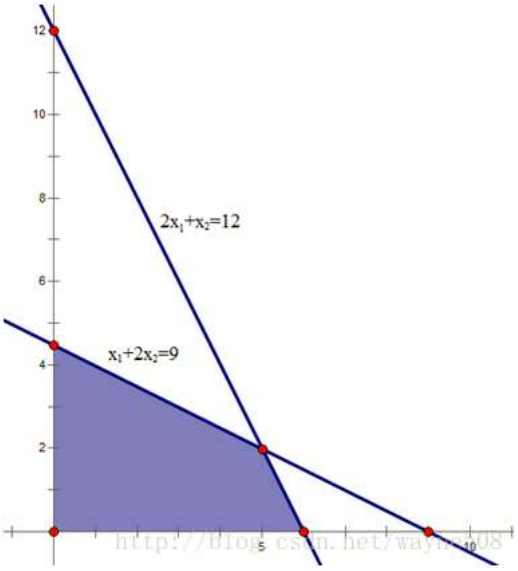


图1 可行域

4.线性规划的标准形式

在说明单纯形法的原理之前，需要明白线性规划的标准形式。因为单纯形算法是通过线性规划的标准形来求解的。一般，规定线性规划的标准形式为：

$$\max z = \sum_{j=1}^{n} c_j x_j$$
$$s.t. \begin{cases} \sum_{j=1}^{n} a_{ij} x_j = b_j, i = 1, 2, ..., m \\ xj \geq 0, j = 1, 2, ..., n \end{cases}$$

写成矩阵形式：

$$\max z = CX$$
$$AX = b$$
$$X \geq 0$$
$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

标准形的形式为：

1）目标函数要求max

2）约束条件均为等式

3）决策变量为非负约束

普通线性规划化为标准形：

1）若目标函数为最小化，可以通过取负，求最大化

2）约束不等式为小于等于不等式，可以在左端加入非负松弛变量，转变为等式，比如：

$$x_1 + 2x_2 \leq 9 \Rightarrow \begin{cases} x_1 + 2x_2 + x_3 = 9 \\ x_3 \geq 0 \end{cases}$$

同理，约束不等式为大于等于不等式时，可以在左端减去一个非负松弛变量，变为等式。

3）若存在取值无约束的变量，可转变为两个非负变量的差，比如：

$$-\infty \leq x_k \leq +\infty \Rightarrow \begin{cases} x_k = x_m - x_n \\ x_m, x_n \geq 0 \end{cases}$$

本文最开始的线性规划问题转化为标准形为：

$$\max z = x_1 + x_2$$
$$s.t. \begin{cases} 2x_1 + x_2 + x_3 = 12 \\ x_1 + 2x_2 + x_4 = 9 \\ x_1, x_2, x_3, x_4 \geq 0 \end{cases}$$

5.单纯形法

5.1几何意义

在标准形中，有m个约束条件（不包括非负约束），n个决策变量，且（n>=m）。首先，选取m个基变量 $x'_j(j = 1, 2, ..., m)$ ，基变量对应约束系数矩阵的列向量线性无关。通过矩阵的线性变换，基变量可由非基变量表示：

$$x'_i = C_i + \sum_{j=m+1}^{n} m_{ij} x'_j (i = 1, 2, ..., m)$$

如果令非基变量等于0，可求得基变量的值 ：

$$x'_i = C_i$$

如果为可行解的话，Ci大于0。那么它的几何意义是什么呢？

还是通过上述具体的线性规划问题来说明。

$$\max z = x_1 + x_2$$
$$s.t. \begin{cases} 2x_1 + x_2 + x_3 = 12 \\ x_1 + 2x_2 + x_4 = 9 \\ x_1, x_2, x_3, x_4 \geq 0 \end{cases}$$

如果选择x2、x3为基变量，那么令x1、x4等于0，可以去求解基变量x2、x3的值。对系数矩阵做行变换，如下所示，x2=9/2，x3=15/2

$$\begin{bmatrix} \mathrm{X} & x_1 & x_2 & x_3 & x_4 & b \\ & 2 & 1 & 1 & 0 & 12 \\ & 1 & 2 & 0 & 1 & 9 \\ \mathrm{C} & 1 & 1 & 0 & 0 & z \end{bmatrix} \rightarrow \begin{bmatrix} \mathrm{X} & x_1 & x_2 & x_3 & x_4 & b \\ & \frac{3}{2} & 0 & 1 & -\frac{1}{2} & \frac{15}{2} \\ & \frac{1}{2} & 1 & 0 & \frac{1}{2} & \frac{9}{2} \\ \mathrm{C} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & z - \frac{9}{2} \end{bmatrix}$$

