序列二次规划算法

1 序列二次规划法简介

非线性规划问题是目标函数或约束条件中包含非线性函数的规划问题。一般说来，解非线性规划要比解线性规划问题困难得多。而且，也不像线性规划有单纯形法这一通用方法，非线性规划目前还没有适于各种问题的一般算法，各个方法都有自己特定的适用范围。当目标函数和约束条件具有良好的分析性质时，人们喜欢用间接法分析与求解约束最优化问题。利用间接法求解最优化问题的途径一般有两种，另一种途径是在可行域内使目标函数下降的迭代点法，如可行点法。另一种是利用目标函数和约束条件构造增广目标函数，借此将约束最优化问题转化为无约束最优化问题，然后利用求解无约束最优化问题的方法间接求解新目标函数的局部最优解或稳定点，如人们所熟悉的惩罚函数法，乘子法和序列二次规划等。

序列二次规划算法是目前公认的求解约束非线性优化问题的最有效方法之一，与其它优化算法相比，其最突出的优点是收敛性好、计算效率高、边界搜索能力强，受到了广泛重视及应用。但其迭代过程中的每一步都需要求解一个或多个二次规划子问题。一般地，由于二次规划子问题的求解难以利用原问题的稀疏性、对称性等良好特性，随着问题规模的扩大，其计算工作量和所需存贮量是非常大的。因此，目前的序列二次规划方法一般只适用于中小型问题。另外，由于大型二次规划问题的求解通常使用迭代法，所需解的精度越高，花费的时间就越多，稳定性也就越差，相对线性方程组的求解理论来说，二次规划的求解是不完善的。

2 序列二次规划的研究

最优化理论及方法是一个具有广泛应用背景的研究领域。它研究诸如从众多的方案中选出最优方案等问题，常见的各种模型如线性规划，二次规划，非线性规划，多目标规划等。最优化理论及方法已经在经济计划，工程设计，生产管理，

交通运输等领域得到了广泛的应用，吸引了很多学者的研究。

一般非线性约束的数学规划问题是：

$$\begin{aligned} &\min \ f(X) \\ &s.t.\ g_u(X) \le 0 \quad (u = 1,2,...,p) \\ &\quad h_v(X) = 0 \quad (v = 1,2,...,m) \end{aligned}$$

其中 $X \in R^n$ 是决策变量，$f(X)$ 是目标函数，$g_u(X)$，$h_v(X)$ 分别是不等式约束函数和等式约束函数。

在求解上述问题的诸多算法中序列二次规划(SQP)是最有效的一类方法之一"由于它具有超线性收敛速度,对它的研究一直是非线性规划的一个热点,许多学者对它进行了研究和改进"

序列二次规划（SQP）算法最早由 Wilson（1963）提出。Wilson 提出了 Newton-SQP 方法。60 年代末和 70 年代初，随着解无约束优化问题的拟 Newton 法的发展，拟 Newton-SQP 算法的研究引起了学者们的高度重视。Polomares-Mangasarian（1976）提出了一类拟 Newton-SQP 方法。在该类方法中，他采用对 Lagrange 函数的整体 Hesse 矩阵（即既对 x 又对 Lagrange 乘子的 Hesse 矩阵）的近似进行修正的方法来修正。Han（1976，1977）证明了 PSB-SQP 和 BFGS-SQP 方法的局部收敛性。而且，若采用精确罚函数进行搜索，则算法具有全局收敛性。自此，SQP 算法的研究受到了广泛的重视。迄今为止，SQP 算法的研究工作已取得了丰硕的成果。

SQP 算法的主要优点之一是它的全局收敛性和局部超线性收敛性。为使算法具有全局收敛性，通常要求子问题（QP）中的矩阵 H 对称正定。使得（QP）产生的方向 S 是某个效益函数的下降方向。另一方面，若矩阵 H 对称正定，则子问题（QP）是一个严格凸二次规划。此时，该问题有唯一解。而且，其求解也相对容易。因此，研究矩阵 H 的对称正定性是一项非常重要的工作。并已引起了学者们的重视。Powell（1976）提出了一个修正的 BFGS 公式。该公式可保证产生的矩阵是对称正定的。数值计算的结果表明利用该修正公式的 SQP 算法的数值效果较好。然而该算法的局部收敛性质尚不清楚。

Coleman 和 Conn（1984）提出了一种既约 Hessian-SQP 方法。即用一系列

的正定矩阵去逼近解$[S^*,\lambda^*]$处的既约 Hessian 矩阵。Noeedal 和 Overton（1985）提出了单边及双边的投影 Hessian 方法。在单边投影 Hessian 方法中证明了局部 1-步超线性收敛性而在双边投影 Hessian 方法中证明了在某种假设条件下的 2-步超线性收敛性质。但他的 BFGS 修正公式是在满足一定的修正准则才进行修正。Xie 和 Byrd（1999）在 RHSQP 方法的基础上加上线性搜索得到全局收敛性算法。他并且把 Nocedal 和 Overton 的修正法则改为了一种新的修正法则一正曲率法则。这样能得到更好的数值结果。并且证明了当利用两个常用的效益函数，Flether 效益函数和l_1精确罚函数进行线性搜索时，算法具有全局收敛性和 R-线性收敛性。

