最优化方法复习笔记（四）拟牛顿法与SR1,DFP,BFGS三种拟牛顿算法的推导与代码实现

锦恢
想做小说家的AI全栈工程师、弹吉他和做网站只是业余爱好

关注他

上一章传送门：

锦恢：最优化方法复习笔记（三）牛顿法及其收敛性分析
341 赞同 · 37 评论　文章

下个星期课变得好少，我最爱的咸鱼生活开始喽~~~

接上上次的牛顿法，这次开始的两篇文章写写拟牛顿法。

目录：

- 拟牛顿法
- 拟牛顿法框架
- 拟牛顿法是在 $||\cdot||_{B_k}$ 下的最速下降法
- 几种经典的拟牛顿算法
  - SR1
  - DFP
  - BFGS
- SR1,DFP,BFGS之间的关系
- Broyden族
- 代码实现三种拟牛顿算法

拟牛顿法

回顾一下牛顿法的表达式：

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k)$$

上一节说过了牛顿法的缺陷主要在于 $\nabla^2 f(x_k)$ 的求取和存储很消耗资源，对于较高维度的数据使用牛顿法求解比较困难。既然 $\nabla^2 f(x_k)$ 的求取和存储很困难，那么我们是否可以近似求得 $\nabla^2 f(x_k)$ 呢?

拟牛顿法(Quasi-Newton methods)的思路就是通过在牛顿法的迭代中加入近似求取每一步Hessian矩阵的迭代步，仅通过迭代点处的梯度信息来求取Hessian矩阵的近似值。现在我们把Hessian矩阵在第 k 步的迭代的近似值记作 B_k 。很显然，我们希望 $\nabla^2 f(x_k) \approx B_k$ ，在原本的牛顿法中，我们有：

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k)$$

那么通过 B_k 使用代替 $\nabla^2 f(x_k)$ ，我们的迭代式就变成了：

$$x_{k+1} = x_k - B_k^{-1} \nabla f(x_k)$$

当然，既然是近似，我们当然不能保证 B_k 能够比较好地近似 $k+1$ 步迭代的Hessian矩阵，所

以我们需要通过某种映射来对 B_k 多迭代得到 B_{k+1} 来使得 B_{k+1} 能够比较好地近似第 $k+1$ 步的Hessian矩阵。除此之外，你会发现，牛顿法需要的Hessian矩阵实际上只是需要使用它的逆矩阵，既然计算Hessian矩阵很烦，计算逆矩阵也很烦，那我们何不直接近似Hessian矩阵的逆矩阵呢？我们记 $H_k = B_k^{-1}$ ，再结合迭代近似值的想法，我们有如下的式子：

$$\begin{aligned} x_{k+1} &= x_k - H_k \nabla f(x_k) \\ H_{k+1} &= f(H_k) \end{aligned}$$

fine，其实我们已经得到拟牛顿法了。不同于之前的优化方法，拟牛顿法是一个思想而不是具体的方法，**只要迭代形式满足上述迭代式，并且能够收敛**，那么这样的方法都是拟牛顿法。那么接下来介绍的每一个具体的拟牛顿法都是在确定 H_k 怎么确定，怎么更新。

拟牛顿法框架

有了之前的认识，我们可以大致写出拟牛顿法的算法过程。

令$H_0 = I$,给出迭代初值x_0和迭代停止阈值$\epsilon > 0$,重复如下过程：
计算梯度：$\nabla f(x_k)$
如果$||\nabla f(x_k)|| < \epsilon$，$break$
计算搜索方向：$d_k = -H_k \nabla f(x_k)$
更新迭代点$x_{k+1} = x_k + d_k$
根据x_k处的信息更新H_k

当然，你也可以在更新迭代点时增加步长因子，使其变成阻尼拟牛顿法：

令$H_0 = I$,给出迭代初值x_0和迭代停止阈值$\epsilon > 0$,重复如下过程：
计算梯度：$\nabla f(x_k)$
如果$||\nabla f(x_k)|| < \epsilon, break$
计算搜索方向：$d_k = -H_k \nabla f(x_k)$
计算最优搜索步长$\alpha_k = \arg\min_{\alpha} f(x_k + \alpha d_k)$
更新迭代点$x_{k+1} = x_k + \alpha_k d_k$
根据x_k处的信息更新H_k

