数值最优化方法大作业

21231024 曹致瑾

一、实验代码说明

该实验的全部代码由 matlab2016b 编写，对代码文件中各函数与脚本的说明如下表所示：

表一 各函数与脚本的说明

脚本/函数名称	脚本/函数功能
get_problem.m	脚本，生成所需要求解的问题函数（匿名函数形式） 由于本次作业中 m、n 较大 因此生成匿名函数形式的原函数与梯度函数花费时间较长 由于相同情况下 5 种求解方法用的匿名函数是相同的 因此每种情况匿名函数只需生成一次保存在 get_f_g.m 中 后面需要用时复用即可
get_f_g.m	脚本，加载所需要求解问题相应 n 下的 f，g
get_x0.m	脚本，加载所需要求解的问题的初值
wolf.m	函数，利用 wolf 准则(系数缩减型)进行线搜索， 返回满足 wolf 准则的线搜索步长
wolf_wp.m	函数，利用 wolf 准则（区间逼近型）进行线搜索， 返回满足 wolf 准则的线搜索步长
L_BFGS_H.m	L_BFGS 主函数， m 为算法中存储信息步数 epsilon = 10^(-6)为迭代精度参数 flag 为算法终止标志，为 1 则终止迭代跳出循环 k 为迭代次数 bound = min（m，k）为当前时刻最后已知点号 x(:,bound)为当前点，x(:,bound + 1)为更新点
FR.m	非线性共轭梯度 FR 方法主函数 epsilon、flag、k 定义均与 L_BFGS_H.m 相同 x(:,1)为当前点，x(:,2)为更新点

PRP.m	非线性共轭梯度 PRP+方法主函数 所有变量定义均与 FR.m 相同

该实验中，get_problem.m 以匿名函数形式生成不同问题不同 n 下的 f_function、g_function，将其保存在相应的.mat 文件中，在进行不同算法验证时只需调用相应的.mat 文件加载 f_function、g_function 即可，各.mat 文件说明如下表所示。

表二 各.mat 文件说明

文件名称	内容说明
P21_n100.mat	存储问题 21，n=100 情况下，生成的匿名函数 f_function、g_function
P21_n500.mat	存储问题 21，n=500 情况下，生成的匿名函数 f_function、g_function
P21_n1000.mat	存储问题 21，n=1000 情况下，生成的匿名函数 f_function、g_function
P25_n100.mat	存储问题 25，n=100 情况下，生成的匿名函数 f_function、g_function
P25_n500.mat	存储问题 25，n=500 情况下，生成的匿名函数 f_function、g_function
P25_n1000.mat	存储问题 25，n=1000 情况下，生成的匿名函数 f_function、g_function
P28_n100.mat	存储问题 28，n=100 情况下，生成的匿名函数 f_function、g_function
P28_n500.mat	存储问题 28，n=500 情况下，生成的匿名函数 f_function、g_function
P28_n1000.mat	存储问题 28，n=1000 情况下，生成的匿名函数 f_function、g_function

该实验中，原实验说明要求 n=1000、5000、10000，而由于 cpu 限制（例：28 题在 n=1000 时，生成匿名函数大概需要花费 10 分钟，按照 n^2（m×n)量级估算，n=5000 时生成匿名函数需要 250 分钟，约 4 小时；n=10000 时生成匿名函数需要 1000 分钟，约 16 小时，这还是在单一问题下的时间花费。且 matlab 生成匿名函数是逐项生成求导的，合并功能比较差，这可能造成后续算法运行时间也比较长），出于效率考虑，我决定将这一数量级减少到 1/10 进行研究，即 n=100、500、1000。

以上情况也让我思考，或许对于规模较大的优化问题，还是用带 numpy 加速的 python 代码编写解决比较好，这一方法可以在之后留作研究。

二、解大规模优化问题算法的比较

2.1问题(21)——Extended Rosenbrock function

2.1.1问题描述

$$
\begin{aligned}
&(a)\ \ n \quad \text{variable but even,} \qquad m = n \\
&(b)\ \ f_{2i-1}(x) = 10(x_{2i} - x^2{}_{2i-1}) \\
&\qquad f_{2i}(x) = 1 - x_{2i-1} \\
&(c)\ \ x_0 = (\xi_j) \quad \text{where} \ \ \xi_{2j-1} = -1.2, \xi_{2j} = 1 \\
&(d)\ \ f = 0 \text{ at } (1,...,1)
\end{aligned}
$$