R.Fontecilla（1988）提出了一种 2-步超线性收敛算法。与 Coleman 和 Conn 不同之处在于他将步长犷分为互相垂直的两个向量：即水平方向的步长h_k和垂直方向的步长v_k。且仅利用水平方向的步长h_k对H进行修正。与 coleman 和 Conn 不同之处在于对S^k进行分解时他使用了一个正交投影算子而不是标准正交基，比较容易满足连续性条件。比前者适用的范围更广。而且，他把这一方法用在了几种不同的乘子修正公式中。在二阶充分条件下，证明了该算法是 2-步超线性收敛的。若在每一次迭代中再计算一次约束函数的值，则算法为 1-步超线性收敛。但没有全局收敛性研究。

3 序列二次规划算法推导过程

序列二次规划（SQP）算法是将复杂的非线性约束最优化问题转化为比较简单的二次规划（QP）问题求解的算法。所谓二次规划问题就是目标函数为二次函数，约束函数为线性函数的最优化问题。二次规划问题是最简单的非线性约束最优化问题。

3.1 序列二次规划算法思想

非线性约束最优化问题：

$$\begin{aligned} &\min \ \ f(X) \\ &s.t.\ g_u(X) \le 0 \quad (u = 1,2,...,p) \\ &\quad h_v(X) = 0 \quad (v = 1,2,...,m) \end{aligned} \tag{1-1}$$

利用泰勒展开把非线性约束问题式（1-1）的目标函数在迭代点 X^k 简化成二次函数，把约束函数简化成线性函数后得到的就是如下的二次规划问题：

$$\begin{aligned} &\min \ \ f(X) = \frac{1}{2}[X - X^k]^{\mathrm{T}} \nabla^2 f(X^k)[X - X^k] + \nabla f(X^k)^{\mathrm{T}}[X - X^k] \\ &s.t.\ \nabla g_u(X^k)^{\mathrm{T}}[X - X^k] + g_u(X^k) \le 0 \quad (u = 1,2,...,p) \\ &\quad \nabla h_v(X^k)^{\mathrm{T}}[X - X^k] + h_v(X^k) = 0 \quad (v = 1,2,...,m) \end{aligned} \tag{1-2}$$

此问题是原约束最优化问题的近似问题，但其解不一定是原问题的可行点。为此，令

$$S = X - X^k$$

将上述二次规划问题变成关于变量的 S 的问题，即

$$\begin{aligned} &\min \ \ f(X) = \frac{1}{2} S^{\mathrm{T}} \nabla^2 f(X^k) S + \nabla f(X^k)^{\mathrm{T}} S \\ &s.t.\ \nabla g_u(X^k)^{\mathrm{T}} S + g_u(X^k) \le 0 \quad (u = 1,2,...,p) \\ &\quad \nabla h_v(X^k)^{\mathrm{T}} S + h_v(X^k) = 0 \quad (v = 1,2,...,m) \end{aligned} \tag{1-3}$$

令

$$\begin{aligned} &H = \nabla^2 f(X^k) \\ &C = \nabla f(X^k) \\ &A_{\mathrm{eq}} = [\nabla h_1(X^k), \nabla h_2(X^k), ..., \nabla h_m(X^k)]^{\mathrm{T}} \\ &A = [\nabla g_1(X^k), \nabla g_2(X^k), ..., \nabla g_p(X^k)]^{\mathrm{T}} \\ &B_{\mathrm{eq}} = [h_1(X^k), h_2(X^k), ..., h_m(X^k)]^{\mathrm{T}} \\ &B = [g_1(X^k), g_2(X^k), ..., g_p(X^k)]^{\mathrm{T}} \end{aligned} \tag{1-4}$$

将式（1-4）变成二次规划问题的一般形式，即

$$\begin{aligned} &\min \quad \frac{1}{2}S^{\mathrm{T}}HS + C^{\mathrm{T}}S \\ &s.t. \quad AS \le B \\ &\qquad A_{\mathrm{eq}}S = B_{\mathrm{eq}} \end{aligned} \tag{1-5}$$

求解此二次规划问题，将其最优解S^*作为原问题的下一个搜索方向S^k，并在该方向上进行原约束问题目标函数的约束一维搜索，就可以得到原约束问题的一个近似解X^{k+1}。反复这一过程，就可以求得原问题的最优解。

上述思想得以实现的关键在于如何计算函数的二阶导数矩阵H，如何求解式（1-5）所示的二次规划问题。

3.2 二阶导数矩阵的计算

二阶导数矩阵的近似计算可以利用拟牛顿（变尺度）法中变尺度矩阵计算的DFP公式

$$H^{k+1} = H^k + \frac{\Delta X^k [\Delta X^k]^{\mathrm{T}}}{[\Delta q^k]^{\mathrm{T}} \Delta X^k} - \frac{H^k \Delta q^k [\Delta q^k]^{\mathrm{T}} H^k}{[\Delta q^k]^{\mathrm{T}} H^k \Delta q^k} \tag{1-6}$$