> 如何根据 x_k 处的信息更新 H_k 会在后面讲到，它确定了具体的拟牛顿法。

拟牛顿法是 $||\cdot||_{B_k}$ 在下的最速下降法

在具体展开有哪些可以使用的拟牛顿法之前，我觉得我们可以对牛顿法和拟牛顿法来做个比较，来获取一些拟牛顿法本身的性质亦或是难能可贵的认识，这会是在实际工程应用中难以获取的。

我们完全可以从另一个角度来看这个问题，类似于牛顿法的推导过程，我们现在处在迭代点 x_k 处，我们希望在下一步迭代 $x_{k+1} = x_k + d$ 后， $f(x_{k+1})$ 能尽可能地比 $f(x_k)$ 小。为了更好地观察 $f(x_{k+1})$ 和 $f(x_k)$ 之间到底差多少，我们可以使用Taylor展开式：

$$f(x_{k+1}) = f(x_k + d) = f(x_k) + g_k^T d + O(||d||^2)$$

舍去的高阶项，我们有：

$$f(x_{k+1}) = f(x_k + d) = f(x_k) + g_k^T d + O(||d||^2)$$

> 为了使得 $f(x_{k+1}) < f(x_k)$ ，此处默认 $g_k^T d < 0$

之间的关系

牛顿算法(Pyth...

也就是说，在误差允许的情况下(不要太大，Taylor展开式的领域近似，懂的都懂，不懂的我也没有办法=_=)， $f(x_{k+1})$ 比 $f(x_k)$ 小 $|g_k^T d|$ 。因为我们希望 $f(x_{k+1})$ 尽可能比 $f(x_k)$ 小，自然而然我们希望他们两之间的差值 $|g_k^T d|$ 尽可能大。注意到此处的 x_k 是固定的，也就是我们此时考察的函数在迭代点 x_k 处的梯度 g_k 是固定的，所以我们希望动一动 d ，来让 $|g_k^T d|$ 尽可能变大。因此，我们的问题变成了如下的优化问题

$$\max_d |g_k^T d|$$

但是，你会发现，当 g_k 与 d 的夹角固定时，只要我们把 $||d||$ 拉得足够大，那么 $|g_k^T d|$ 就会变得无限大，这样的结果是没有意义的。因此我们需要对 $||d||$ 做一个约束，为了和之前的拟牛顿法的 B_k 联系起来，我们使用椭球范数 $||\cdot||_{B_k}$ 来约束 d 的取值，我们将约束为一个定值（ $||d||_{B_k}$ 定义为 $||d^T B_k d||$ ），不妨就把这个定值设为1吧。那么我们要优化的问题就是如下的式子：

$$\max_d |g_k^T d| \qquad ||d||_{B_k} = 1$$

由**Cauchy-Schwartz**不等式，可得：

$$|g_k^T d| = \sqrt{(g_k^T d)^2} \le \sqrt{(g_k^T B_k^{-1} g_k)(d^T B_k d)} = \sqrt{g_k^T B_k^{-1} g_k}$$

当且仅当 $d = -B_k^{-1} g_k$ 时，上述不等式等号成立。

故，在我们将 d 在 $||\cdot||_{B_k}$ 意义下的椭球范数设为定值时，可以得到下降幅度最大的方向为 $d = -B_k^{-1} g_k$ 。因此我们可以说拟牛顿法是在 $||\cdot||_{B_k}$ 下的最速下降法。

> 由于每次迭代时的 B_k 都会变化，那么度量 d 的范数也会随之变化，因此，我们会将这样的方法称为**变尺度法**

那么当 $B_k = I$ 时，拟牛顿方向变为梯度下降方向，所以我们也可以说梯度下降法是在 $||\cdot||$ （ L_2 范数）下的最速下降法；而普通的牛顿法则是在 $||\cdot||_{\nabla^2 f(x_k)}$ 下的最速下降法。

几种典型的拟牛顿算法

还记得在**拟牛顿法框架**中讲到的模糊不清的最后一步——“根据 x_k 处的信息更新 H_k ”吗？下面所述便是这个步骤的具体展开。

下面介绍三种具体的拟牛顿法，而推导它们的关键则是确定如何迭代得到每步迭代点Hessian矩阵逆矩阵的倒数 $\nabla^2 f(x_k)^{-1}$ 的近似值 H_k 。也就是**如何确定 H_k 的迭代更新式。**

首先做几个符号的规定，来为我们后续的推导做符号运算上的简化。记 $s_k = x_{k+1} - x_k, y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ 。

