最优化第三次作业

162050127 颜劭铭

2022 年 5 月 6 日

1 基础题

1.1 凸函数

1.1.1 证明 $f(x) = \log \sum_{k=1}^{n} \exp(x_k)$ 是凸函数

由题目可知：$f(x) = \log \sum_{k=1}^{n} \exp(x_k) = \log(e^{x_1} + e^{x_2} + \cdots + e^{x_n}), x \in \mathbf{R}^n$

因为对数的底数取多少并不影响函数 $f(X)$ 的凸性，所以为了方便求导，我们取底数为 e。

因此对 $f(x)$ 求一阶偏导：

$$\frac{\partial f}{\partial x_i} = \frac{e^{x_i}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}}$$

接着对 $f(x)$ 求二阶偏导：

当 $i \neq j$ 的时候：

$$\frac{\partial^2 f}{\partial x_i \partial y_i} = \frac{-e^{x_i} e^{x_j}}{(e^{x_1} + e^{x_2} + \cdots + e^{x_n})^2}$$

当 $i = j$ 的时候:

$$\frac{\partial^2 f}{\partial x_i \partial y_i} = \frac{e^{x_i}(e^{x_1} + e^{x_2} + \cdots + e^{x_n}) - e^{x_i} e^{x_j}}{(e^{x_1} + e^{x_2} + \cdots + e^{x_n})^2}$$

此时定义一个列向量：$z = [e^{x_1}, e^{x_2}, \cdots, e^{x_n}]^T$

因此,Hessian 矩阵为：

$$H=\begin{vmatrix} \frac{e^{x_1}(e^{x_1}+e^{x_2}+\cdots+e^{x_n})-e^{x_1}e^{x_1}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} & \frac{-e^{x_1}e^{x_2}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} & \cdots & \frac{-e^{x_1}e^{x_n}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} \\ \frac{-e^{x_2}e^{x_1}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} & \frac{e^{x_2}(e^{x_1}+e^{x_2}+\cdots+e^{x_n})-e^{x_2}e^{x_2}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} & \cdots & \frac{-e^{x_2}e^{x_n}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-e^{x_n}e^{x_1}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} & \frac{-e^{x_n}e^{x_2}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} & \cdots & \frac{e^{x_n}(e^{x_1}+e^{x_2}+\cdots+e^{x_n})-e^{x_n}e^{x_n}}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2} \end{vmatrix}$$

化简可得：

$$H=\frac{1}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2}\begin{bmatrix} e^{x_1}(e^{x_1}+e^{x_2}+\cdots+e^{x_n}) & 0 & \cdots & 0 \\ 0 & e^{x_2}(e^{x_1}+e^{x_2}+\cdots+e^{x_n}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{x_n}(e^{x_1}+e^{x_2}+\cdots+e^{x_n}) \end{bmatrix}$$
$$-\frac{1}{(e^{x_1}+e^{x_2}+\cdots+e^{x_n})^2}\begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}\begin{bmatrix} e^{x_1} & e^{x_2} & \cdots & e^{x_n} \end{bmatrix}$$

用 z 表示为：

$$H=\frac{1}{(1^Tz)^2}\left(1^Tz\left(diag(z)\right)-zz^T\right).$$

由二阶条件可得：当 $H\geqslant 0$ 时，即 H 为半正定矩阵时，$f(x)$ 为凸函数。
又 $\frac{1}{1^Tz}$ 一定 >0，所以我们定义矩阵 $K\in R^{n\times n}$ 为 $k=1^Tz\left(diag(z)\right)-zz^T$.
因此当且仅当对 $\forall v\in R^n, v^Tkv\geqslant 0$ 时，k 为半正定矩阵。
所以进行计算：

$$\begin{aligned} v^Tkv &= v^T\left(\left(1^Tz\right)\left(diag(z)\right)-zz^T\right)v \\ &= (1^Tz)v^Tdiagzv-v^Tzz^Tkv \\ &= \left(\sum_{i=1}^n z_i\right)\left(\sum_{i=1}^n v_i^2z_i\right)-\left(\sum_{i=1}^n v_iz_i\right)^2 \end{aligned}$$

因此令 $a=v_i\sqrt{z_i}, b=\sqrt{z_i}$，上式可以化成：

$$v^Tkv=\left(b^Tb\right)\left(a^Ta\right)-\left(a^Tb\right)^2$$

由 Cauchy-Schworz 不等式可得：$v^Tkv\geqslant 0$
因此可得 log-sum-up 函数为凸函数。
参考资料：中国科学技术大学研究生院制作课程：《最优化理论》
`https://www.bilibili.com/video/BV1es411u7pK?p=12`

1.2 梯度下降法

1.2.1 算法的收敛速率

已知 $f(\mathbf{x})=\frac{1}{2}\mathbf{x}^TQ\mathbf{x}+b^T\mathbf{x}$ 是正定二次函数，因此 $\nabla f(\mathbf{x_k})=Q\mathbf{x_k}+b$。

极小值点 $\mathbf{x}^*$ 应满足 $\nabla f(\mathbf{x}^*)=Q\mathbf{x}^*+b=0\Rightarrow\mathbf{x}^*=-Q^{-1}b$

假设新的迭代点为 $\mathbf{x}_{k+1}=\mathbf{x}_k-\alpha d_k$，其中由于使用最速下降法，$d_k=-\nabla f(\mathbf{x}_k)$

因此问题转化为考虑一维最优化问题 $\min\limits_{\alpha}h(\alpha)=f(\mathbf{x}_k-\alpha\nabla f(\mathbf{x}_k))$

将上式展开，可得：

$$
\begin{aligned}
\min_{\alpha}h(\alpha)&=\frac{1}{2}(x_k-\alpha\nabla f(\mathbf{x}_k))^TQ(x_k-\alpha\nabla f(\mathbf{x}_k))+b^T(x_k-\alpha\nabla f(\mathbf{x}_k)\\
&=\frac{1}{2}(x_k^T-\alpha\nabla f(\mathbf{x}_k))(Qx_k-\alpha\nabla Qf(\mathbf{x}_k))+b^Tx_k-\alpha b^T\nabla f(\mathbf{x}_k)\\
&=\frac{1}{2}x_k^TQx_k-\frac{1}{2}\alpha x_k^TQ\nabla f(\mathbf{x}_k)-\frac{1}{2}\alpha\nabla f(\mathbf{x}_k)^TQx_k+\frac{1}{2}\alpha^2\nabla f(\mathbf{x}_k)^TQ\nabla f(\mathbf{x}_k)+b^Tx_k-\alpha b^T\nabla f(\mathbf{x}_k)\\
&=\frac{1}{2}\alpha^2\nabla f(\mathbf{x}_k)^TQ\nabla f(\mathbf{x}_k)-\alpha\nabla f(\mathbf{x}_k)^T(Qx_k+b)+\frac{1}{2}x_k^TQx_k+b^Tx_k)\\
&=\frac{1}{2}\alpha^2\nabla f(\mathbf{x}_k)^TQ\nabla f(\mathbf{x}_k)-\alpha\nabla f(\mathbf{x}_k)^T\nabla f(\mathbf{x}_k)+f(x_k))\\
&=\frac{1}{2}\|\nabla f(x_k)\|_Q^2\alpha^2-\|\nabla f(x_k)\|^2\alpha_t+f(x_k)
\end{aligned}
$$