或BFGS公式

$$\begin{aligned} H^{k+1} = H^k + \frac{1}{[\Delta X^k]^{\mathrm{T}} \Delta q^k} \Big\{ & \Delta X^k [\Delta X^k]^{\mathrm{T}} + \\ & \frac{\Delta X^k [\Delta X^k]^{\mathrm{T}} [\Delta q^k]^{\mathrm{T}} H^k \Delta q^k}{[\Delta X^k]^{\mathrm{T}} \Delta q^k} - H^k \Delta q^k [\Delta q^k]^{\mathrm{T}} - \Delta X^k [\Delta q^k]^{\mathrm{T}} H^k \Big\} \end{aligned} \tag{1-7}$$

3.3 二次规划问题的求解

二次规划问题式（1-7）的求解分为以下两种情况。

（1） 等式约束二次规划问题

$$\begin{aligned} &\min \ \ f(X)=\frac{1}{2}S^{\mathrm{T}}HS+C^{\mathrm{T}}S \\ &s.t. \ \ A_{\mathrm{eq}}S=B_{\mathrm{eq}} \end{aligned} \tag{1-8}$$

其拉格朗日函数为

$$\min \ \ L(\mathrm{S},\lambda)=\frac{1}{2}S^{\mathrm{T}}HS+C^{\mathrm{T}}S+\lambda^{\mathrm{T}}(A_{\mathrm{eq}}S-B_{\mathrm{eq}})$$

由多元函数的极值条件$\nabla L(\mathrm{S},\lambda)=0$得

$$\begin{aligned} HS+C+A_{\mathrm{eq}}^{\mathrm{T}}\lambda=0 \\ A_{\mathrm{eq}}S-B_{\mathrm{eq}}=0 \end{aligned}$$

写成矩阵形式，即

$$\begin{bmatrix} H & A_{\mathrm{eq}}^{\mathrm{T}} \\ A_{\mathrm{eq}} & 0 \end{bmatrix}\begin{bmatrix} S \\ \lambda \end{bmatrix}=\begin{bmatrix} -C \\ B_{\mathrm{eq}} \end{bmatrix} \tag{1-9}$$

式（1-9）其实就是以$[S,\lambda]^{\mathrm{T}}$为变量的线性方程组，而且变量数和方程数都等于$n+m$。由线性代数知，此方程要么无解，要么有惟一解。如果有解，利用消元变换可以方便的地求出该方程的惟一解，记作$[S^{k+1},\lambda^{k+1}]^{\mathrm{T}}$。根据 k-t 条件，若此解中的乘子向量$\lambda^{k+1}$不全为零，则$S^{k+1}$就是等式约束二次规划问题式（1-8）的最优解$S*$，即$S*=S^{k+1}$。

（2） 一般约束二次规划问题

对于一般约束下的二次规划问题式（1-5），在不等式约束条件中找出迭代点X^k的起作用的约束，将等式约束和起作用的约束组成新的约束条件，构成新的等式约束问题：

$$\begin{aligned} &\min \ \ f(X)=\frac{1}{2}S^{\mathrm{T}}HS+C^{\mathrm{T}}S \\ &s.t. \ \ \sum_{i\in E\bigcup I_k}\sum_{j=1}^{n}a_{ij}s_j=b_j \end{aligned} \tag{1-10}$$

其中，E代表等式约束下的集合，I_k代表不等式约束中起作用约束的下标集合。

此式即式（1-8），可以用同样的方法求解。在求得式（1-10）的解$[S^{k+1},\lambda^{k+1}]^{T}$之后，根据 k-t 条件，若解中对应原等式约束条件的乘子不全为零，对应起作用约束条件的乘子不小于零，则S^{k+1}就是所求一般约束二次规划问题式（1-5）的最优解$S*$。

综上所述，在迭代点X^{k}上先进行矩阵H^{k}的变更，在构造和求解相应的二次规划子问题，并该子问题最优解$S*$作为下一次迭代的搜索方向S^{k}。然后在该方向上对原非线性最优化问题目标函数进行约束一维搜索，得到下一个迭代点X^{k+1}，并判断收敛精度是否满足。重复上述过程，直到迭代点X^{k+1}最终满足终止准则，得到原非线性最约束问题的最优解$X*$为止。这种算法称为二次规划法，它是目前求解非线性约束最优化问题的常用方法，简称 SQP 法。

序列二次规划算法的迭代步奏如下：

①给定初始点X^{0}、收敛精度ε，令$H^{0}=I$（单位矩阵），置$k=0$.