很明显，我们就有：

$$B_{k+1} s_k = y_k$$
$$H_{k+1} y_k = s_k$$

SR1

SR1的全称为Symmetric Rank-One，是William C. Davidon在1956年提出的，但由于缺少收敛性分析，所以Davidon的论文被拒稿，导致SR1算法没能成为第一个公认的拟牛顿法。

SR1确定迭代式的逻辑比较简单——根据 x_k 处的信息得到一个修正量 ΔH_k 来直接加上 H_k 来更新 H_k ，也就是如下的式子：

$$H_{k+1} = H_k + \Delta H_k$$

由于我们希望 $H_k \approx \nabla^2 f(x_k)^{-1}, H_{k+1} \approx \nabla^2 f(x_{k+1})^{-1}$ ，所以我们有：

$$\Delta H_k \approx \nabla^2 f(x_{k+1})^{-1} - \nabla^2 f(x_k)^{-1}$$

由于 $\nabla^2 f(x_{k+1})^{-1}$ 和 $\nabla^2 f(x_k)^{-1}$ 都是对称矩阵，所以 ΔH_k 也应该是一个对称矩阵。

而SR1算法就直接把这个对称矩阵 ΔH_k 设为 $\beta u u^T$ ，那么迭代式为：

$$H_{k+1} = H_k + \beta u u^T \quad ①$$

其中 $\beta \in \mathbb{R}, u \in \mathbb{R}^n$ 。

很显然 $\beta u u^T$ 是对称矩阵。至少这个设置上是合理的，那么接下来就根据整个拟牛顿法中的其他条件来把 β 和 u 给表示或是整合成其他的表达。

我们在①的两边乘上 y_k ，再根据 $H_{k+1} y_k = s_k$ ，我们有：

$$\begin{aligned} H_{k+1} y_k =& H_k y_k + \beta u u^T y_k = s_k \\ \Rightarrow s_k - H_k y_k =& \beta (u^T y_k) u \quad ② \end{aligned}$$

其中 $\beta(u^T y_k)$ 是实数，所以我们有：

$$u = \frac{1}{\beta(u^T y_k)}(s_k - H_k y_k)$$

为了后续的运算方便，我们把 $\frac{1}{\beta(u^T y_k)}$ 记作 γ ，也就是：

$$u = \gamma(s_k - H_k y_k) \quad ③$$

②式还可以写成 $s_k - H_k y_k = \beta u u^T y_k$ ，我们将③式带入其中可得：

$$\begin{aligned} s_k - H_k y_k &= \beta \gamma^2 (s_k - H_k y_k)(s_k - H_k y_k)^T y_k \quad ④ \\ &= \beta \gamma^2 (s_k - H_k y_k) \left[(s_k - H_k y_k)^T y_k \right] \end{aligned}$$

注意到 $\beta \gamma^2$ 和 $(s_k - H_k y_k)^T y_k$ 都是实数，所以上式也能写成：

$$s_k - H_k y_k = \left[\beta \gamma^2 (s_k - H_k y_k)^T y_k \right] (s_k - H_k y_k)$$

很显然 $\beta \gamma^2 (s_k - H_k y_k)^T y_k = 1$ 才能使上式成立，因此：

$$\beta \gamma^2 = \frac{1}{(s_k - H_k y_k)^T y_k} \quad ⑤$$

将③和⑤代入①中可得：

$$\begin{aligned} H_{k+1} &= H_k + \beta \gamma^2 (s_k - H_k y_k)(s_k - H_k y_k)^T \\ &= H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k} \end{aligned}$$

因此我们得到了SR1更新的迭代式：

$$H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k}$$

结合之前的拟牛顿法框架，我们可以整合出SR1算法的具体步骤：

令$H_0 = I$,给出迭代初值x_0和迭代停止阈值$\epsilon > 0$,重复如下过程：

计算梯度：$\nabla f(x_k)$

如果$||\nabla f(x_k)|| < \epsilon$

计算搜索方向：$d_k = -H_k \nabla f(x_k)$

更新迭代点$x_{k+1} = x_k + d_k$

更新矩阵：$H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k}$

SR1虽然能近似 $\nabla^2 f(x_k)^{-1}$ ，但是我们无法证明SR1更新得到的 H_k 一定是正定的。也就是说，SR1拟牛顿法确定的拟牛顿方向不一定是下降方向=_=。