这是一个关于变量 α 的二次函数。

接下来用精确线搜索确定步长：

$$h^{'}(\alpha)=\frac{\partial h}{\partial\alpha}=\alpha\|\nabla f(x_k)\|_Q^2-\|\nabla f(x_k)\|^2$$

令 $h^{'}(\alpha)=0$，可得：

$$\alpha_k=\frac{\|\nabla f(x_k)\|^2}{\|\nabla f(x_k)\|_Q^2}$$

因此下一个迭代点为：$\mathbf{x}_{k+1}=\mathbf{x}_k-\alpha d_k=\mathbf{x}_k-\frac{\|\nabla f(x_k)\|^2}{\|\nabla f(x_k)\|_Q^2}\nabla f(x_k)$

令 $g(k)=\nabla f(x_k)$，所以 $\alpha_k=\frac{g_k^TQg_k}{g_k^Tg_k}$

因此下一个迭代点的函数值为 $f(x_{k+1})=f(x_k)-\alpha_kg_k^Tg_k+\frac{1}{2}\alpha^2g_k^TQg_k=f(x_k)-\frac{1}{2}\frac{(g_k^Tg_k)^2}{g_k^TQg_k}$

因为 $f(x^*)=\frac{1}{2}b^TQ^{-1}b$

所以可得：

$$
\begin{aligned}
\frac{f(x_{k+1})-f(x^*)}{f(x_k)-f(x^*)} &= \frac{f(x_k)-\frac{1}{2}\frac{(g_k^Tg_k)^2}{g_k^TQg_k}+\frac{1}{2}b^TQ^{-1}b}{f(x_k)+\frac{1}{2}b^TQ^{-1}b} \\
&= 1+\frac{-\frac{1}{2}\frac{(g_k^Tg_k)^2}{g_k^TQg_k}}{\frac{1}{2}x_k^TQx_k+b^Tx_k+\frac{1}{2}b^TQ^{-1}b} \\
&= 1-\frac{\frac{1}{2}\frac{(g_k^Tg_k)^2}{g_k^TQg_k}}{\frac{1}{2}(Qx_k+b)^TQ^{-1}(Qx_k+b)} \\
&= 1-\frac{\frac{1}{2}\frac{(g_k^Tg_k)^2}{g_k^TQg_k}}{\frac{1}{2}g_k^TQ^{-1}g_k} \\
&= 1-\frac{(g_k^Tg_k)^2}{(g_k^TQg_k)(g_k^TQ^{-1}g_k)}
\end{aligned}
$$

因为 Q 为正定矩阵，所以由 Kantorovich 不等式可得：

$$\frac{(g_k^Tg_k)^2}{(g_k^TQg_k)(g_k^TQ^{-1}g_k)} \geqslant \frac{4\lambda_1\lambda_d}{(\lambda_1+\lambda_d)^2}, \text{其中}\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_d > 0, \lambda_i\text{为 Q 的特征值}$$

所以代入原式可得：

$$\frac{f(x_{k+1})-f(x^*)}{f(x_k)-f(x^*)} \leqslant 1-\frac{4\lambda_1\lambda_d}{(\lambda_1+\lambda_d)^2} = \left(\frac{(\lambda_1-\lambda_d)}{(\lambda_1+\lambda_d)}\right)^2$$

由于 $Q = diag(1,2,10)$，可得：$\lambda_1 = 10, \lambda_d = 1$，所以

$$\left(\frac{(\lambda_1-\lambda_d)}{(\lambda_1+\lambda_d)}\right)^2 = \frac{81}{121}$$

因此收敛速率

$$\frac{f(x_{k+1})-f(x^*)}{f(x_k)-f(x^*)} \leqslant \frac{81}{121}$$

1.2.2 要使 $f(x_T) - f(x^*) < 10^{-10}$，大约需要多少次迭代

因为 $b = (1,0,0)^T$，初始点 $x_0 = (1,1,1)^T, Q = diag(1,2,10)$，而且

$$\frac{f(x_{k+1})-f(x^*)}{f(x_k)-f(x^*)} \leqslant \left(1-\frac{2}{1+\kappa}\right)^2$$

可得：$\kappa = \frac{\lambda_1}{\lambda_d} = 10$

所以，对于式子进行化简可得：

$$f(x_{k+1})-f(x^*)\leqslant\left(1-\frac{2}{1+\kappa}\right)^2(f(x_k)-f(x^*))=\left(1-\frac{2}{1+\kappa}\right)^{2k}(f(x_0)-f(x^*))$$

令 $R=f(x_0)-f(x^*)$

要计算 $f(x_T)-f(x^*)<10^{-10}$ 的迭代次数，即计算 $\left(1-\frac{2}{1+\kappa}\right)^{2(T-1)}R<10^{-10}$

接下来计算 $f(x_0)$ $f(x^*)$:

$$f(x_0)=\frac{1}{2}(1,1,1)diag(1,2,10)(1,1,1)^T+(1,0,0)(1,1,1)^T=\frac{15}{2}$$
$$f(x^*)=-\frac{1}{2}(1,0,0)diag(1,2,10)(1,0,0)^T=-\frac{1}{2}$$

所以 R=8

所以转化为计算：

$$\left(1-\frac{2}{11}\right)^{2t}8<10^{-10}$$

将其转化为对数形式进行计算可得：$t\geqslant 63$

所以大约需要 63 次迭代。

1.3 强凸与光滑问题判定

1.3.1 计算 f(x) 相对于 x 的梯度

由题意可得：

$$f(x)=\frac{1}{2}\|Ax-y\|^2=\frac{1}{2}(Ax-y)^T(Ax-y)=\frac{1}{2}(x^TA^T-y^T)(Ax-y)$$
$$=\frac{1}{2}\left(x^TA^TAx-x^TA^Ty-y^TAx+y^Ty\right)$$