②在点X^{k}简化原问题为二次规划问题式（1-10）。

③求解二次规划问题，并令$S^{k}=S*$。

④在方向S^{k}上对原问题目标函数进行约束一维搜索，得点X^{k+1}。

⑤终止判断，若X^{k+1}满足给定的精度的终止准则，则令$X*=X^{k+1}$，$f*=f(X^{k+1})$，输出最优解，终止计算，否则转⑥。

⑥按式（1-6）或（1-7）修正H^{k+1}，令$k=k+1$，转②继续迭代。

4 MATLAB 程序

```
%*************************************************************************************

%*************************************************************************************
function [d,mu,lam,val,k]=qpsubp(dfk,Bk,Ae,hk,Ai,gk)
%功能：求解二次规划子问题
%输入：dfk是xk处的梯度，Bk是第k次近似Hesse阵，Ae，hk线性等式约束
```

```
%的有关参数，Ai，gk是线性不等式约束的有关参数
%输出：d，val分别是是最优解和最优值，mu,lam是乘子向量，k是迭代次数
%功能：求解二次规划子问题
%输入：dfk是xk处的梯度，Bk是第k次近似Hesse阵，Ae,hk线性等式约束
%的有关参数，Ai,gk是线性不等式约束的有关参数
%输出：d，val分别是是最优解和最优值，mu,lam是乘子向量，k是迭代次数
.n=length(dfk); l=length(hk); m=length(gk);
gamma=0.05; epsilon=1.0e-6; rho=0.5; sigma=0.2;
ep0=0.05; mu0=0.05*zeros(l,1); lam0=0.05*zeros(m,1);
d0=ones(n,1); u0=[ep0;zeros(n+l+m,1)];
z0=[ep0; d0; mu0;lam0,];
k=0; %k为迭代次数
z=z0; ep=ep0; d=d0; mu=mu0; lam=lam0;
while (k<=150)
    dh=dah(ep,d,mu,lam,dfk,Bk,Ae,hk,Ai,gk);
    if(norm(dh)<epsilon)
        break;
    end
    A=JacobiH(ep,d,mu,lam,dfk,Bk,Ae,hk,Ai,gk);
    b=beta(ep,d,mu,lam,dfk,Bk,Ae,hk,Ai,gk,gamma)*u0-dh;
    dz=A\b;
    if(l>0&&m>0)
        de=dz(1); dd=dz(2:n+1); du=dz(n+2:n+l+1); dl=dz(n+l+2:n+l+m+1);
    end
    if(l==0)
        de=dz(1); dd=dz(2:n+1); dl=dz(n+2:n+m+1);
    end
    if(m==0)
        de=dz(1); dd=dz(2:n+1); du=dz(n+2:n+l+1);
    end
    i=0; %mk=0;
    mm=0;
    while (mm<=20)
        if(l>0&&m>0)
            dh1=dah(ep+rho^i*de,d+rho^i*dd,mu+rho^i*du,lam+rho^i*dl,dfk,Bk,Ae,hk,Ai,gk);
        end
        if(l==0)
            dh1=dah(ep+rho^i*de,d+rho^i*dd,mu,lam+rho^i*dl,dfk,Bk,Ae,hk,Ai,gk);
        end
        if(m==0)
            dh1=dah(ep+rho^i*de,d+rho^i*dd,mu+rho^i*du,lam,dfk,Bk,Ae,hk,Ai,gk);
        end
        if(norm(dh1)<=(1-sigma*(1-gamma*ep0)*rho^i)*norm(dh))
            mk=i; break;
```

```
            end
            i=i+1;
            if(i==20), mk=10; end
        end
        alpha=rho^mk;
        if(l>0&&m>0)
            ep=ep+alpha*de; d=d+alpha*dd;
            mu=mu+alpha*du; lam=lam+alpha*dl;
        end
        if(l==0)
            ep=ep+alpha*de; d=d+alpha*dd;
            lam=lam+alpha*dl;
        end
        if(m==0)
            ep=ep+alpha*de; d=d+alpha*dd;
            mu=mu+alpha*du;
        end
        k=k+1;
end
%*************************************************************************************
function p=phi(ep,a,b)
p=a+b-sqrt(a^2+b^2+2*ep^2);
%*************************************************************************************
function dh=dah(ep,d,mu,lam,dfk,Bk,Ae,hk,Ai,gk)
n=length(dfk); l=length(hk); m=length(gk);
dh=zeros(n+l+m+1,1);
dh(1)=ep;
if(l>0&&m>0)
    dh(2:n+1)=Bk*d-Ae'*mu-Ai'*lam+dfk;
    dh(n+2:n+l+1)=hk+Ae*d;
    for i=1:m
        dh(n+l+1+i)=phi(ep,lam(i),gk(i)+Ai(i,:)*d);
    end
end
if(l==0)
    dh(2:n+1)=Bk*d-Ai'*lam+dfk;
    for i=1:m
        dh(n+1+i)=phi(ep,lam(i),gk(i)+Ai(i,:)*d);
    end
end
if(m==0)
    dh(2:n+1)=Bk*d-Ae'*mu+dfk;
    dh(n+2:n+l+1)=hk+Ae*d;
```

```
end
dh=dh(:);
%*****************************************************************************************
function bet=beta(ep,d,mu,lam,dfk,Bk,Ae,hk,Ai,gk,gamma)
dh=dah(ep,d,mu,lam,dfk,Bk,Ae,hk,Ai,gk);
bet=gamma*norm(dh)*min(1,norm(dh));
%*****************************************************************************************
function [dd1,dd2,v1]=ddv(ep,d,lam,Ai,gk)
m=length(gk);
dd1=zeros(m,m); dd2=zeros(m,m); v1=zeros(m,1);
for(i=1:m)
    fm=sqrt(lam(i)^2+(gk(i)+Ai(i,:)*d)^2+2*ep^2);
    dd1(i,i)=1-lam(i)/fm;
    dd2(i,i)=1-(gk(i)+Ai(i,:)*d)/fm;
    v1(i)=-2*ep/fm;
end
%*****************************************************************************************
function A=JacobiH(ep,d,mu,lam,dfk,Bk,Ae,hk,Ai,gk)
n=length(dfk); l=length(hk); m=length(gk);
A=zeros(n+l+m+1,n+l+m+1);
[dd1,dd2,v1]=ddv(ep,d,lam,Ai,gk);
if(l>0&&m>0)
    A=[1, zeros(1,n), zeros(1,l), zeros(1,m);
       zeros(n,1), Bk, -Ae', -Ai';
       zeros(l,1), Ae, zeros(l,l), zeros(l,m);
       v1, dd2*Ai, zeros(m,l), dd1];
end
if(l==0)
    A=[1, zeros(1,n), zeros(1,m);
       zeros(n,1), Bk, -Ai';
       v1, dd2*Ai, dd1];
end
if(m==0)
    A=[1, zeros(1,n), zeros(1,l);
       zeros(n,1), Bk, -Ae';
       zeros(l,1), Ae, zeros(l,l)];
end
%*****************************************************************************************

%*****************************************************************************************