DFP

DFP是其发现者Davidon, Fletcher, Powell的开头大写字母拼成的，也是第一个公认的拟牛顿法。

其基本思路和SR1差不多，也是想着确定一个对称矩阵修正量 ΔH_k 来更新 H_k ，只不过SR1是令 $\Delta H_k = \beta u u^T$ ，而DFP是给与了 ΔH_k 更大的自由度。DFP中，我们令 $\Delta H_k = \beta u u^T + \gamma v v^T$ ，那么 H_k 的迭代更新式为：

$$H_{k+1} = H_k + \beta u u^T + \gamma v v^T \quad ①$$

其中的 β 和 γ 都是待定的系数。

那么接下来和推导SR1一样咯，我们在①式的两边右乘上 y_k 可得：

$$s_k = H_{k+1} y_k = H_k y_k + \beta u u^T y_k + \gamma v v^T y_k \quad ②$$

由于我们给与了 ΔH_k 更多的自由度，因此，仅凭上述的信息，我们当然是无法确定 β, γ, u, v 的值的。也就说满足上式的的 β, γ, u, v 取值其实是不唯一的。在DFP中，我们取 $u = s_k, v = H_k y_k$ 。别问为什么，问就是推导方便、结果简单、结果收敛性好说明。

接下来便是确定 β 和 γ 的过程了。很显然，这两个量的取值也不唯一。对于②式，我们可以改写为：

$$s_k - H_k y_k = \beta u u^T y_k + \gamma v v^T y_k$$

为了好算且方便，我们干脆直接令 $\beta u u^T y_k = s_k, \gamma v v^T y_k = -H_k y_k$ 。

对于 $\beta u u^T y_k = s_k$ ，由于我们之前取了 $u = s_k$ ，所以

$$u = s_k = \beta u u^T y_k = (\beta u^T y_k) u$$

为了使得上式恒成立，我们有 $\beta u^T y_k = 1$ ，因此，我们有 $\beta = \frac{1}{u^T y_k} = \frac{1}{s_k^T y_k}$ 。

同理我们也可以得到 $\gamma v^T y_k = -1$ ，因此，我们有 $\gamma = -\frac{1}{v^T y_k} = -\frac{1}{y_k^T H_k y_k}$ 。

将 β, γ, u, v 的取值代入①式中，我们就可以得到DFP更新 H_k 的迭代式了：

$$H_{k+1} = H_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k}$$

DFP的推导过程看似随意，但是DFP的迭代公式也可以通过如下的优化问题的解得到：

$$\min_{H} ||W^{-T}(H-H_k)W^{-1}||_F$$
$$st.\ H=H^T, Hy_k=s_k$$

其中 $W \in \mathbb{R}^{n\times n}$ 且 W^TW 满足拟牛顿方程。此处不做推导，无聊的同学请自行推导。

结合之前的拟牛顿法框架，我们可以整合出DFP算法的具体步骤（水行数警告）：

令$H_0=I$,给出迭代初值x_0和迭代停止阈值$\epsilon>0$,重复如下过程：

计算梯度：$\nabla f(x_k)$

如果$||\nabla f(x_k)|| < \epsilon$

计算搜索方向：$d_k=-H_k\nabla f(x_k)$

更新迭代点$x_{k+1}=x_k+d_k$

更新矩阵：$H_{k+1}=H_k+\frac{s_ks_k^T}{s_k^Ty_k}-\frac{H_ky_ky_k^TH_k}{y_k^TH_ky_k}$

BFGS

BFGS这个算法的记忆没有诀窍=_=，因为它是下面这四个人分别提出的，所以和DFP一样，用四个人的名字的开头字母命名。

其推导其实和DFP一样，不过BFGS是从 B_k 出发来推导的。（还记得 B_k 是啥吗？ B_k 是我们对 $\nabla^2 f(x_k)$ 的近似）

对于 B_k ，我们采用和DFP一样的思路，也就是考虑 B_k 的rank-tow修正：

$$B_{k+1}=B_k+\beta uu^T+\gamma vv^T$$

后面和DFP一模一样，我们可以得到 B_k 的迭代公式：

$$B_{k+1}=B_k+\frac{y_ky_k^T}{y_k^Ts_k}-\frac{B_ks_ks_k^TB_k}{s_k^TB_ks_k} \quad ①$$