因此要求 f(x) 相对于 x 的梯度可转化为求 $\frac{\partial f}{\partial x}$

因为

$$\frac{\partial x^TAx}{\partial x}=(A+A^T)x,\frac{\partial A^Tx}{\partial x}=A,\frac{\partial x^TA}{\partial x}=A$$

所以对上式进行化简：

$$\frac{\partial f}{\partial x}=\frac{1}{2}\left(\left(A^TA+\left(A^TA\right)^T\right)x-A^Ty-A^Ty\right)=\frac{1}{2}\left(2A^TAx-2A^Ty\right)=A^T(Ax-y)$$

所以 $\nabla f(x)=A^T\left(Ax-y\right)$

1.3.2 设 $A = diag(-1, 5)$，证明 f(x) 是强凸函数，计算参数 μ

要证 f(x) 是强凸函数，只需构造 $g(x) = f(x) - \frac{1}{2}\|x\|^2$ 并证明其为凸函数
证明：

$$
\begin{aligned}
g(x) &= f(x) - \frac{1}{2}\|x\|^2 \\
\nabla g(x) &= A^T(Ax - y) - x \\
\nabla g(x)^2 &= A^T A - I = \begin{bmatrix} 0 & 0 \\ 0 & 24 \end{bmatrix} \geqslant 0
\end{aligned}
$$

因此，f(x) 为强凸函数
若 f(x) 是强凸函数且二阶连续可微，那么 $\nabla^2 f(x) \geqslant \mu I$
因为 $\nabla^2 f(x) = \frac{\partial \nabla f(x)}{\partial x} = AA^T$
因此可以转化为：$AA^T \geqslant \mu I$
因为 $A = diag(-1, 5)$，所以：

$$
AA^T = \begin{bmatrix} 1 & 0 \\ 0 & 25 \end{bmatrix} \geqslant \mu I
$$

因此可得：$\mu = 1$

1.3.3 设 $A = diag(-1, 5)$，证明 f(x) 是光滑函数，计算光滑函数的参数 L

要证 f(x) 是光滑函数，只需构造 $g(x) = \frac{25}{2}\|x\|^2 - f(x)$ 并证明其为凸函数
证明：

$$
\begin{aligned}
g(x) &= \frac{25}{2}\|x\|^2 - f(x) \\
\nabla g(x) &= 25x - A^T(Ax - y) \\
\nabla g(x)^2 &= 25I - A^T A = \begin{bmatrix} 24 & 0 \\ 0 & 0 \end{bmatrix} \geqslant 0
\end{aligned}
$$

因此，f(x) 为光滑函数
若 f(x) 是光滑函数且二阶连续可微，那么 $\nabla^2 f(x) \leqslant LI$
因为 $\nabla^2 f(x) = \frac{\partial \nabla f(x)}{\partial x} = AA^T$
因此可以转化为：$AA^T \leqslant LI$
因为 $A = diag(-1, 5)$，所以：

$$AA^T = \begin{bmatrix} 1 & 0 \\ 0 & 25 \end{bmatrix} \leqslant LI$$

因此可得：$L = 25$

1.3.4 设 $A = diag(1,2)$，并且步长为 $\alpha_k = \frac{1}{4}$，求算法的单步下降幅度 $f(x_{k+1}) - f(x_k)$

由光滑函数的定义：

$$f(y) \leqslant f(x) + \nabla f^T(x)(y-x) + \frac{L}{2}\|y-x\|^2$$

因为步长：$x_{k+1} = x_k - \frac{1}{4}\nabla f(x_k) = x_k - \frac{1}{4}\left[A^T\left(Ax-y\right)\right]$

那么对于上式进行化简可得：

$$\begin{aligned} f(x_{k+1}) &\leqslant f(x_k) + \nabla^T f(x_k)(x_{k+1}-x_k) + \frac{L}{2}\|x_{k+1}-x_k\|^2 \\ &= f(x_k) - \frac{1}{4}\|\nabla f(x_k)\|^2 + \frac{1}{2L}\|\nabla f(x_k)\|^2 \end{aligned}$$

由 1.3.3 可得：因为 f(x) 是光滑函数，所以 $\nabla^2 f(x) \leqslant LI$

因为 $A = diag(1,2)$，所以 $\nabla^2 f(x) = AA^T = diag(1,4) \leqslant LI \Rightarrow L = 4$

所以可得：

$$f(x_{k+1}) - f(x_k) \leqslant -\frac{1}{4}\|\nabla f(x_k)\|^2 + \frac{1}{8}\|\nabla f(x_k)\|^2 = -\frac{1}{8}\|\nabla f(x_k)\|^2$$

因此算法的单步下降幅度 $f(x_{k+1}) - f(x_k) = \frac{1}{8}\|\nabla f(x_k)\|^2$

1.3.5 设 $A = diag(1,2)$，算法使用固定步长，问如果要使迭代序列是下降序列，步长应满足什么条件

设要使用固定步长为 α，且 A=diag(1,2)，由 1.3.4 可得：

$$f(x_{K+1}) - f(x_k) \leqslant -(\alpha - \frac{1}{8})\|\nabla f(x_k)\|^2$$

要保证为下降序列，可转化为：$\alpha - \frac{1}{8} > 0$

因此步长 $\alpha > \frac{1}{8}$

1.3.6 设 $A = diag(1,2)$，如果 $f(x_0) - f(x^*) = 1$，要使 $f(x_k) - f(x^*) < 10^{-3}$，试估计需要多少次迭代

由 3.1 ∼ 3.5 的证明可得:f(x) 是光滑且强凸的,并且它的 Hessian 矩阵满足 $\mu I \leqslant \nabla^2 f(x) \leqslant LI$

因此梯度下降法的收敛速率为 $(1-\frac{\mu}{L})$

$$A = diag(1,2) \Rightarrow \mu I \leqslant \nabla^2 f(x) = AA^T = diag(1,4) \leqslant LI \Rightarrow \mu = 1, L = 4$$
$$\Rightarrow 1 - \frac{\mu}{L} = \frac{3}{4}$$

又因为 $f(x_0) - f(x^*) = 1$

所以要求迭代次数，可转化为：

$$\begin{aligned} f(x_k) - f(x^*) = (\frac{3}{4})^k f(x_0) - f(x^*) < 10^{-3} \\ \Rightarrow & k\left(\ln 3 - \ln 4\right) < -3\ln 10 \\ \Rightarrow & k > 24.0118 \end{aligned}$$

因此大约需要 25 次迭代.