X1=0表示可行解在x轴上；X4=0表示可行解在x1+2x2=9的直线上。那么，求得的可行解即表示这两条直线的交点，也是可行域的顶点，如图所示：

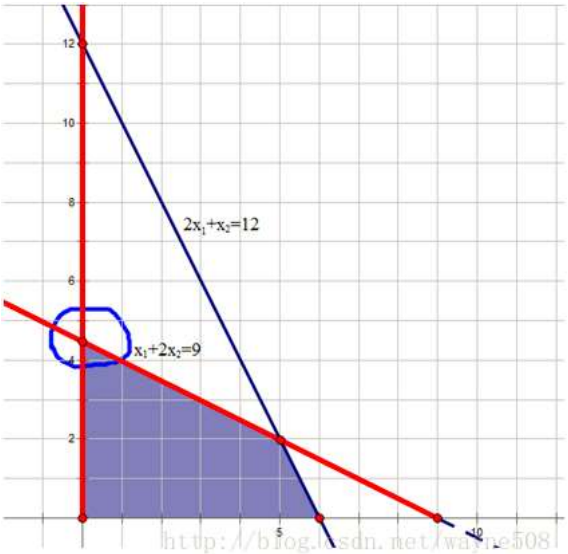


图2

所以，通过选择不同的基变量，可以获得不同的可行域的顶点。

5.2如何判断最优

如前所述，基变量可由非基变量表示：

$$x_i' = C_i + \sum_{j=m+1}^{n} m_{ij} x_j' (i = 1, 2, ..., m)$$

目标函数z也可以完全由非基变量表示：

$$z = z_0 + \sum_{j=m+1}^{n} \sigma_j x_j'$$

当达到最优解时，所有的 σ_j 应小于等于0。当存在j， σ_j >0时，当前解不是最优解，为什么？

当前的目标函数值为z0，其中所有的非基变量值均取0。由之前分析可知， x_j' =0代表可行域的某个边界，是 x_j' 的最小值。如果可行解逐步离开这个边界， x_j' 会变大，因为 σ_j >0，显然目标函数的取值也会变大，所以当前解不是最优解。我们需要寻找新的基变量。

5.3如何选择新的基变量

如果存在多个 σ_j >0，选择最大的 σ_j >0对应的变量作为基变量，这表示目标函数随着 x_j' 的增加，增长的最快。

5.4如何选择被替换的基变量

假如我们选择非基变量 x_s' 作为下一轮的基变量，那么被替换基变量 x_j' 在下一轮中作为非基变量，等于0。选择 x_j' 的原则：替换后应该尽量使 x_s' 值最大（因为上面已分析过，目标函数会随着 x_s' 的增大而增大）。

继续通过上面的例子来说明：

$$\begin{bmatrix} \mathrm{X} & x_1 & x_2 & x_3 & x_4 & b \\ & 2 & 1 & 1 & 0 & 12 \\ & 1 & 2 & 0 & 1 & 9 \\ \mathrm{C} & 1 & 1 & 0 & 0 & z \end{bmatrix} \rightarrow \begin{bmatrix} \mathrm{X} & x_1 & x_2 & x_3 & x_4 & b \\ & \frac{3}{2} & 0 & 1 & -\frac{1}{2} & \frac{15}{2} \\ & \frac{1}{2} & 1 & 0 & \frac{1}{2} & \frac{9}{2} \\ \mathrm{C} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & z - \frac{9}{2} \end{bmatrix}$$

从最后一行可以看到，x1的系数为1/2>0，所以选x2、x3为基变量并没有是目标函数达到最优。下一轮选取x1作为基变量，替换x2、x3中的某个变量。

第一行是符号

第二行：若x1替换x3作为基变量，x3=0时，x1=(15/2)/(3/2)=5

第三行：若x1替换x2作为基变量，x2=0时，x1=(9/2)/(1/2)=9

显然，应该把x2作为非基变量。

5.5终止条件

当目标函数用非基变量的线性组合表示时，所有的系数均不大于0，则表示目标函数达到最优。

如果，有一个非基变量的系数为0，其他的均小于0，表示目标函数的最优解有无穷多个。这是因为目标函数的梯度与某一边界正交，在这个边界上，目标函数的取值均相等，且为最优。