function [x,mu,lam,val,k]=sqpm(x0,mu0,lam0)
%功能：用基于拉格朗日函数Hesse阵的SQP方法求解约束优化问题：
%    min    f(x)              s.t.     h_i(x)=0，i=1,...,l.
```

```
%输入：  x0是初始点, mu0是乘子向量的初始值
%输出：x，mu分别是近似最优点及相应的乘子
%val是最优值，mh是约束函数的模，k是迭代次数。
maxk=1000;     %最大迭代次数
n=length(x0); l=length(mu0); m=length(lam0);
rho=0.5; eta=0.1; B0=eye(n);
x=x0; mu=mu0; lam=lam0;
Bk=B0; sigma=0.8;
epsilon1=1e-6; epsilon2=1e-5;
[hk,gk]=cons(x); dfk=df1(x);
[Ae,Ai]=dcons(x); Ak=[Ae; Ai];
k=0;
while(k<maxk)
    [dk,mu,lam]=qpsubp(dfk,Bk,Ae,hk,Ai,gk); %求解子问题
    mp1=norm(hk,1)+norm(max(-gk,0),1);
    if(norm(dk,1)<epsilon1)&&(mp1<epsilon2)
        break;
    end %检验终止准则
    deta=0.05; % 罚参数更新
    tau=max(norm(mu,inf),norm(lam,inf));
    if(sigma*(tau+deta)<1)
        sigma=sigma;
    else
        sigma=1.0/(tau+2*deta);
    end
    im=0; %Armijo搜索
    while(im<=20)
        if(phi1(x+rho^im*dk,sigma)-phi1(x,sigma)<eta*rho^im*dphi1(x,sigma,dk))
            mk=im;
            break;
        end
        im=im+1;
        if(im==20), mk=10; end
    end
    alpha=rho^mk; x1=x+alpha*dk;
    [hk,gk]=cons(x1); dfk=df1(x1);
    [Ae,Ai]=dcons(x1); Ak=[Ae; Ai];
    lamu=pinv(Ak)'*dfk; %计算最小二乘乘子
    if(l>0&&m>0)
        mu=lamu(1:l);
        lam=lamu(l+1:l+m);
    end
    if(l==0), mu=[]; lam=lamu; end
    if(m==0), mu=lamu; lam=[]; end
```

```
    sk=alpha*dk; %更新矩阵Bk
    yk=dlax(x1,mu,lam)-dlax(x,mu,lam);
    if(sk'*yk>0.2*sk'*Bk*sk)
        theta=1;
    else
        theta=0.8*sk'*Bk*sk/(sk'*Bk*sk-sk'*yk);
    end
    zk=theta*yk+(1-theta)*Bk*sk;
    Bk=Bk+zk*zk'/(sk'*zk)-(Bk*sk)*(Bk*sk)'/(sk'*Bk*sk);
    x=x1; k=k+1;
end
    val=f1(x);
%*************************************************************************************

%*************************************************************************************
%**************************精确价值函数*****************************%
function p=phi1(x,sigma)
        f=f1(x); [h,g]=cons(x); gn=max(-g,0);
        l0=length(h); m0=length(g);
        if(l0==0), p=f+1.0/sigma*norm(gn,1); end
        if(m0==0), p=f+1.0/sigma*norm(h,1); end
        if(l0>0&&m0>0)
            p=f+1.0/sigma*(norm(h,1)+norm(gn,1));
        end
%*************************************************************************************
%************************价值函数的方向导数*****************************%
function dp=dphi1(x,sigma,d)
        df=df1(x); [h,g]=cons(x); gn=max(-g,0);
        l0=length(h); m0=length(g);
        if(l0==0), dp=df'*d-1.0/sigma*norm(gn,1); end
        if(m0==0), dp=df'*d-1.0/sigma*norm(h,1); end
        if(l0>0&&m0>0)
            dp=df'*d-1.0/sigma*(norm(h,1)+norm(gn,1));
        end
%*************************************************************************************
%**************************拉格朗日函数*****************************%

function l=la(x,mu,lam)
    f=f1(x); %调用目标函数文件
    [h,g]=cons(x); %调用约束函数文件
    l0=lemgth(h); m0=length(g);
    if(l0==0), l=f-lam*g; end
    if(m0==0), l=f-mu'*h; end
```

```
    if(l0.0&m0.0)
        l=f-mu'*h-lam'*g;
    end
%*************************************************************************************
%************************拉格朗日函数的梯度*************************%
function dl=dlax(x,mu,lam)
    df=df1(x);   %调用目标函数梯度文件
    [Ae,Ai]=dcons(x); %调用约束函数Jacobi矩阵文件
    [m1,m2]=size(Ai); [l1,l2]=size(Ae);
    if(l1==0), dl=df-Ai'*lam; end
    if(m1==0), dl=df-Ae'*mu; end
    if(l1>0&&m1>0), dl=df-Ae'*mu-Ai'*lam; end
%*************************************************************************************
function f=f1(x)
  f=        ；%(目标函数)
%*************************************************************************************
function df=df1(x)
  df=[        ];   %目标函数关于各个变量的导数，中间用“;”隔开
%*************************************************************************************
function [h,g]=cons(x)
  h=[];   %输入等式约束函数，中间用“;”隔开
  g=[];   %输入不等式约束函数，中间用“;”隔开
%*************************************************************************************
function [dh,dg]=dcons(x)
  dg=[];   %等式约束函数关于各个变量的导数，中间用“,”隔开
  dh=[];   %不等式约束函数关于各个变量的导数，中间用“,”隔开

%*************************************************************************************
  %在MATLAB的命令窗口输入以下命令
  x0=[]';   %各个变量的初值，中间用“空格”隔开；
  >> mu0=[]';   %等式约束函数的个数，有几个就有几个“0”，中间用“空格”隔开
  >> lam0=[]';   %不等式约束函数的个数，有几个就有几个“0”，中间用“空格”隔开
  [x,mu,k]=sqpm(x0,mu0,lam0)
%*************************************************************************************
%*************************************************************************************
```

