计91 刘程华 2018011687

一、考虑等式约束优化问题

$$
\begin{array}{ll}
\min & f(x)=\sum_{i=1}^{n} x_i \log x_i \\
\text{s.t.} & Ax=b
\end{array}
$$

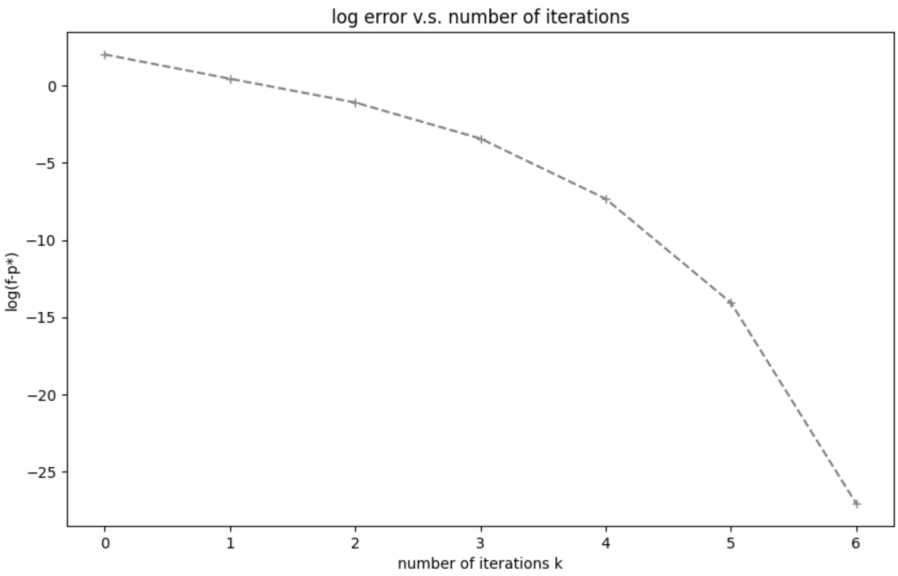
其中 $\operatorname{dom} f=R_{++}^{n}, \quad A \in R^{m \times n}, \quad m<n$。(1) 采用标准 Newton 法求解上述问在 $m=30, n=100$ ，可行初始点为 x_0 时的最优解 x^* 和 p^* 。 采用回溯直线搜索，合理选择回溯参数，要求误差 $\eta=10^{-10}$, 并画出 $\log\left(f\left(x^{(k)}\right)-p^*\right)$ 和 k 迭代次数的关系图。 (2) 采用不可行初始点 Newton 法求解上述问题在 $m=30, n=100$ ，不可行初始点为 x_1 时的最 优解 x^{**} 和 p^{**} 。采用回溯直线搜索，合理选择回溯参数，要求误差 $\eta=10^{-10}$, 并画出$\log\left(f\left(x^{(k)}-p^{**}\right)\right)$ 和迭代次数 k 的关系图。

---

(1)我们设置回溯参数$\alpha=0.2 \quad \beta=0.8$得到最优值为$p^*=-31.01283021981505$最优解如下：

```
[0.17532642 0.54611282 0.76368044 0.96131916 0.37445166 0.3527156
 0.36777798 1.02142455 0.37812831 0.25007758 0.18606974 0.46315939
 0.37151271 0.48309865 0.30150414 0.08321218 0.36235094 0.88603499
 0.53080102 0.63463179 0.42959254 0.64033416 0.10741741 0.33371242
 0.44113777 0.71131122 0.23786689 0.34488757 0.38944621 0.40185213
 0.56908289 0.4489786  0.33322532 0.22082084 0.43813891 0.39601092
 0.3475417  0.44491479 0.62185117 0.09579495 0.3485656  0.43462242
 0.63587205 0.28342355 0.84877756 0.29001809 0.30134103 0.63604236
 0.5311949  0.49482199 0.50632888 0.4759541  0.48189151 0.25931878
 0.38322847 0.1634453  0.33063722 0.48008413 0.88358649 0.3790492
 0.40072391 0.62255794 0.26867464 0.31341895 0.6235641  0.53191447
 0.12308642 0.53365359 0.4167203  0.37954069 0.39805615 0.32331447
 0.35841583 0.35528431 0.92949532 0.27903386 0.97611158 0.4973101
 0.47220904 0.29674416 0.27887256 0.36241406 0.22652063 0.29707326
 0.1071736  0.58546588 0.94957098 0.79686147 0.51160996 0.83246085
 0.87389594 0.65444774 0.53527937 0.27798009 0.28755737 0.19735683
 0.39527824 0.55940325 0.55001295 0.4225168 ]
```