使用单纯型法来求解线性规划，输入单纯型法的松弛形式，是一个大矩阵，第一行为目标函数的系数，且最后一个数字为当前轴值下的 z 值。下面每一行代表一个约束，数字代表系数每行最后一个数字代表 b 值。

算法和使用单纯性表求解线性规划相同。

对于线性规划问题：

Max x1 + 14* x2 + 6*x3

s . t . x1 + x2 + x3 <= 4

x1 *<= 2*

x3 <= 3

*3*x2 + x3 <= 6*

x1,x2,x3 >= 0

我们可以得到其松弛形式：

Max x1 + 14*x2 + 6*x3

s.t. x1 + x2 + x3 + x4 = 4

x1 + x5 = 2

x3 + x6 = 3

3*x2 + x3 + x7 = 6

x1 , x2 , x3 , x4 , x5 , x6 , x7 ≥ 0

我们可以构造单纯性表，其中最后一行打星的列为轴值。

单纯性表

x1	x2	x3	x4	x5	x6	x7	b
c1=1	c2=14	c3=6	c4=0	c5=0	c6=0	c7=0	-z=0
1	1	1	1	0	0	0	4
1	0	0	0	1	0	0	2
0	0	1	0	0	1	0	3
0	3	1	0	0	0	1	6
			*	*	*	*	

在单纯性表中，我们发现非轴值的x上的系数大于零，因此可以通过增加这些个x的值，来使目标函数增加。我们可以贪心的选择最大的c，再上面的例子中我们选择c2作为新的轴，加入轴集合中，那么谁该出轴呢？

其实我们由于每个x都大于零，对于x2它的增加是有所限制的，如果x2过大，由于其他的限制条件，就会使得其他的x小于零，于是我们应该让x2一直增大，直到有一个其他的x刚好等于0为止，那么这个x就被换出轴。

我们可以发现，对于约束方程1，即第一行约束，x2最大可以为4（4/1），对于约束方程4，x2最大可以为3(6/3），因此x2最大只能为他们之间最小的那个，这样才能保证每个x都大于零。因此使用第4行，来对各行进行高斯行变换，使得二列第四行中的每个x都变成零，也包括c2。这样我们就完成了把x2入轴，x7出轴的过程。变换后的单纯性表为：

单纯性表

x1	x2	x3	x4	x5	x6	x7	b
c1=1	c2=0	c3=1.33	c4=0	c5=0	c6=0	c7=-4.67	-z=-28
1	0	0.67	1	0	0	-0.33	2
1	0	0	0	1	0	0	2
0	0	1	0	0	1	0	3

0	1	0.33	0	0	0	0.33	2
	*		*	*	*		

继续计算，我们得到：

单纯性表

x1	x2	x3	x4	x5	x6	x7	b
c1=-1	c2=0	c3=0	c4=0	c5=-2	c6=0	c7=0	-z=-32
1.5	0	1	1.5	0	0	-0.5	3
1	0	0	0	1	0	0	2
0	0	1	0	0	1	0	3
0	1	0.33	0	0	0	0.33	2
	*		*	*	*		

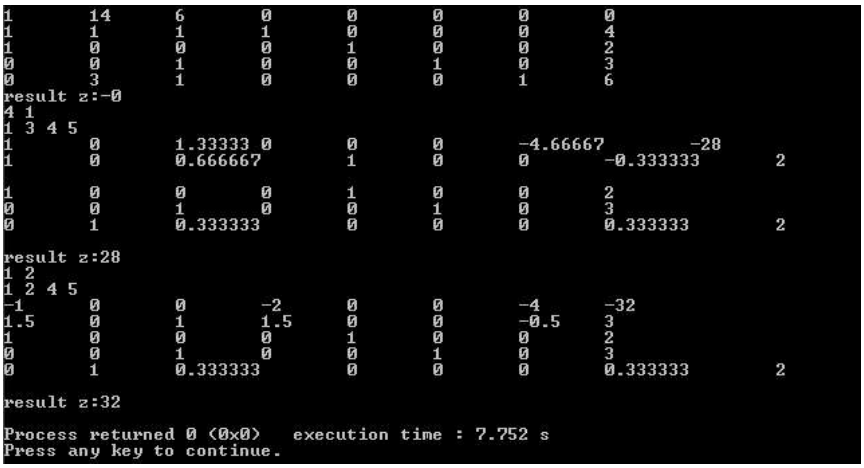
此时我们发现，所有非轴的x的系数全部小于零，即增大任何非轴的x值并不能使得目标函数最大，从而得到最优解32.