5 实例检验和计算

对于上述程序，我们首先应该验证其正确性，才能用来解决问题。

例 1：

$$\begin{aligned} \min \quad & f(x) = -\pi x_1^2 x_2 \\ \text{s.t.} \quad & \pi x_1 x_2 + \pi x_1^2 - 150 = 0, \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$

根据例 1 的要求，改动相应的 f1.m，df1.m，cons.m，dcons.m 文件，在命令窗口输入相应的命令，结果如下图所示：

```
>> x0=[3 3]';
mu0=[0]';
lam0=[0 0]';
[x,mu,lam,val,k]=sqpm(x0,mu0,lam0)

x =

   3.98942280535240
   7.97884560241570


mu =

  -3.98942280337298


lam =

  1.0e-007 *

  -0.42673868599712
  -0.00395338899706


val =

   -3.989422803883991e+002


k =

     8
```

对于上述结果，我们可以用相应的 MATLAB 的优化函数来解决，看所得结果是否相同。

编写 MATLAB 目标函数文件

```
function [c,ce]=cxmcon(x)
ce=[pi*x(1)*x(2)+pi*x(1)^2-150];
c=[    ];
```

非线性约束函数返回变量分别是 c 和 ce 两个量，其中，前者为不等式约束的数学描述，后者为非线性等式约束，如果某个约束不存在，则应该将其赋值为空矩阵。

观察可知，例 1 只有非线性等式约束，没有非线性不等式约束，则 A，B，Aeq，Beq，都将为空矩阵了。另外，应该给出搜索的初值，向量 x0=[3,3]’，因此，可以调用 fmincon()函数求解此约束最优化问题。

在命令窗口输入以下内容：

```
>>f=@(x)-pi*x(1)^2*x(2);
   ff=optimset;
   ff.LargeScale='off';
   ff.TolX=1e-15;ff.TolFun=1e-20;ff.TolCon=1e-20;
   x0=[1;1];
   xm=[0;0];xM=[];A=[];B=[];Aeq=[];Beq=[];
   [x,f_opt,c,d]=fmincon(f,x0,A,B,Aeq,Beq,xm,xM,@cxmcon)
```

程序运行后的优化结果：

```
x =

   3.98942281041081
   7.97884558244272


f_opt =

   -3.989422804014328e+002


c =

     1


d =

       iterations: 14
        funcCount: 65
         stepsize: 1
        algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
    firstorderopt: 1.213606992678251e-011
     cgiterations: []
          message: [1x144 char]
```

例 2：

$$\begin{aligned} \min \quad & f(x) = x_1^2 + x_2^2 - 16x_1 - 10x_2 \\ \text{s.t.} \quad & -x_1^2 + 6x_1 - 4x_2 + 11 \geq 0, \\ & x_1x_2 - 3x_2 - \mathrm{e}^{x_1-3} + 1 \geq 0, \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$