1.4 共轭梯度法性质

1.4.1 求 $x^{(1)}$ 处的搜索方向

因为在共轭梯度法中，梯度等于残差，并且由题意得：$f : \mathbf{R}^3 \to \mathbf{R}$，

$$\left.\frac{\partial f(x)}{\partial x_1}\right|_{x^{(1)}} = \left.\frac{\partial f(x)}{\partial x_2}\right|_{x^{(1)}} = -2$$

因此可得：(其中 a 是由于我们不知道残差 r_1 的第三个分量，所以进行假设)

$$r_1 = \begin{bmatrix} -2 \\ -2 \\ a \end{bmatrix}, d_0 = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}, r_0 = -d_0 = \begin{bmatrix} -1 \\ 1 \\ -2 \end{bmatrix}$$

因为如下公式：

$$\alpha_0 = \frac{r_0^T r_0}{d_0^T G d_0}, r_1 - r_0 = \alpha_0 G d_0, \beta_0 = \frac{r_1^T r_1}{r_0^T r_0}, d_1 = -r_1 + \beta_0 d_0$$

所以我们可以设：

$$\alpha_0 G d_0 = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

进行如下计算：

$$d_0^T \cdot \alpha_0 G d_0 = r_0^T r_0 = \|r_0\|^2 = 6 \Rightarrow a_1 - a_2 + 2a_3 = 6$$

结合 $r_1 - r_0 = alpha_0 G d_0$，我们可以得到如下式子：

$$\begin{cases} -2+1 = a_1 \\ -2-1 = a_2 \\ a+2 = a_3 \\ a_1 - a_2 + 2a_3 = 6 \end{cases}$$

解得：

$$a_1 = -1, a_2 = -3, a_3 = 2, a = 0, \alpha_0 G d_0 = \begin{bmatrix} -1 \\ -3 \\ 2 \end{bmatrix}, r_1 = \begin{bmatrix} -2 \\ -2 \\ 0 \end{bmatrix}$$

接着进行计算可得：

$$\beta_0 = \frac{4}{3}, d_1 = -r_1 + \beta_0 d_0 = \begin{bmatrix} \frac{10}{3} \\ \frac{2}{3} \\ \frac{8}{3} \end{bmatrix}$$

因此可得到 $x^{(1)}$ 处的搜索方向 d_1。

1.4.2 如果在 $x^{(2)}$ 处梯度长度为 $\|\nabla f(x^{(2)})\| = 1$，求 $\nabla f(x^{(2)})$

已知：

$$r_1 = \begin{bmatrix} -2 \\ -2 \\ 0 \end{bmatrix} d_1 = \begin{bmatrix} \frac{10}{3} \\ \frac{2}{3} \\ \frac{8}{3} \end{bmatrix}$$

由共轭梯度法可得：残差等与梯度，因此要求 $\nabla f(x^{(2)})$ 可转化为求 r_2，设 $r_2 = [b_1, b_2, b_3]^T$

因为 $\|\nabla f(x^{(2)})\| = 1 \Rightarrow \|r_2\| = 1 \Rightarrow b_1^2 + b_2^2 + b_3^2 = 1$

因为 r_2 与已有共轭方向 d_0 d_1 正交，所以对以上式子进行联立可得：

$$\begin{cases} b_1 - b_2 + 2b_3 = 0 \\ 5b_1 + b_2 + 4b_3 = 0 \\ b_1^2 + b_2^2 + b_3^2 = 1 \end{cases}$$

可得：

$$b_1 = -\frac{1}{\sqrt{3}}, b_2 = \frac{1}{\sqrt{3}}, b_3 = \frac{1}{\sqrt{3}}$$

所以可得：

$$\nabla f(x^{(2)}) = \left[-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right]^T$$

1.5 共轭梯度法的等价性

1.5.1 给定下降方向 d，计算精确线搜索确定的步长

已知：$f(x) = \frac{1}{2}x^TQx + b^Tx + c$，下降方向为 d，因此精确线搜索确定的步长可以转化为:$\alpha^* = \arg\min_{\alpha} f(x_k + \alpha d)$，令 $g = -d$

$$\begin{aligned}
\alpha^* &= \arg\min_{\alpha} f(x_k - \alpha g) \\
&= \arg\min_{\alpha} (\frac{1}{2}(x_k - \alpha g)^T Q (x_k - \alpha g) + b^T(x_k - \alpha g) + c) \\
&= \arg\min_{\alpha} (\frac{1}{2}(x_k^T - \alpha g^T)(Qx_k - \alpha Qg) + b^T x_k - \alpha b^T g + c) \\
&= \arg\min_{\alpha} (\frac{1}{2}x_k^T Q x_k - \frac{1}{2}\alpha x_k^T Q g - \frac{1}{2}\alpha g^T Q x_k + \frac{1}{2}\alpha^2 g^T Q g + b^T x_k - \alpha b^T g + c) \\
&= \arg\min_{\alpha} (\frac{1}{2}\alpha^2 g^T Q g - \alpha g^T (Qx_k + b) + \frac{1}{2}x_k^T Q x_k + b^T x_k + c) \\
&= \arg\min_{\alpha} (\frac{1}{2}g^T Q g \alpha^2 - g^T g \alpha + f(x_k))
\end{aligned}$$

可以看出上述函数是关于 α 的二次函数

所以精确线搜索确定的步长为：

$$\alpha^* = \frac{g^T g}{g^T Q g} = \frac{d^T d}{d^T Q d}$$

1.5.2 证明，在上述问题上，使用精确线搜索确定步长时，三种共轭梯度法所确定的点列 $\{x_i\}$ 是相同的

由共轭梯度法的计算过程可以看出三种方法的差别主要在为 β 的迭代公式不同，因此要证明三种共轭梯度法所确定的点列是否相同，即证明 β 迭代公式的等价性。

这边给出 PPT 上的六种 β 迭代公式，其中前三种已经在 PPT 上给出证明，因此这里主要对于后三种进行证明：

$$\begin{aligned}
&\beta_{k-1}^{HS}=\frac{g_k^T G d_{k-1}}{d_{k-1}^T G d_{k-1}} \quad (\text{ Hestenes } - \text{ Stiefel 公式}) \\
&\beta_{k-1}^{CW}=\frac{g_k^T\left(g_k-g_{k-1}\right)}{d_{k-1}^T\left(g_k-g_{k-1}\right)} \quad (\text{Crowder - Wolfe 公式}) \\
&\beta_{k-1}^{FR}=\frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \quad (\text{ Fletcher } - \text{ Reeves 公式}) \\
&\beta_{k-1}^{PRP}=\frac{g_k^T\left(g_k-g_{k-1}\right)}{g_{k-1}^T g_{k-1}} \quad (\text{ Polak - Ribiere - Polyak 公式}) \\
&\beta_{k-1}^{D}=-\frac{g_k^T g_k}{d_{k-1}^T g_{k-1}} \quad (\text{ Dixon 公式}) \\
&\beta_{k-1}^{DY}=\frac{g_k^T g_k}{d_{k-1}^T\left(g_k-g_{k-1}\right)} \quad (\textbf{Dai} - \textbf{Yuan} \text{ 公式})
\end{aligned}$$