整个过程代码如下所示：

```
#include <bits/stdc++.h>
using namespace std;
vector<vector<double> > Matrix;
double Z;
set<int> P;
size_t cn, bn;

bool Pivot(pair<size_t, size_t> &p)//返回0表示所有的非轴元素都小于0
{
    int x = 0, y = 0;
    double cmax = -INT_MAX;
    vector<double> C = Matrix[0];
    vector<double> B;

    for( size_t i = 0 ; i < bn ; i++ )
    {
        B.push_back(Matrix[i][cn-1]);
    }

    for( size_t i = 0 ; i < C.size(); i++ )//在非轴元素中找最大的c
    {
        if( cmax < C[i] && P.find(i) == P.end())
        {
            cmax = C[i];
            y = i;
        }
    }
    if( cmax < 0 )
    {
        return 0;
    }

    double bmin = INT_MAX;
    for( size_t i = 1 ; i < bn ; i++ )
    {
        double tmp = B[i]/Matrix[i][y];
       if( Matrix[i][y] != 0 && bmin > tmp )
       {
           bmin = tmp;
           x = i;
       }
    }

    p = make_pair(x, y);

    for( set<int>::iterator it = P.begin() ; it != P.end() ; it++)
    {
        if( Matrix[x][*it] != 0 )
        {
            //cout<<"erase "<<*it<<endl;
            P.erase(*it);
            break;
        }
    }
    P.insert(y);
    //cout<<"add "<<y<<endl;
    return true;
}

void pnt()
{
```

```
    for( size_t i = 0 ; i < Matrix.size() ; i++ )
    {
        for( size_t j = 0 ; j < Matrix[0].size() ; j++ )
        {
            cout<<Matrix[i][j]<<"\t";
        }
    cout<<endl;
    }
    cout<<"result z:"<<-Matrix[0][cn-1]<<endl;
}

void Gaussian(pair<size_t, size_t> p)//行变换
{
    size_t  x = p.first;
    size_t y = p.second;
    double norm = Matrix[x][y];
    for( size_t i = 0 ; i < cn ; i++ )//主行归一化
    {
        Matrix[x][i] /= norm;
    }
    for( size_t i = 0 ; i < bn && i != x; i++ )
    {
        if( Matrix[i][y] != 0)
        {
            double tmpnorm = Matrix[i][y];
            for( size_t j = 0 ; j < cn ; j++ )
            {
                Matrix[i][j] = Matrix[i][j] - tmpnorm * Matrix[x][j];
            }
        }
    }
}

void solve()
{
    pair<size_t, size_t> t;
    while(1)
    {

        pnt();
        if( Pivot(t) == 0 )
        {
            return;
        }
        cout<<t.first<<" "<<t.second<<endl;
        for( set<int>::iterator it = P.begin(); it != P.end()  ; it++ )
        {
            cout<<*it<<" ";
        }
        cout<<endl;
        Gaussian(t);
    }
}

int main(int argc, char *argv[])
{
    //ifstream fin;
    //fin.open("./test");
    cin>>cn>>bn;
    for( size_t i = 0 ; i < bn ; i++ )
    {
        vector<double> vectmp;
        for( size_t j = 0 ; j < cn ; j++)
        {
            double tmp = 0;
            cin>>tmp;
            vectmp.push_back(tmp);
        }
        Matrix.push_back(vectmp);
    }

    for( size_t i = 0 ; i < bn-1 ; i++ )
    {
        P.insert(cn-i-2);
    }
    solve();
}
////////////////////////////////////
//glpk input:
///* Variables */
//var x1 >= 0;
//var x2 >= 0;
//var x3 >= 0;
///* Object function */
//maximize z: x1 + 14*x2 + 6*x3;
///* Constrains */
//s.t. con1: x1 + x2 + x3 <= 4;
```

```
//s.t. con2: x1  <= 2;
//s.t. con3: x3  <= 3;
//s.t. con4: 3*x2 + x3  <= 6;
//end;
////////////////////////////////////
//myinput:
/*
8 5
1 14 6 0 0 0 0 0
1 1 1 1 0 0 0 4
1 0 0 0 1 0 0 2
0 0 1 0 0 1 0 3
0 3 1 0 0 0 1 6
*/
////////////////////////////////////
```