根据例 2 的要求，改动相应的 f1.m，df1.m，cons.m，dcons.m 文件，在命令窗口输入相应的命令，结果如下图所示：

```
>> x0=[4 4]';
mu0=[ ]';
lam0=[0 0 0 0]';
[x,mu,lam,val,k]=sqpm(x0,mu0,lam0)

x =

   5.23960911550414
   3.74603775243890


mu =

   Empty matrix: 0-by-1


lam =

   0.81328649230256
   0.33274622295902
  -0.00000000000000
  -0.00000000000000


val =

 -79.80782084648367


k =

     5
```

调用相应的 MATLAB 的优化函数来解决，看所得结果是否相同。

编写 MATLAB 目标函数文件：

```
function [c,ce]=ccon(x)
ce=[];
c=[-x(1)^2+6*x(1)-4*x(2)+11;x(1)*x(2)-3*x(2)-exp(x(1)-3)+1;x(1);x(2)];
```

在命令窗口输入以下内容：

```
>>f=@(x)x(1)^2+x(2)^2-16*x(1)-10*x(2);
   ff=optimset;
   ff.LargeScale='off';
```

```
ff.TolX=1e-15;ff.TolFun=1e-20;ff.TolCon=1e-20;
x0=[1;1];
xm=[0;0];xM=[];A=[];B=[];Aeq=[];Beq=[];
[x,f_opt,c,d]=fmincon(f,x0,A,B,Aeq,Beq,xm,xM,@cxmcon)
```

程序运行后的优化结果：

```
x =

   5.19805200809520
   3.98740493869316


f_opt =

 -80.12373869247764


c =

     1


d =

       iterations: 6
        funcCount: 28
         stepsize: 1
        algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
    firstorderopt: 9.411185937224786e-007
     cgiterations: []
          message: [1x144 char]
```

通过比较上述两个例子的计算结果，发现两者的最优点和优化值非常的接近，可见程序是有效的。

例 3：

7.22.5 Heat Exchanger Design [7.42]

Objective function: Minimize $f(\mathbf{X}) = x_1 + x_2 + x_3$

Constraints:

$$g_1(\mathbf{X}) = 0.0025(x_4 + x_6) - 1 \le 0$$
$$g_2(\mathbf{X}) = 0.0025(-x_4 + x_5 + x_7) - 1 \le 0$$
$$g_3(\mathbf{X}) = 0.01(-x_5 + x_8) - 1 \le 0$$
$$g_4(\mathbf{X}) = 100x_1 - x_1x_6 + 833.33252x_4 - 83{,}333.333 \le 0$$
$$g_5(\mathbf{X}) = x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 \le 0$$
$$g_6(\mathbf{X}) = x_3x_5 - x_3x_8 - 2500x_5 + 1{,}250{,}000 \le 0$$

$$g_7 : 100 \le x_1 \le 10{,}000 : g_8$$
$$g_9 : 1000 \le x_2 \le 10{,}000 : g_{10}$$
$$g_{11} : 1000 \le x_3 \le 10{,}000 : g_{12}$$
$$g_{13} \text{ to } g_{22} : 10 \le x_i \le 1000, \quad i = 4, 5, \ldots, 8$$

Optimum solution: $\mathbf{X}^* = \{567 \quad 1357 \quad 5125 \quad 181 \quad 295 \quad 219 \quad 286 \quad 395\}^T$, $f^* = 7049$

根据例 3 的要求，改动相应的 f1.m，df1.m，cons.m，dcons.m 文件，在命令窗口输入相应的命令，结果如下图所示：

```
>> x0=[1000 1200 5000 150 300 200 200 380]';
>> mu0=[ ]';%等式个数
>> lam0=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';%不等式个数
>> [x,mu,lam,Val,k]=sqpm(x0,mu0,lam0)

x =

  1.0e+003 *

   1.10464683366032
   1.00349996220130
   5.57894391011143
   0.17797041898109
   0.27684224359548
   0.22202949012628
   0.30112817538562
   0.37684224359551
```

```
Val =

    7.687090705973056e+003


k =

        1000
```

调用相应的 MATLAB 的优化函数来解决，看所得结果是否相同。

编写 MATLAB 目标函数文件：

```
function [c,ce]=cxmcon(x)
ce=[];
 c=[0.0025*(x(4)+x(6))-1;0.0025*(-x(4)+x(5)+x(7))-1;0.01*(-x(5)+x(8))-1;
 100*x(1)-x(1)*x(6)+833.33252*x(4)-83333.333;x(2)*x(4)-x(2)*x(7)-1250*x(4)+1250*x(5);
x(3)*x(5)-x(3)*x(8)-2500*x(5)+1250000];
```