已知：$d_k^T g_{k+1}=-g_k^T g_{k+1}^T+\beta_{k-1} d_{k-1}^T g_{k+1}$，由子空间扩展定理：$d_k^T g_{k+1}=0, d_{k-1}^T g_{k+1}=0$，可得：$g_{k+1}^T g_k=0$

由梯度的正交性 $g_k^T d_{k-1}=0$，因此可由 Crowder-Wolfe 公式推出 PRP 公式。

因为 $d_{k-1}^T g_{k-1}=-g_{k-1}^T g_{k-1}$，代入 Crowder-Wolfe 公式可以推出 Dixon 公式。

由梯度的正交性：$g_k^T g_{k-1}=0$，可以由 Crowder-Wolfe 公式推出 Dai-Yuan 公式。

1.6 共轭梯度法的子空间性质与收敛性

1.6.1 每次迭代的搜索方向 $\mathbf{d}_{\mathrm{i}}$ 都可以写成下面向量组的线性组合

$$\left\{\mathbf{d}, Q \mathbf{d}, Q^2 \mathbf{d}, \cdots, Q^i \mathbf{d}\right\}$$

由子空间扩展定理可得：每次迭代 g_k 与整个子空间 $span\{d_0, d_1, \cdots, d_{k-1}\}$ 正交，与之前搜索过的所有方向均相互独立，并且正交。

因为 $r_i=g_i=Q x_i-b$，可得：

$$
\begin{aligned}
r_{i+1} = g_{i+1} = Qx_{i+1} - b &= Q(x_i + \alpha_i d_i) - b \\
&= Qx_i - b + \alpha_i Q d_i \\
&= r_i + \alpha_i Q d_i
\end{aligned}
$$

因此要证明每次搜索方向 d_i 可以写成 $\{d, Qd, Q^2d, \cdots .Q^i d\}$ 的线性组合，可用数学归纳法证明如下：

当 $i = 0$ 时，$d_0 = g_0$，易得：显然 d_0 可以写成 d_0 自己的线性组合。

假设 $r_i, d_i \in span\{d, Qd, Q^2d, \cdots, Q^i d\}$

则由上式证明可得：

$$
r_{i+1} = r_i + \alpha_i Q d_i \in span\{d, Qd, Q^2d, \cdots, Q^i d, Q^{i+1} d\}
$$

$$
d_{i+1} = r_{i+1} + \beta d_i \in span\{d, Qd, Q^2d, \cdots, Q^i d, Q^{i+1} d\}
$$

因此可得每次搜索方向 d_i 可以写成 $\{d, Qd, Q^2d, \cdots .Q^i d\}$ 的线性组合。

1.6.2 对于第 $i+1$ 次迭代的序列 x_{i+1}，证明，对于任意由上述向量线性组合的向量 $x^{'}$，都有 $f(x^{'}) \geqslant f(x_{i+1})$

由共轭梯度法可得：

$$
\begin{aligned}
x_{i+1} &= x_i + \alpha_i d_i \\
&= x_0 + \sum_{k=0}^{i} \alpha_k d_k = x_0 + \begin{bmatrix} \alpha_0 & \alpha_1 & \cdots & \alpha_i \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_i \end{bmatrix} \\
&= x_0 + A^T D
\end{aligned}
$$

令子空间：

$$
\begin{aligned}
V &= x_0 + span\{d, Qd, Q^2d, \cdots, Q^i d\} \\
&= x_0 + span\{d, d_1, d_2, \cdots, d_i\}
\end{aligned}
$$

所以对于任意迭代点 $x^{'}$，可转化为：

$$
\begin{aligned}
x^{'} &= x_0 + \begin{bmatrix} c_0 & c_1 & \cdots & c_i \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_i \end{bmatrix} \\
&= x_0 + c^T D \in V
\end{aligned}
$$

因此，要证明对于任意由上述向量线性组合的向量 $x^{'}$，都有 $f(x^{'}) \geqslant f(x_{i+1})$，可以转化为求 $\arg\min_{x^{'} \in V} f(x^{'})$

即 $\nabla f(x^{'}) = \nabla f(x_0 + c^T D)^T D = 0$

由共轭的条件可知：$\nabla f(x_{i+1})^T D = 0$

即 $x^{'} = x_{i+1}$ 的时候有最小值

因此 $f(x^{'}) \geqslant f(x_{i+1})$

1.6.3 使用上述结论证明，共轭梯度法在二次函数 $f(x)$ 上迭代 K 次即可收敛到最优点，其中 K 是矩阵 Q 的不同的特征值的数目

通过翻阅参考资料《Numerical Optimization》可知：

$$
\|x_{k+1} - x^*\|_Q^2 \leq \min_{P_k} \max_{1 \leq i \leq n} \left[1 + \lambda_i P_k(\lambda_i)\right]^2 \|x_0 - x^*\|_Q^2
$$

假设 Q 的 N 个特征值只能取 k 个互不相同的值且满足 $k_1 < k_2 < \cdots < k_k$

那么令：

$$
Q_k(\lambda) = \frac{(-1)^k}{k_1 k_2 \cdots k_k} (\lambda - k_1)(\lambda - k_2) \cdots (\lambda - k_k)
$$

对上式进行进一步化简变换可得：

$$
\begin{aligned}
& Q_k(\lambda_i) = 0 \quad i = 1, 2, \cdots, k \\
& Q_k(0) = 1 \\
& Q_k(\lambda) - 1 \text{ 为k 阶多项式}
\end{aligned}
$$

因此可得：

$$
\bar{P}_{k-1}(\lambda) = \frac{Q_k(\lambda) - 1}{\lambda} \text{ 为k} - 1 \text{ 阶多项式}
$$

所以可得出以下结论：

$$0 \leq \min_{P_{k-1}} \max_{1 \leq i \leq n} \left[1+\lambda_i P_{k-1}\left(\lambda_i\right)\right]^2 \leq \max_{1 \leq i \leq n} \left[1+\lambda_i \bar{P}_{k-1}\left(\lambda_i\right)\right]^2 = \max_{1 \leqslant i \leqslant n} Q_k^2\left(\lambda_i\right) = 0$$

因此可证明：k 次迭代后 $\|x_k - x^*\|_Q^2 = 0$
所以，$x_k = x^*$，即共轭梯度法在二次函数 $f(x)$ 上迭代 K 次即可收敛到最优点.