【理论罗列】：

1.标准型

m个约束 n个变量用x向量表示 A是一个m*n的矩阵 c是一个n的向量 b是一个m的向量

最大化 cx

满足约束 Ax<=b x>0

2.松弛型

基本变量 B |B|=m 一个约束对应一个 表示松弛量 叫做松弛变量(基本变量)

非基变量 N |N|=n

xn+i=bi-sigma{aijxj}>=0

3.替入变量 xe(非基变量)

替出变量 xl(基本变量)

4.可行解

基本解：所有非基变量设为0

基本可行解

5.单纯形法的过程中B和N不断交换，在n维空间中不断走

“相当于不等式上的高斯消元”

【代码实现】：

pivot是转动操作

基本思想就是改写l这个约束为xe作为基本变量，然后把这个新xe的值带到其他约束和目标函数中，就消去xe了

改写和带入时要修改b和a 目标函数则是 c和v

转动时l和e并没有像算法导论上一样a矩阵用了两行分别是a[l][]和a[e][]（这样占用内存大），而是用了同一行，这样a矩阵的行数=|B|，列数=|N|

也就是说，约束条件只用m个，尽管B和N不断交换，但同一时间还是只有m个约束(基本变量)n个非基变量

注意改写成松弛型后a矩阵实际系数为负

（一个优化 a[i][e]为0的约束没必要带入了

simplex是主过程

基本思想是找到一个c[e]>0的，然后找对这个e限制最紧的l，转动这组l e

注意精度控制eps

c[e]>eps

还有找l的时候a[i][e]>eps才行

【对偶原理】：

1.原始线性规划 对偶线性规划

2.对于

最大化 cx

满足约束 Ax<=b x>0

对偶问题为

最小化 bx

满足约束 ATx>=c x>0 （ AT为A的转置 ）

可以转化很多问题来避免初始解不可行

我来秀智商了……

说从前有个线性规划
min c x^T
Ax = b
x >= 0
这里面A是矩阵，x、b、c都是向量
x^T表示转置

啊……我们假设以上出现的所有元素都是整数……

那么Ax = b要是恰好方程数等于未知数数，且解出来恰好为正数是不是就超开心？（假设是线性无关的）
根据那个啥法则，x_i = det(A_i) / det(A)
det(A)表示A的行列式
A_i表示把A的第i列换为b之后的矩阵
如果det(A_i)恰好是det(A)的倍数那不就超开心？这样
但是现实是残酷的，往往这家伙会除出来小数，然后整数规划就坑爹了。

但是人类的智慧是无穷的！
我们现在还是假定“恰好方程数等于未知数数，且解出来恰好为正数”
我们再加个限制：det(A) = 1或-1
就233了吧。
解一定是整数了。
于是可以顺利变为整数规划。我们把det(A) = 1, -1的矩阵称为幺模矩阵。

但是现实是残酷的，“恰好方程数等于未知数数，且解出来恰好为正数”往往不满足。
但是其实没关系。由于每个方程都对应着一个平面，所以解的空间是单纯形，最优解一定会出现在顶点上。
何为顶点？就是平面的交点。
何为平面？一共m + n个：Ax = b是m个方程，x = 0是n个方程。（本来是x >= 0，我们只靠虑切割空间的平面……）
要是顶点都是整点不是超开心？等价于从这m + n个方程中任取n个方程把它解掉得到的解是整数解。
通过前面的分析我们知道，如果恰好选取的这n个方程的系数矩阵的行列式值为1，-1就超开心了。当然要是行列式值为0对应着无解或无穷多解的情况，它又不是顶点管它做甚……
考察系数矩阵
一个是A，好大好大
另一个是x = 0的系数，易知就是单位矩阵I
你从I中选哪几行……由于行列式的性质……一行*k加到另一行上去行列式的值不变，那么对应的未知数就会被干掉……
所以等价于……
从A中任意选取是一个子方阵，它的行列式的值只可能为1, -1, 0。
这样的矩阵我们称为全幺模矩阵。