你会发现，我们其实就是把DFP中的 s_k 与 y_k 互换位置，把 H_k 换成 B_k 罢了。

那么我们要求的 H_{k+1} 不就是 B_{k+1}^{-1} 吗？那么我们直接对①式两边取逆就行了。你会发现右式的取逆你算不出来，这时你就需要SMW公式了：

Shermann-Morrison-Woodbury Let A is $n \times n$ invertible matrix, $u, v \in \mathbb{R}^n$. Then $A + uv^T$ is invertible iff $1 + v^T A^{-1} u \neq 0$, and

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1} u v^T A^{-1}}{1 + v^T A^{-1} u}$$

我们可以递归地使用上面的式子，令 $A = B_k + \frac{y_k y_k^T}{y_k^T s_k}, u = -\frac{B_k s_k}{s_k^T B_k s_k}, v = B_k s_k$ 代入SMW公式中。其中 A^{-1} 的计算再用一次SMW公式，反正通过一通暴算，最终我们得到：

$$\begin{aligned} H_{k+1} = B_{k+1}^{-1} &= \left(B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \right)^{-1} \\ &= H_k + \left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k} \right) \frac{s_k s_k^T}{s_k^T y_k} - \left(\frac{s_k y_k^T H_k + H_k y_k s_k^T}{s_k^T y_k} \right) \end{aligned}$$

如此，我们便得到了BFGS H_k 的迭代更新式：

$$H_{k+1} = H_k + \left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k} \right) \frac{s_k s_k^T}{s_k^T y_k} - \left(\frac{s_k y_k^T H_k + H_k y_k s_k^T}{s_k^T y_k} \right)$$

于是，我们得到了BFGS拟牛顿法的具体步骤(我真不是在水行数~~~)：

令$H_0 = I$,给出迭代初值x_0和迭代停止阈值$\epsilon > 0$, 重复如下过程：

计算梯度：$\nabla f(x_k)$

如果$||\nabla f(x_k)|| < \epsilon$

计算搜索方向：$d_k = -H_k \nabla f(x_k)$

更新迭代点$x_{k+1} = x_k + d_k$

更新矩阵：$H_{k+1} = H_k + \left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k} \right) \frac{s_k s_k^T}{s_k^T y_k} - \left(\frac{s_k y_k^T H_k + H_k y_k s_k^T}{s_k^T y_k} \right)$

SR1, DFP, BFGS之间的关系

有了之前的SMW公式，我们可以尝试求取上述的三个拟牛顿法的 B_{k+1} 的更新公式，毕竟 B_k 才是对Hessian矩阵的近似，我们需要知道每步对Hessian矩阵的近似情况，这个对收敛性分析会有比较大的帮助。

SR1的迭代式为

$$H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k}$$

两边取逆，使用SMW公式，可以得到SR1中， B_k 的迭代式

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$$

可以发现，结果就是将 s_k 和 y_k 交换，把 B_k 和 H_k 交换。

因此，我们说SR1是**自对偶**的。

再看看DFP和BFGS迭代公式两边取逆后的结果：

$$\mathbf{DFP}: H_{k+1}=H_k+\frac{s_k s_k^T}{s_k^T y_k}-\frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k}$$

$$B_{k+1}=B_k+\left(1+\frac{s_k^T B_k s_k}{s_k^T y_k}\right)\frac{y_k y_k^T}{s_k^T y_k}-\left(\frac{y_k s_k^T B_k+B_k s_k y_k^T}{s_k^T y_k}\right)$$

$$\mathbf{BFGS}: H_{k+1}=H_k+\left(1+\frac{y_k^T H_k y_k}{s_k^T y_k}\right)\frac{s_k s_k^T}{s_k^T y_k}-\left(\frac{s_k y_k^T H_k+H_k y_k s_k^T}{s_k^T y_k}\right)$$

$$B_{k+1}=B_k+\frac{y_k y_k^T}{y_k^T s_k}-\frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

可以看到DFP和BFGS公式高度对称，只需 $s_k \leftrightarrow y_k, B_k \leftrightarrow H_k$ 就可以在DFP和BFGS公式之间相互转换。

因此，我们说DFP和BFGS**互为对偶。**

Broyden族

既然DFP和BFGS是互为对偶的，那用哪一个比较好呢？你当然可以通过若干组实验来测试哪个的性能的更优，或者对其收敛一通验证。但是一个比较的朴素的做法就是“我都要”，也就是取DFP迭代式和BFGS迭代式的正加权组合：

$$B_{k+1}^{\phi}=\phi_k B_{k+1}^{DFP}+(1-\phi_k)B_{k+1}^{BFGS}$$

其中 $\phi_k \in [0,1]$ 。我们记 $w_k=\frac{y_k}{y_k^T s_k}-\frac{B_k s_k}{s_k^T B_k s_k}$ ，然后将 B_{k+1}^{DFP} 和 B_{k+1}^{BFGS} 的表达式代入其中可得：

$$B_{k+1}^{\phi}=B_k-\frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}+\frac{y_k y_k^T}{s_k^T y_k}+\phi_k(s_k^T B_k s_k)w_k w_k^T$$