1.7 约束优化问题的 KKT 条件

1.7.1 写出其 KKT 条件，并分析 KKT 点

题目可进行以下转化：

$$\begin{cases} \arg\max\limits_{v} \quad tr(v^T S v) \\ s.t. \quad v^T v = I_k \end{cases}$$

构造拉格朗日函数 $L(v,\lambda) = tr(v^T S v) + \lambda(I_k - v^T v)$
所以求一阶导可得：

$$\nabla_v L(v,\lambda) = (S+S^T)v - 2\lambda v, \quad \nabla_\lambda = I_k - v^T v$$

因此令 $\nabla_v L(v,\lambda) = 0$，可得：

$$(S+S^T)v - 2\lambda v = 0 \Rightarrow \lambda v^* = \frac{S+S^T}{2} v^*$$

即 v^* 是矩阵 $\frac{S+S^T}{2}$ 的对应于特征值 λ 的特征向量。
由 $\lambda v^* = \frac{S+S^T}{2} v^*$，等式两边左乘 v^{*T}，并有 $v^T v = I_k$ 可得：

$$\lambda^* = v^{*T} (\frac{S+S^T}{2}) v^*$$

因此 λ^* 为 $f(v) = v^T(\frac{S+S^T}{2})v$ 的最大值。

1.7.2 如果使用投影梯度法进行求解上述问题，写出投影公式

投影公式如下：

- 如果 $v_t + \alpha_t d_t \in V$, 则 $v_{t+1} = v_t + \alpha_t d_t$
- 如果 $v_t + \alpha_t d_t \notin V$, 则 $v_{t+1} = arg\min\limits_{x \in V} \|x - (v_t + \alpha_t + d_t)\|$

2 思考题

2.1 共轭方向与特征向量

2.1.1 证明：方向组 $1, \cdots, n$ 是关于 Q 相互共轭的

依题意得：Q 是正定对称矩阵，并且其特征分解为 $Q = U\Lambda U^T$，且特征值满足 $\lambda_1 > \lambda_2 > \cdots > \lambda_n > 0$，其对应的特征向量分别为 $1, 2, \cdots, n$.

由对称矩阵特征分解的定理可得：矩阵 $U = [u_1, u_2, \cdots, u_n]$ 为正交矩阵，且满足 $u_i^T u_j = \delta_{ij}$，其中当 i=j 的时候为 1，否则为 0.

因此要证明方向组 $1, \cdots, n$ 是关于 Q 相互共轭的，可将方向组用 $u_1, u_2, \cdots, u_n$ 表示，对于 $i \neq j$：

$$
\begin{aligned}
u_i^T Q u_j &= u_i^T \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \cdots & \\ & & & \lambda_n \end{pmatrix} u_j \\
&= u_i^T \begin{pmatrix} 0 \\ \cdots \\ \lambda_j \\ \cdots \\ 0 \end{pmatrix} \\
&= 0
\end{aligned}
$$

这满足共轭的定义，因此得证：方向组 $1, \cdots, n$ 是关于 Q 相互共轭的。

2.1.2 对于任意 $k = 1, \cdots, n-1$，求单步改进幅度 $f(x_{k+1}) - f(x_k)$

由题意得：$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \beta_k k$，其中 k 指的是矩阵 Q 的特征向量 λ_i，且由第一题得到特征向量是共轭方向组，因此可得：$k = -\nabla f(x_k)$，所以直接展开进行计算：

其中涉及到的有 2.1.1 的共轭方向组的性质以及对称矩阵的特征分解一些性质，例如当 i=j 的时候，$u_i u_j = 1$。

$$
\begin{aligned}
f(x_{k+1})-f(x_k) &= f(x_k-\beta_k k)-f(x_k)\\
&= \frac{1}{2}(x_k-\beta_k k)^TQ(x_k-\beta_k k)+b^T(x_k-\beta_k k)-\frac{1}{2}x_K^TQx_k-b^Tx_k\\
&= \frac{1}{2}(x_k^TQ-\beta_k k^TQ)(x_k-\beta_k k)+b^Tx_k-b^T\beta_k k-\frac{1}{2}x_K^TQx_k-b^Tx_k\\
&= \frac{1}{2}(x_k^TQx_k)-\frac{1}{2}\beta_k x_k^TQk-\frac{1}{2}\beta_k k^TQx_k+\frac{1}{2}\beta_k^2k^TQk-\frac{1}{2}x_kQ^Tx_k-\beta_k b^Tk\\
&= \frac{1}{2}\beta_k^2k^TQk-\beta_k k^T(Qx_k+b)\\
&= \frac{1}{2}\beta_k^2k^T\lambda_k k+\beta_k\nabla f^T(x_k)\nabla f(x_k)\\
&= \frac{1}{2}\beta_k^2\lambda_k+\beta_k\|\nabla f(x_k)\|^2
\end{aligned}
$$

因此单步改进幅度为：$\frac{1}{2}\beta_k^2\lambda_k+\beta_k\|\nabla f(x_k)\|^2$

2.1.3 若对于 $k=1,\cdots,n-1$，每一次迭代的改进幅度 $\delta_k=|f(x_{k+1})-f(x_k)|$ 都相等，则步长 β_k 满足什么条件

结合 2.1.2 可得，因为每一次迭代的改进幅度 $\delta_k=|f(x_{k+1})-f(x_k)|$ 都相等

$$\frac{1}{2}\beta_k^2\lambda_k+\beta_k\|\nabla f(x_k)\|^2=\frac{1}{2}\beta_{k+1}^2\lambda_{k+1}+\beta_{k+1}\|\nabla f(x_{k+1})\|^2$$