番外篇：

1. 必要不充分：只含1，-1，0。因为单个元素可以看作行列式……
2. 充分必要：对它进行高斯消元的主元操作……（好像叫转轴？啊反正就是消别人的未知数……），得来的还是全幺模矩阵……这个是因为那个啥啥幺模矩阵组成了一个乘法群？用这个性质我们可以不用double。
3. 您可以手工屠矩阵用定义证它是全幺模！
4. 如果数学太差，您也可以写一个O(4^n * n^3)的强程序证它是全幺模！

orzorzorz

附上一道题BZOJ 1061

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>
using namespace std;
typedef long long ll;
const int M=10005,N=1005,INF=1e9;
const double eps=1e-6;
inline int read(){
    char c=getchar();int x=0,f=1;
    while(c<'0'||c>'9'){if(c=='-')f=-1; c=getchar();}
    while(c>='0'&&c<='9'){x=x*10+c-'0'; c=getchar();}
    return x*f;
}

int n,m;
double a[M][N],b[M],c[N],v;
void pivot(int l,int e){
    b[l]/=a[l][e];
    for(int j=1;j<=n;j++) if(j!=e) a[l][j]/=a[l][e];
    a[l][e]=1/a[l][e];

    for(int i=1;i<=m;i++) if(i!=l&&fabs(a[i][e])>0){
        b[i]-=a[i][e]*b[l];
        for(int j=1;j<=n;j++) if(j!=e) a[i][j]-=a[i][e]*a[l][j];
        a[i][e]=-a[i][e]*a[l][e];
    }

    v+=c[e]*b[l];
    for(int j=1;j<=n;j++) if(j!=e) c[j]-=c[e]*a[l][j];
    c[e]=-c[e]*a[l][e];

    //swap(B[l],N[e])
}

double simplex(){
    while(true){
        int e=0,l=0;
        for(e=1;e<=n;e++) if(c[e]>eps) break;
        if(e==n+1) return v;
        double mn=INF;
        for(int i=1;i<=m;i++)
            if(a[i][e]>eps&&mn>b[i]/a[i][e]) mn=b[i]/a[i][e],l=i;
        if(mn==INF) return INF;//unbounded
        pivot(l,e);
    }
}

int main(){
    n=read();m=read();
    for(int i=1;i<=n;i++) c[i]=read();
    for(int i=1;i<=m;i++){
        int s=read(),t=read();
        for(int j=s;j<=t;j++) a[i][j]=1;
        b[i]=read();
    }
    printf("%d",(int)(simplex()+0.5));
}
```

附上几道题的题号，练习学习一下：

BZOJ 3550

BZOJ 3112

BZOJ 3265

BZOJ 1061

BZOJ 1061

POJ 1275

标准型

constraint：

$$, \ i = 1, 2, \ldots, m$$

$$\geq 0, \ j = 1, 2, \ldots, n$$

个变量，

个约束

构造

的矩阵 ， 维向量 ， 维向量

constraint：

转化为标准型:

若目标函数要求取最小值，那么可以对其取相反数变成取最大值。对于限制条件$f(x_1, x_2, \ldots, x_n) = b$，可以用两个不等式$f(x_1, x_2, \ldots, x_n) \leq b, -f(x_1, x_2, \ldots, x_n) \leq -b$描述，对于限制条件$f(x_1, x_2, \ldots, x_n) \geq b$，可以用不等式$-f(x_1, x_2, \ldots, x_n) \leq -b$描述。对于无限制的变量$x$，可以将其拆为两个非负变量$x_0, x_1$，使得$x = x_0 - x_1$。

松弛型

松弛变量

–

, – 0

等式左侧为**基本变量**，右侧为**非基本变量**

单纯型算法

每个约束定义了

维空间中的一个半空间(超平面)，交集形成的可行域是一个凸区域称为单纯型

目标函数是一个超平面，最优解在凸区域定点处取得

基本解：非基本变量值为

，基本变量为右侧的常数

基本可行解：所有

通过不断的转轴操作，在

维凸区域的顶点上不断移动(转轴)，使得基本解的目标值不断变大，最终达到最优解

转轴：

选取一个非基本变量

为替入变量，基本变量

为替出变量，将其互换

为了防止循环，**根据**

规则，选择下标最小的变量

初始化：

算法导论上有一个辅助线性规划的做法

但我发现好多人都用了**随机初始化**的黑科技

在所有

的约束中随机选一个作为 ，再随机选一个 的作为 ，然后 后

就变正了...

代码实现：

直接用一个

来保存目标函数和约束

里的各种操作推导一下很清楚，用了两个

避免了一些判断

用来保存基本变量和非基本变量集合

针对全幺模矩阵可以进行提取非零系数的优化

posted @ 2017-06-30 11:00 Angel_Kitty 阅读(70) 评论(0) 编辑 收藏