随着 ϕ_k 遍历 $[0,1]$ 内的值，我们可以得到一系列的 B_{k+1}^{ϕ} 。我们将这一系列的 B_{k+1}^{ϕ} 的集合称为Broyden族。

而且根据定义， $\phi_k=0$ 时， B_{k+1}^{ϕ} 即为BFGS校正； $\phi_k=1$ 时， B_{k+1}^{ϕ} 即为DFP校正。而且可以证明，DFP和BFGS校正都是保持正定的，且我们的 $\phi_k \geq 0$ ，所以我们只要满足 $y_k^T s_k>0$ 就可以保证Broyden族校正都是保持正定的。

代码实现三种拟牛顿算法(Python)

终于到了我最爱的编程环节了，这次我们就不像上次的梯度下降法一样造轮子了，我们直接调取写好的优化库来快速完成项目需求。

因为笔者是个Python小白，所以关于数值优化的Python第三方库我知道如下三个：`cvxpy`，`cvxopt`，`scipy`。其中 `scipy.optimize` 子库有关于优化的算子。

> 由于上述的三个库都是基于 `numpy` 库开发的，所以在调用 `pip` 指令安装时，请先安装 `numpy` 库

我翻了一上午的源代码和document，并没有在 `cvxpy` 和 `cvxopt` 中找到关于拟牛顿法的算子。因此，笔者下面采用 `scipy.optimize` 来实现三种拟牛顿法。如果你没有安装相关的依赖库，请打开命令行，输入以下命令来自动安装（请确保你的Python解释器的Script文件夹的路径已被添加到环境变量Path中）：

```
pip install numpy, matplotlib, scipy -i https://pypi.tuna.tsinghua.edu.cn/simple
```

此处我们分别使用SR1，DFP和BFGS拟牛顿法优化如下的函数：

$$\min f(x)=\sum_{i=1}^{4} r_i(x)^2$$

其中$r_{2i-1}(x)=10(x_{2i}-x_{2i-1}^2), r_{2i}(x)=1-x_{2i-1}$ ， $x\in\mathbb{R}^4$ ， x_k 代表向量 x 的第 k 个维度上的元素。

已知上述的优化问题的最优点为 $(1,1,1,1)^T$ ，取迭代初值为 $x_0=(1.2,1,1,1)^T$ 。

我们首先先实现上述的函数，我通过一个函数获取一个映射 f ：

> 下面所有的代码都写在一个文件中

```
# -*- utf-8 -*-
# date: 2020-10-22
# author: 锦恢
```

```
from scipy.optimize import fmin_powell, fmin_bfgs, fmin_cg, minimize, SR1
import numpy as np
import matplotlib.pyplot as plt

def r(i, x):
    if i % 2 == 1:
        return 10 * (x[i] - x[i - 1] ** 2)  # 因为ndarray数组的index是从0开始的， i多减一
    else:
        return 1 - x[i - 2]

def f(m):
    def result(x):
        return sum([r(i, x) ** 2 for i in range(1, m + 1)])
    return result
```

通过 $f(4)$ 我们就可以获取上述需要优化的函数的映射。

为了观察拟牛顿法运行过程中迭代点的下降情况，我们需要计算

```
# 获取retall的每个点的值损失|f(x) - f(x^*)|
def getLosses(retall, target_point, func):
    """
    :param retall: 存储迭代过程中每个迭代点的列表，列表的每个元素时一个ndarray对象
    :param target_point: 最优点，是ndarray对象
    :param func: 优化函数的映射f
    :return: 返回一个列表，代表retall中每个点到最优点的欧氏距离
    """
    losses = []
    for point in retall:
        losses.append(np.abs(func(target_point) - func(point)))
    return losses
```