写到这里写不下去了，这个式子不知道该怎么化简。

3 编程题

3.1 使用 Newton-CG 与 L-BFGS 法求解 LASSO 问题

3.1.1 Newton-CG 法

用 python 实现的 Newton-CG 法代码：

```
# 导入相关包

import numpy as np
import matplotlib.pyplot as plt

# 写入目标函数

def testFun(x, y):
```

```python
    t = 4.0*x - 1.0*y
    t1 = x - 1.0
    z = np.power(t, 2) + np.power(t1, 4)
    return z

# 求函数的梯度

def gradTestFun(x, y):
    delta_x = 1e-6      #x方向差分小量
    delta_y = 1e-6      #y方向差分小量
    grad_x = (testFun(x+delta_x, y)-testFun(x-delta_x, y))/(2.0*delta_x)
    grad_y = (testFun(x, y+delta_y)-testFun(x, y-delta_y))/(2.0*delta_y)
    grad_xy = np.array([grad_x, grad_y])
    return grad_xy

# 采用牛顿法，精确线性搜索确定移动步长

def getStepLengthByNewton(array_xy, array_d):
    a0 = 1.0            #初始猜测值
    e0 = 1e-6           #退出搜索循环的条件
    delta_a = 1e-6      #对a作差分的小量
    while(1):
        new_a = array_xy + a0*array_d
        new_a_l = array_xy + (a0-delta_a)*array_d
        new_a_h = array_xy + (a0+delta_a)*array_d
        diff_a0 = (testFun(new_a_h[0], new_a_h[1]) - testFun(new_a_l[0], new_a_l[1]))/(2.0*delta_a)
        if np.abs(diff_a0) < e0:
            break
        ddiff_a0 = (testFun(new_a_h[0], new_a_h[1]) + testFun(new_a_l[0], new_a_l[1]) - 2.0*testFun(new_a[0], new_a[1]))/(delta_a*delta_a)
        a0 = a0 - diff_a0/ddiff_a0
    return a0

# 可视化

def plotResult(array_xy_history):
    x = np.linspace(-1.0, 4.0, 100)
    y = np.linspace(-4.0, 8.0, 100)
    X, Y = np.meshgrid(x, y)
    Z = testFun(X, Y)
    plt.figure(dpi=300)
    plt.xlim(-1.0, 4.0)
    plt.ylim(-4.0, 8.0)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.contour(X, Y, Z, 40)
```

```
        plt.plot(array_xy_history[:,0], array_xy_history[:,1], marker='.', ms=10)
        xy_count = array_xy_history.shape[0]
        for i in range(xy_count):
            if i == xy_count-1:
                break
            dx = (array_xy_history[i+1][0] - array_xy_history[i][0])*0.6
            dy = (array_xy_history[i+1][1] - array_xy_history[i][1])*0.6
            plt.arrow(array_xy_history[i][0], array_xy_history[i][1], dx, dy, width=0.1)

    # 使用CG算法优化，用FR公式计算组合系数

    def mainFRCG():
        ls_xy_history = []                              #存储初始坐标的迭代结果
        xy0 = np.array([4.0, -2.0])                      #初始点
        grad_xy = gradTestFun(xy0[0], xy0[1])
        d = -1.0*grad_xy                                #初始搜索方向
        e0 = 1e-6                                       #迭代退出条件
        xy = xy0
        while(1):
            ls_xy_history.append(xy)
            grad_xy = gradTestFun(xy[0], xy[1])
            tag_reach = np.abs(grad_xy) < e0
            if tag_reach.all():
                break
            step_length = getStepLengthByNewton(xy, d)
            xy_new = xy + step_length*d
            grad_xy_new = gradTestFun(xy_new[0], xy_new[1])
            b = np.dot(grad_xy_new, grad_xy_new)/np.dot(grad_xy, grad_xy)        #根据FR公式
计算组合系数
            d = b*d - grad_xy_new
            xy = xy_new
        array_xy_history = np.array(ls_xy_history)
      plotResult(array_xy_history)
      return array_xy_history

    # 主函数

    mainFRCG()
```

选择目标函数后的可视化结果：

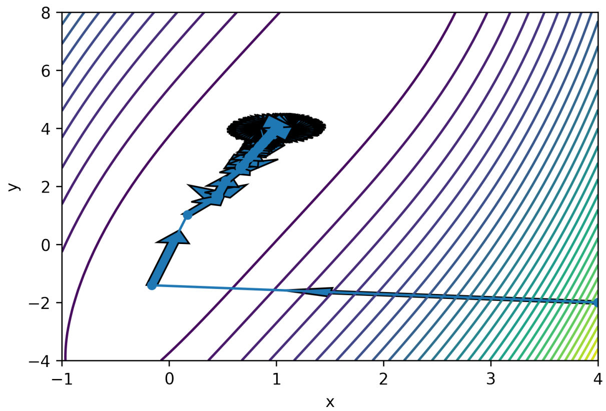

图 1: Newton-CG 可视化结果

3.1.2 应用牛顿-共轭梯度法解逻辑回归问题

看了那个题目提供的界面的代码后，发现里面的数据集是下载不下来的，只好换了个数据集进行运行，然后题目中的可视化纵坐标使用的是迭代点的梯度 2 范数，这个需要使用自动微分，不知道咋写，所以只能将纵坐标设为损失函数：所有模型误差的平方和进行可视化。

代码如下：

```
import numpy as np

# 读取文件的内容转化成数组的形式

def file2matrix(filename,length=2):
    fr=open(filename)
    lines=fr.readlines()
    numberoflines=len(lines)
    returnmat=np.zeros((numberoflines,length))
    index=0
    for line in lines:
        line=line.strip()
        listofline=line.split('    ')
        returnmat[index,:]=listofline
        index=index+1
    return returnmat

# 数据归一化

def normalizefeature(x):
    x_norm=x
    mu=np.mean(x,axis=0)
```

```python
    sigma=np.std(x,axis=0)
    x_norm=(x-mu)/sigma
    return x_norm,mu,sigma

# 数据的可视化

import matplotlib.pyplot as plt

def plot_data(*agrs):
    X = file2matrix('./ex4x.dat')
    y = file2matrix('./ex4y.dat', 1)

    X,_,_=normalizefeature(X)
    pos = list(np.where(y == 1.0)[0])
    X_pos = X[pos]
    neg = list(np.where(y == 0.0)[0])
    X_neg = X[neg]
    plt.plot(X_pos[:,0],X_pos[:,1],'+',label='admitted')
    plt.plot(X_neg[:,0],X_neg[:,1],'o',label='Not admitted')
    plt.xlabel("exam1 score")
    plt.ylabel("exam2 score")
    plt.legend()
    plt.show()

# 假设函数为Sigmoid函数

def sigmoid(z):
    res=1/(1+np.exp(-z))
    return res

# 计算代价的文件

def compute_loss(x,y,theta):
    loss=-np.sum((np.dot(y.T,np.log(sigmoid(np.dot(x,theta))))
    +np.dot((1-y).T,np.log(1-sigmoid(np.dot(x,theta)))))/x.shape[0])
    return loss

# 牛顿法进行参数的更新

a=np.diag(np.array([1,2]))
print(a)

def nt(x,y,theta,iterations):
    n,m=x.shape
    J_loss=[]
    orig_loss=np.inf
    for i in range(iterations):
```

```
            l=compute_loss(x,y,theta)
            J_loss.append(l)
            h=sigmoid(np.dot(x,theta))#(n,1)
            j_first_order=1/n*np.dot(x.T,h-y)#(m,1)
            j_second_order=1/n*np.dot(np.dot(np.dot(x.T,np.diag(h.reshape(n))),np.diag(1-h.
    reshape(n))),x)
            theta=theta-np.dot(np.linalg.inv(j_second_order),j_first_order)
    return theta,J_loss

    if __name__=="__main__":
        X = file2matrix('./ex4x.dat')
        y = file2matrix('./ex4y.dat', 1)

        n, m = X.shape
        X = np.column_stack((np.ones(n), X))
        m = m + 1
        theta = np.zeros((m, 1))

        theta, J_his = nt(X, y, theta, 5)

        print("theta", theta)
        print("Loss", J_his)
        plt.xlabel("iteration")
        plt.ylabel("Loss")
        plt.plot(np.arange(5), J_his)
        plt.show()

        pos = list(np.where(y == 1.0)[0])
        X_pos = X[pos,1:3]
        neg = list(np.where(y == 0.0)[0])
        X_neg = X[neg,1:3]
        plt.plot(X_pos[:, 0], X_pos[:, 1], '+', label='admitted')
        plt.plot(X_neg[:, 0], X_neg[:, 1], 'o', label='Not admitted')
        plt.xlabel("exam1 score")
        plt.ylabel("exam2 score")
        plt.legend()

        xx = np.linspace(20, 70, 6)
        yy = []
        for i in xx:
            res = (i * -(theta[1][0]) - (theta[0][0])) / (theta[2][0])
            yy.append(res)
        plt.plot(xx, yy)
        plt.show()
```