$\log\left(f\left(x^{(k)}\right)-p^{**}\right)$ 和迭代次数 k 的关系图如下：



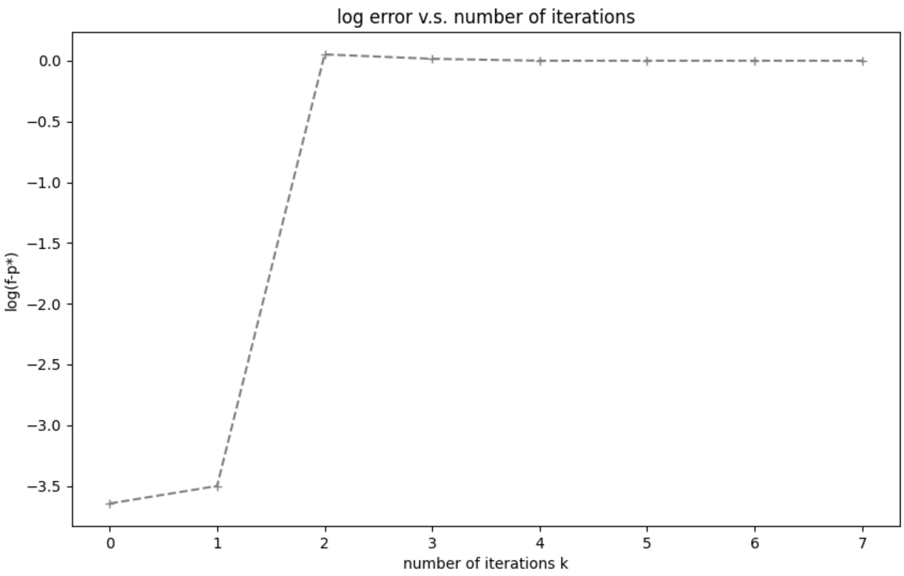
---

(2)我们设置回溯参数$\alpha = 0.1 \quad \beta = 0.5$得到最优值为$p^* = -31.012830219815058$最优解如下：

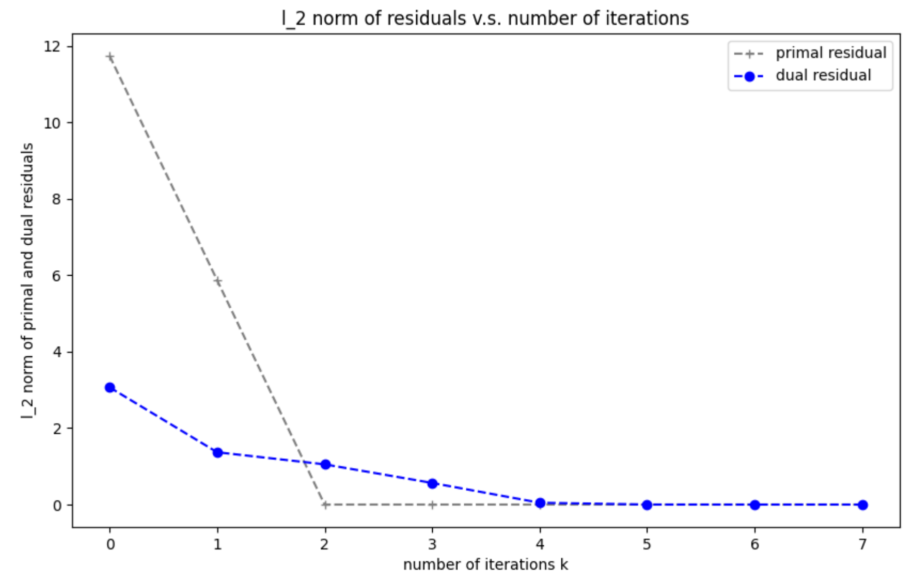
```
[0.17532642 0.54611282 0.76368044 0.96131916 0.37445166 0.3527156
 0.36777798 1.02142455 0.37812831 0.25007758 0.18606974 0.46315939
 0.37151271 0.48309865 0.30150414 0.08321218 0.36235094 0.88603499
 0.53080102 0.63463179 0.42959254 0.64033416 0.10741741 0.33371242
 0.44113777 0.71131122 0.23786689 0.34488757 0.38944621 0.40185213
 0.56908289 0.4489786  0.33322532 0.22082084 0.43813891 0.39601092
 0.3475417  0.44491479 0.62185117 0.09579495 0.3485656  0.43462242
 0.63587205 0.28342355 0.84877756 0.29001809 0.30134103 0.63604236
 0.5311949  0.49482199 0.50632888 0.4759541  0.48189151 0.25931878
 0.38322847 0.1634453  0.33063722 0.48008413 0.88358649 0.3790492
 0.40072391 0.62255794 0.26867464 0.31341895 0.6235641  0.53191447
 0.12308642 0.53365359 0.4167203  0.37954069 0.39805615 0.32331447
 0.35841583 0.35528431 0.92949532 0.27903386 0.97611158 0.4973101
 0.47220904 0.29674416 0.27887256 0.36241406 0.22652063 0.29707326
 0.1071736  0.58546588 0.94957098 0.79686147 0.51160996 0.83246085
 0.87389594 0.65444774 0.53527937 0.27798009 0.28755737 0.19735683
 0.39527824 0.55940325 0.55001295 0.4225168 ]
```