2.1.2实验环境设置

wolfe搜索准则参数：alpha = 1;%初始步长

index = 0.6; %缩减步长系数

rho = 0.0001; %wolfe算法参数

sigma = 0.9; %wolfe算法参数

终止条件：$\| g(k) \| < \varepsilon,\ \varepsilon = 10^{-6}$

2.1.3数值结果

(1) n = 100时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	1.9829e-16	3.3627e-16	2.2297e-14
迭代次数	39	50	47
函数调用次数	152	179	167
$\|g_k\|$	6.0067e-07	8.0262e-07	9.5859e-07
cpu时间	0.135 s	0.140 s	0.165 s
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	/	6.8389e-13	
迭代次数	/	191	
函数调用次数	/	2973	
$\|g_k\|$	/	9.7886e-07	
cpu时间	/	0.169 s	
程序运行状况说明	失败，线搜索无法找到满足wolfe准则的步长，最后点$\|g_k\|$为208.1187，不理想	收敛到全局极小值点	

(2) n =500时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	3.6809e-16	2.6891e-16	1.7838e-14
迭代次数	40	51	48
函数调用次数	155	182	170
$\|g_k\|$	3.3500e-07	7.1772e-07	8.5739e-07
cpu时间	0.590 s	0.582 s	0.597 s
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	/	/	
迭代次数	/	/	
函数调用次数	/	/	
$\|g_k\|$	/	/	
cpu时间	/	/	
程序运行状况说明	失败，线搜索无法找到满足wolfe准则的步长，最后点$\|g_k\|$为465.3675，不理想	失败，线搜索无法找到满足wolfe准则的步长，最后点$\|g_k\|$为0.0024，已经较为接近全局极小值点	

(3) n=1000时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	7.1854e-16	8.6008e-17	5.7080e-15
迭代次数	40	52	49
函数调用次数	155	185	173
$\|g_k\|$	4.7986e-07	4.0589e-07	4.8502e-07
cpu时间	1.357 s	1.359 s	1.373 s
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	/	6.3381e-14	
迭代次数	/	200	
函数调用次数	/	3074	
$\|g_k\|$	/	9.1578e-07	
cpu时间	/	1.657 s	
程序运行状况说明	失败，线搜索无法找到满足wolfe准则的步长，最后点$\|g_k\|$为658.1290，不理想	收敛到全局极小值点	

2.1.4结果分析

（1）拟牛顿方法L_BFGS与非线性共轭梯度方法FR、PRP^+对比：在该题目中，拟牛顿方法L_BFGS在求解成功率、迭代次数、调用函数次数三方面均优于非线性共轭梯度方法FR、PRP^+。

对这一结果的原因分析如下。拟牛顿方法L_BFGS近似得到了函数的二阶信息，近似程度较高，算法能以超线性速度收敛；而非线性共轭梯度方法FR、PRP^+只利用了函数的一阶信息，近似程度较低，算法收敛速度相对较慢。

（2）L_BFGS取不同m对比：在该题目中，不同m下，算法在迭代次数、函数调用次数、cpu时间、程序运行状况上并无明显差别。

对这一结果的原因分析如下。1.增加m的值可以提高算法的精度和收敛速度，但同时也会增加程序运行的内存和计算量。2.从理论上讲，增加m的值可以提高算法的精度和收敛速度，因为更多的历史信息被用来近似Hessian矩阵。然而，实际上对于大多数任务，m的值在3到20之间能满足需求，可见对于具体问题只要取满足要求的m即可，完全不必要让m尽可能大。