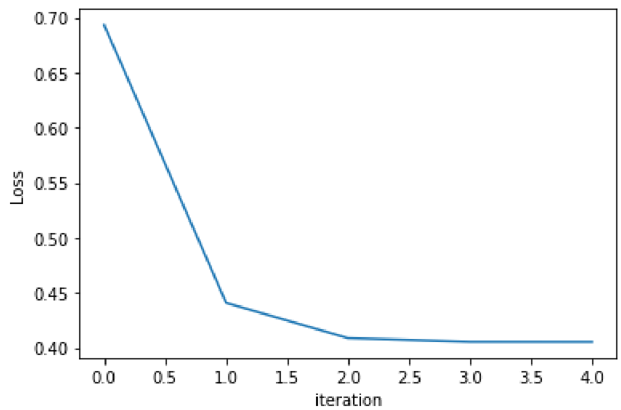

图 2: Newton-CG 的逻辑回归问题可视化结果

3.1.3 L-BFGS 算法的 MATLAB 实现

用 python 实现的 L-BFGS 算法代码：

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import empty
from numpy import dot

# 向量转化

def twoloop(s, y, rho,gk):

    n = len(s) #向量序列的长度

    if np.shape(s)[0] >= 1:
    h0 = 1.0*np.dot(s[-1],y[-1])/np.dot(y[-1],y[-1])
    else:
        h0 = 1

    a = empty((n,))

    q = gk.copy()
    for i in range(n - 1, -1, -1):
        a[i] = rho[i] * dot(s[i], q)
        q -= a[i] * y[i]
    z = h0*q

    for i in range(n):
        b = rho[i] * dot(y[i], z)
```

```
        z += s[i] * (a[i] - b)

    return z

# LBFGS

def lbfgs(fun,gfun,x0,m=5):
    # fun和gfun分别是目标函数及其一阶导数,x0是初值,m为储存的序列的大小
    maxk = 5000
    rou = 0.55
    sigma = 0.4
    epsilon = 1e-5
    k = 0
    W = np.zeros((2, 10 ** 3))
    n = np.shape(x0)[0]
    Bk = np.eye(n) # 初始对称正定矩阵, Bk=np.linalg.inv(hess(x0))
    W[:, 0] = x0

    s, y, rho = [], [], []

    while k < maxk :
        gk = gfun(x0)
        if np.linalg.norm(gk) < epsilon:
            break

        dk = -1.0*twoloop(s, y, rho,gk)

        m0=0;
        mk=0
        while m0 < 20: # 用Armijo搜索求步长
            if fun(x0+rou**m0*dk) < fun(x0)+sigma*rou**m0*np.dot(gk,dk):
                mk = m0
                break
            m0 += 1

        x = x0 + rou**mk*dk
        sk = x - x0
        yk = gfun(x) - gk

        if np.dot(sk,yk) > 0: #增加新的向量
            rho.append(1.0/np.dot(sk,yk))
            s.append(sk)
            y.append(yk)
        if np.shape(rho)[0] > m: #弃掉最旧向量
            rho.pop(0)
            s.pop(0)
            y.pop(0)
```

```python

            k += 1
            x0 = x

        W = W[:,0:k]  # 记录迭代点
        return x0,fun(x0),k,W#分别是最优点坐标，最优值，迭代次数

# 主函数及可视化

# 函数表达式fun
fun = lambda x:100*(x[0]**2-x[1])**2 + (x[0]-1)**2

# 梯度向量 gfun
gfun = lambda x:np.array([400*x[0]*(x[0]**2-x[1])+2*(x[0]-1), -200*(x[0]**2-x[1])])

# 海森矩阵 hess
hess = lambda x:np.array([[1200*x[0]**2-400*x[1]+2, -400*x[0]],[-400*x[0],200]])

X0 = np.arange(-1.5,1.5-0.05,0.05)
X1 = np.arange(-3.5,2+0.05,0.05)
[x0,x1] = np.meshgrid(X0,X1)
f=100*(x1-x0**2)**2+(1-x0)**2 # 给定的函数
plt.contour(x0,x1,f,20)


x0,fun0,k,W=lbfgs(fun,gfun,np.array([0,0]))  # 此处x0是行向量，计算时要转成列向量
print(x0,fun0,k)

plt.plot(W[0,:],W[1,:],'g*',W[0,:],W[1,:])  # 画出迭代点轨迹
plt.show()

```

选择目标函数后的可视化结果：

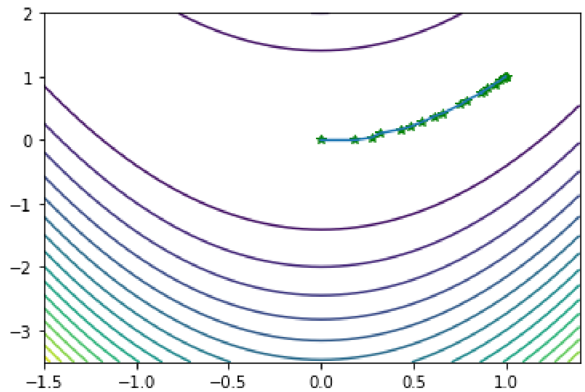

图 3: L-BFGS 算法可视化结果

3.1.4 利用 L-BFGS 方法求解逻辑回归问题

这道题不知道该咋解决了，因为之前的 L-BFGS 方法就实现得很奇怪。