在命令窗口输入以下内容：

```
>>clear P;
 P.nonlcon=@cxmcon;
 P.solver='fmincon';
 P.options=optimset;
 P.objective=@(x)(x(1)+x(2)+x(3));
 ff=optimset;
 P.lb=[100;1000;1000;10;10;10;10;10];P.ub=[10000;10000;10000;1000;1000;100
 0;1000;1000];
 P.x0=[0;000;000;0;0;0;0;0];
 ff.TolX=1e-30;ff.TolFun=1e-30;P.options=ff;
 [x,f1,flag]=fmincon(P)
```

```
x =

  1.0e+003 *

    0.5793
    1.3600
    5.1100
    0.1820
    0.2956
    0.2180
    0.2864
    0.3956


f1 =

  7.0492e+003


flag =

     0
```

书上给出的优化结果：

Optimum solution: $\mathbf{X}^* = \{567 \quad 1357 \quad 5125 \quad 181 \quad 295 \quad 219 \quad 286 \quad 395\}^{\mathrm{T}}$, $f^* = 7049$

将上述两者的答案与书所给的答案进行比较，发现 MATLAB 自带的优化函数的得出来的结果与书本所给的答案相近，而用自己编的程序所算出来的答案与上述两者相差很大。具体原因分析将在下一节展开。

6 结果讨论：

在计算各例子的时候，发现对于不同的初值，采用自己所编的程序时 MATLAB 求出来的最优点会发生较大变动。

对于例 1，当给定初值 x0=[0,0]’时，结果如下，MATLAB 并不能给出最后的结果。

```
>> x0=[0 0]';
mu0=[0]';
lam0=[0 0]';
[x,mu,lam,val,k]=sqpm(x0,mu0,lam0)
Warning: Matrix is singular to working precision.
```

对于例 2，当给定初值较小时，有时会有警告，但也能计算出结果，当使用 x0=[100,100]'时和例 1 一样，MATLAB 并不能给出最后的结果。

对于例 3，也有同样的现象，

```
>> x0=[1200 1000 4000 200 200 200 200 380]';
mu0=[ ]';%等式个数
lam0=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';%不等式个数
[x,mu,lam,Val,k]=sqpm(x0,mu0,lam0)

x =

  1.0e+003 *

   0.09999737695083
   1.16553651241568
   6.33079172529323
   0.09099043618512
   0.24554326904396
   0.04142077678939
   0.25664627925859
   0.34601304380223

Val =

    7.596325614659751e+003


k =

        1000
```

由此，可见，当改变初值时，其计算出来的结果会相差很大，并且每次都达到最大循环次数 1000。

对于上述现象的原因解释：

自己所编的程序中有一个设定，就是最大循环次数为 1000 次，也就是他不可能无限次求解下去，第二，虽然所编程序是按照数学计算推导而来，但是，程序并不能完整地重现数学计算那般精确；此外，还有很多情况并未考虑进去。

对于例 3，由于例 3 有 8 个变量，且最优点的值都是上百，甚至还有 5000 之上的，根据例 3 的最优点值，结合上面两个例子易推出，自己所编 SQP 存在缺陷，对于较多的变量，或者当最优点比较大的时候，容易陷入死循环，最终，无法计算出结果。相对而言，MATLAB 自带优化函数毕竟是经过了几十年的发展，他们会根据用户反映的问题，逐个解决，逐步完善，因此，它不能解决的问题自然是比较少，也才会有这么多人都愿意使用 MATLAB。

7 总结：

对于自己所编的算法程序，基本能解决一部分简单的非线性规划问题。肯定的，它还存在很多问题，需要进一步改善，但也不是一朝一夕就能完成的。我想再复杂的程序，也是从漏斗百出小程序建立起来的。MATLAB 刚开始的时候，肯定只能解决一些基本初级的问题，但是随着对算法的改进，小程序之间的组合，所解决的东西越来越多。虽然这个 SQP 不是自己编的，是从书上抄的，不过这次作业给我的意义还是非常的大的。作为一个马上就要开始进入编程行业的我来说，我所需要学的东西实在是非常的多，我也知道编程不同于其他，需要把基础打扎实，才能在以后的编程生涯中，少犯错误，也只有这样，才能逐步成长。很惭愧，没有自己亲手编出这个程序来，一是自己懒，懒得去阅读新的东西，懒得去动手做一些实物，二是耐心的缺乏，耐心不足，导致我易急躁，不能静下心去把书从头到尾看一遍，就想看出来作者是怎么把这个程序搞出来的。希望自己能够改正不足，继续向前走去。

参考文献

[1]薛定宇等. 高等应用数学问题的 MATLAB 求解[M]. 清华大学出版社有限公司, 2004.

[2]李元科. 工程最优化设计[M]. 清华大学出版社, 2006.

[3]最优化方法及其 Matlab 程序设计[M]. 科学出版社, 2010.

[4]张奇. 非线性约束条件下 SQP 算法的研究[D]. 青岛: 青岛大学, 2008.

[5]石国春. 关于序列二次规划(SQP)算法求解非线性规划问题的研究[D]. 兰州大学, 2009.

[6]蒋莉. 修正 BFGS 公式在 SQP 算法中的应用[D]. 湖南大学, 2003.

[7]裴杰. 求解等式约束优化问题的基于拟牛顿校正的既约 Hessian SQP 方法[D]. 湖南大学, 2008.