（3）FR与PRP^+性能对比：在该题目中，从求解成功率、失败结果与正确结果的近似程度来看，PRP^+算法的效果要优于FR算法的效果。

对这一结果的原因分析如下。FR方法可能出现连续小步长的情况，这使得它难以成功收敛至极小点。

2.2问题(25)——Variable dimensioned function

2.2.1问题描述

$$(a)\ \ n \quad \text{variable but even,} \qquad m = n + 2$$

$$(b)\ \ f_i(x) = x_i - 1, \qquad i = 1,...,n$$

$$f_{n+1}(x) = \sum_{j=1}^{n} j(x_j - 1)$$

$$f_{n+2}(x) = \left(\sum_{j=1}^{n} j(x_j - 1) \right)^2$$

$$(c)\ \ x_0 = (\xi_j) \quad \text{where} \quad \xi_j = 1 - (j/n)$$

$$(d)\ \ f = 0 \quad \text{at} \quad (1,...,1)$$

2.2.2实验环境设置

wolfe搜索准则参数（n=100时使用）：alpha = 1;%初始步长

index = 0.6; %缩减步长系数

rho =1e-20 ; %wolfe算法参数

sigma = 0.9; %wolfe算法参数

wolfe_wp搜索准则参数（n=500、1000时使用）：

alpha = 1;%初始步长

rho =1e-25 ; %wolfe算法参数

sigma = 1 - 1e-5; %wolfe算法参数

终止条件：$\| g(k) \| < \varepsilon,\ \varepsilon = 10^{-6}$

2.2.3数值结果

(1) n = 100时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	2.9078e-15	1.1562e-15	1.4233e-18
迭代次数	271	84	66
函数调用次数	5942	621	261
$\|g_k\|$	9.8674e-07	7.5050e-08	2.4654e-07
cpu时间	12.337 s	2.690 s	2.005 s
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	1.0425e-19	6.4349e-19	
迭代次数	50	504	
函数调用次数	1564	23006	
$\|g_k\|$	3.7561e-07	9.3322e-07	
cpu时间	4.345 s	44.821 s	
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	

(2) n =500时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	6.1386e-20	/	2.8250e-18
迭代次数	375	/	83
函数调用次数	7894	/	713
$\|g_k\|$	7.5258e-07	/	3.7631e-07
cpu时间	0.651 s	/	0.256 s
程序运行状况说明	收敛到全局极小值点	失败，线搜索无法找到满足wolfe_wp准则的步长，最后点$\|g_k\|$为0.0249，已经较为接近全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	4.8634e-14	1.5607e-13	
迭代次数	368	2089	
函数调用次数	5403	55187	
$\|g_k\|$	9.9755e-07	9.9769e-07	
cpu时间	0.926 s	2.870 s	
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	

(3) n=1000时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	5.0560e-19	/	6.3811e-20
迭代次数	1131	/	388
函数调用次数	25148	/	6738
$\|g_k\|$	1.4221e-09	/	5.0522e-10
cpu时间	2.964 s	/	1.654 s
程序运行状况说明	收敛到全局极小值点	失败，线搜索无法找到满足wolfe_wp准则的步长，最后点$\|g_k\|$为2.1271e-06，已经非常接近全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	3.6755e-13	2.0445e-14	
迭代次数	233	2181	
函数调用次数	3404	57475	
$\|g_k\|$	5.4215e-07	9.7622e-07	
cpu时间	1.798 s	7.096 s	
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	

2.2.4结果分析

（1）拟牛顿方法L_BFGS与非线性共轭梯度方法FR、PRP^+对比：在该题目中，只要m大到一定值，整体上拟牛顿方法L_BFGS在求解成功率、迭代次数、调用函数次数三方面仍优于非线性共轭梯度方法FR、PRP^+。

对这一结果的原因分析如下。拟牛顿方法L_BFGS近似得到了函数的二阶信息，近似程度较高，算法能以超线性速度收敛；而非线性共轭梯度方法FR、PRP^+只利用了函数的一阶信息，近似程度较低，算法收敛速度相对较慢。

（2）L_BFGS取不同m对比：在该题目中，m越小，从迭代次数、调用函数次数与cpu时间来看，L_BFGS算法的性能越好。

对这一结果的原因分析如下。这个函数除了$f_{n+1}(x)$、$f_{n+2}(x)$这两项，剩下的全都是正定二次函数的形式，这就意味着二阶信息的多少显著决定了最终计算出的拟牛顿方向的精确程度。而这一显著性是如此巨大，使得即使m增大会使每次迭代中的计算量稍稍增大、计算时间稍稍增长，但精确性带来了迭代次数和调用次数的显著降低，因此最终的cpu时间仍是m大的占优。