`scipy.optimize` 子库中的许多执行拟牛顿法的算子提供了 `call_back` 参数，该参数要求传入一个函数对象，在拟牛顿每步迭代完后，传入的 `call_back` 函数会被调用。由于使用 `SR1` 法的算子 `minimize` 无法返回迭代过程中的每一个迭代点（也就是 `retall` ），于是我们需要 `call_back` 函数来将迭代完的点传入外部的列表，从而获取SR1的 `retall` 。除此之外，我们可以使用 `call_back` 函数来指定迭代停止的条件。

我们编写一个返回函数对象的函数，它会根据我们传入参数的不同返回不同的 `call_back` 函数：

```
sr1_losses = [] # 存储SR1的retall的列表
func = f(4) # 获取需要优化的函数
```

```
# 通过callback方法来添加迭代的停止条件
def getCallback(func, target_point, ftol, retall):
    """
    :param func: 优化目标的函数
    :param target_point:  目标收敛点
    :param ftol: 收敛条件：|f(x) - f(x^*)| < ftol时，迭代停止
    :param retall: 是否存储迭代信息
    :param extern_retall: 如果retall为True， 填入一个列表，迭代信息会存在这个列表中
    :return: call_back函数对象
    """
    def result(xk, state=None):
        if retall:
            global func, sr1_losses
        loss = np.abs(func(target_point) - func(xk))
        if loss < ftol:
            return True
        else:
            if retall:
                sr1_losses.append(loss)
            return False
    return result
```

为了方便可视化，我们将数据可视化的逻辑封装到一个函数中：

```
# 绘制下降曲线
def plotDownCurve(dpi, losses, labels, xlabel=None, ylabel=None, title=None, grid=True
    plt.figure(dpi=dpi)
    for loss, label in zip(losses, labels):
        plt.plot(loss, label=label)
    plt.xlabel(xlabel, fontsize=12)
    plt.ylabel(ylabel, fontsize=12)
    plt.title(title, fontsize=18)
    plt.yscale("log")
    plt.grid(grid)
    plt.legend()
```

接着我们定义一下迭代初值、最优点和终止条件的阈值 ϵ （ $\left|f\left(x_{k+1}\right)-f\left(x_k\right)\right|<\epsilon$ 时，迭代停止）并获取三个拟牛顿法需要的 call_back 函数。

```
x_0 = np.array([1.2,1.0,1.0,1.0])   # 迭代初值
target_point = np.array([1,1,1,1], dtype="float32")     # 最优点
FTOL = 1e-8     # 终止阈值

sr1_callback = getCallback(func, target_point, ftol=FTOL, retall=True)
dfp_callback = getCallback(func, target_point, ftol=FTOL, retall=False)
bfgs_callback = getCallback(func, target_point, ftol=FTOL, retall=False)
```