（3）FR与PRP^+性能对比：在该题目中，从迭代次数、调用函数次数与cpu时间来看，FR算法的效果要优于PRP^+算法的效果。

一般情况下PRP+数值表现应该是要好于FR的。这里我倾向于认为是初始点的选择使得FR得到的方向更快且更趋于实际的下降方向，这与函数与初始点的特殊性有关。

2.3问题(28)——Discrete boundary value function

2.3.1问题描述

$$(a)\ n \quad \text{variable}, \qquad m = n$$

$$(b)\ f_i(x) = 2x_i - x_{i-1} - x_{i+1} + h^2(x_i + t_i + 1)^3 / 2$$

$$where \quad h = 1/(n+1), \quad t_i = ih, \quad and \quad x_0 = x_{n+1} = 0$$

$$(c)\ x_0 = (\xi_j) \qquad where \quad \xi_j = t_j(t_j - 1)$$

$$(d)\ f = 0$$

2.3.2实验环境设置

wolfe搜索准则参数： alpha = 1;%初始步长

index = 0.6; %缩减步长系数

rho = 0.0001; %wolfe算法参数

sigma = 0.9; %wolfe算法参数

终止条件：$\| g(k) \| < \varepsilon,\ \varepsilon = 10^{-6}$

2.3.3数值结果

(1) n = 100时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	5.3545e-09	2.0752e-10	4.1126e-10
迭代次数	16600	6889	6317
函数调用次数	90000	35628	32854
$\|g_k\|$	8.9986e-07	8.6962e-07	9.1995e-07
cpu时间	9.987 s	4.735 s	4.941 s
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	4.7520e-11	/	
迭代次数	15983	/	
函数调用次数	167610	/	
$\|g_k\|$	9.2610e-07	/	
cpu时间	17.085 s	/	
程序运行状况说明	收敛到全局极小值点	失败，线搜索无法找到满足wolfe准则的步长，最后点$\|g_k\|$为1.4920e-05，已经较为接近全局极小值点	

(2) n =500时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	9.4639e-09	7.7541e-09	6.7440e-09
迭代次数	585	5914	10857
函数调用次数	3156	30602	56588
$\|g_k\|$	9.4842e-07	9.0310e-07	9.8598e-07
cpu时间	6.840 s	32.104 s	47.549 s
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	3.4980e-09	9.8132e-09	
迭代次数	55712	314	
函数调用次数	598194	3109	
$\|g_k\|$	9.9969e-07	7.4112e-07	
cpu时间	368.989 s	6.654 s	
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	

(3) n=1000时数值结果

求解算法	L_BFGS(m=5)	L_BFGS(m=15)	L_BFGS(m=30)
f(x*)	1.2588e-09	1.2263e-09	1.2349e-09
迭代次数	218	857	677
函数调用次数	1175	4461	3524
$\|g_k\|$	9.5194e-07	9.6178e-07	9.8355e-07
cpu时间	6.799 s	20.814 s	17.059 s
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	收敛到全局极小值点

求解算法	FR	PRP^+	
f(x*)	1.0479e-09	1.2822e-09	
迭代次数	23133	45	
函数调用次数	246172	387	
$\|g_k\|$	9.9855e-07	8.8509e-07	
cpu时间	494.239 s	4.610 s	
程序运行状况说明	收敛到全局极小值点	收敛到全局极小值点	

2.3.4结果分析

（1）拟牛顿方法L_BFGS与非线性共轭梯度方法FR、PRP^+对比：在该题目中，整体上看拟牛顿方法L_BFGS在求解成功率、迭代次数、调用函数次数三方面均优于非线性共轭梯度方法FR，但差于PRP^+。

对这一结果的原因分析如下。L_BFGS优于FR很好解释，拟牛顿方法L_BFGS近似得到了函数的二阶信息，近似程度较高，算法能以超线性速度收敛；而非线性共轭梯度方法FR只利用了函数的一阶信息，近似程度较低，算法收敛速度相对较慢。对于为什么PRP^+这里能优于L_BFGS，一个是PRP^+方法在计算上确实比L_BFGS简单多了。但从迭代次数上看，PRP^+收敛得也更快，这个只能说是出于初始点和函数的特殊性，也可能于非精确线搜索带来的偶然性有关。

（2）L_BFGS取不同m对比：在该题目中，不同m下，算法在迭代次数、函数调用次数、cpu时间、程序运行状况上较为混乱、没什么规律。

对这一结果的原因分析如下。定性来看，出于二阶信息和计算量的限制，m确实应该在一个值处表现出较优的结果，但这个值影响因素较多、难以分析。同时，非精确线搜索算法还为求解带来了一定的不确定度，有时就是恰巧在不同方向上，相同非精确线搜索算法得出的步长都能满足要求，但就是在其中一个方向上经过相应步长恰巧落到了极小值点附近，因此比较好收敛。因此，总体上看，总结算法的数值性能，需要参考多个函数、多个初始点下的结果，以消除这种不确定性。

（3）FR与PRP^+性能对比：在该题目中，从求解成功率、失败结果与正确结果的近似程度来看，PRP^+算法的效果要优于FR算法的效果。

对这一结果的原因分析如下。FR方法可能出现连续小步长的情况，这使得它难以成功收敛至极小点，如果可以成功收敛至极小点，往往也需要花费更多的迭代步数与更长的时间。