下面我们我们使用 minimum,fmin_powell,fmin_bfgs 来实现三种拟牛顿法的迭代，并把DFP和BFGS的 retall 存入列表中。

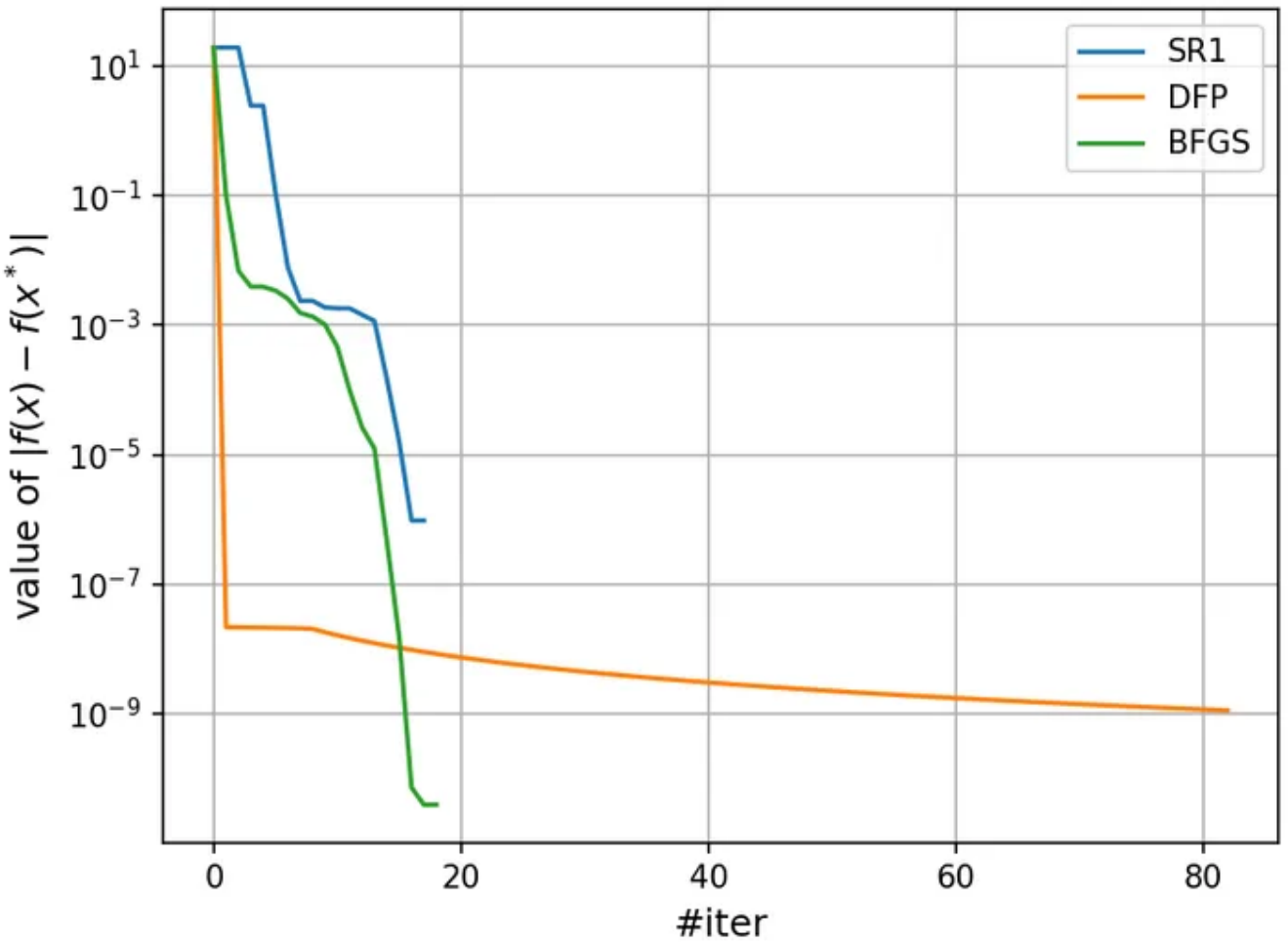
```
minimum = minimize(fun=f(4), x0=x_0,        # 通过minimize函数执行SR1，根据内嵌的callback
                   method="trust-constr",
                   hess=SR1(),
                   callback=sr1_callback)

dfp_minimum, dfp_retall = fmin_powell(func=func, x0=x_0,
                                      retall=True,
                                      disp=False,
                                      callback=dfp_callback)
dfp_losses = getLosses(dfp_retall, target_point, func=func)

bfgs_minimum, bfgs_retall = fmin_bfgs(f=func, x0=x_0,
                                      retall=True,
```

```
                                        disp=False,
                                        callback=bfgs_callback)
bfgs_losses = getLosses(bfgs_retall, target_point, func=func)
```

迭代做完后，我们自然想知道结果如何，可视化是一个直观的方法，我们将 `plt` 画布的分辨率调为150，设置一下各个轴的名称，将它可视化出来：

```
plotDownCurve(dpi=150,
              losses=[sr1_losses, dfp_losses, bfgs_losses],
              labels=["SR1", "DFP", "BFGS"],
              xlabel="#iter",
              ylabel="value of $|f(x) - f(x^*)|$",
              title="losses curve of SR1, DFP and BFGS")
plt.show()
```

out:



三种拟牛顿法的下降曲线

我们可以查看三种方法得到的最优点和它们具体的迭代次数：

```
print(f"SR1\t最终迭代点:{minimum.x.tolist()}, 共经历{minimum.cg_niter}次迭代")
print(f"DFP\t最终迭代点:{dfp_minimum}, 共经历{len(dfp_losses)}次迭代")
print(f"BFGS\t最终迭代点:{bfgs_minimum}, 共经历{len(bfgs_losses)}次迭代")
```

out:

```
SR1 最终迭代点:[0.9999962062462336, 0.9999929560009194, 0.9999943517470878, 0.999991518
DFP 最终迭代点:[0.99998036 0.99996341 1.         1.         ], 共经历83次迭代
BFGS    最终迭代点:[0.99999547 0.9999909  0.99999572 0.99999144], 共经历19次迭代
```

可以看到三种方法都成功收敛到了 $(1,1,1,1)^T$ ，说明程序是没问题滴~~~