考虑到$\log\left(f\left(x^{(k)}\right)-p^{**}\right)$ 取对数的对象是正负交替的且最后会变成零不便于画图，我们不取对数画和迭代次数 k 的关系图如下：

```
#error
[-3.6445288081822085, -3.500412138771651, 0.05198637670705253, 0.015702459466520935, 8.803973978288582e-05, 1.0482501977548964e-08, 0.0, 0.0]
```



为了方便观察求解过程，二范数残差和迭代次数的关系图如下：



二、分别用障碍函数法和原对偶内点法求解下述二次规划问题

$$
\begin{aligned}
&\min (1/2) x^T P x + q^T x \\
&\text{s.t. } Ax = b \\
&x \geq 0
\end{aligned}
$$

其中 $x \in R^n$， $P \in S_+^n$， $q \in R^n, A \in R^{m \times n}$， $b \in R^m$

障碍函数法要求:

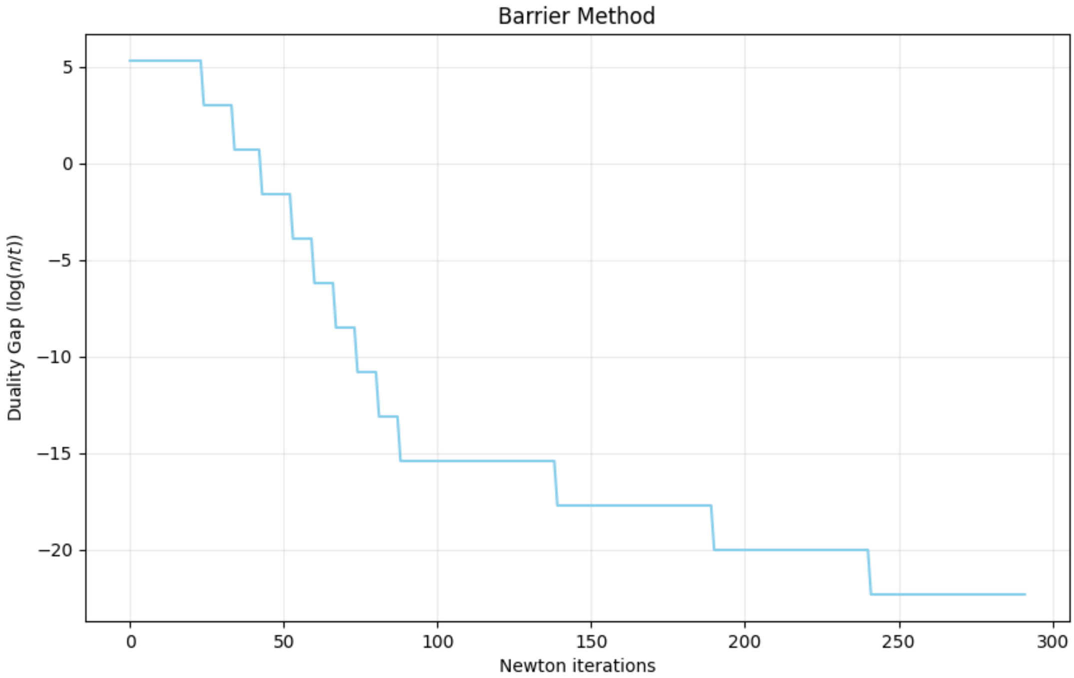
- 间值误差 $\varepsilon = 10^{-10}$;
- 请画出对数对偶间隙 $\log\left(\frac{n}{t}\right)$ 与 Newton 迭代次数 k 之间的关系;
- 给出原对偶最优解 x^*， λ^*， v^* 和最优值 p^*;（障碍函数法中参数 μ 建议选取 $\mu = 10$ ）

原对偶内点法要求:

- 原残差 $\|r_{\text{pri}}\|_2 \le 10^{-10}$, 对偶残差 $\|r_{\text{dual}}\|_2 \le 10^{-10}$, 代理对偶间隙 $\hat{\eta} \le 10^{-10}$;
- 给出最优解 $x^*, \lambda^*,\quad v^*$ 和最优值 p^*;
- 分别画出 $\log \hat{\eta}$ 和 $\log\left\{\left(\|r_{\text{pri}}\|_2^2 + \| r_{\text{dual}}\|_2^2\right)^{1/2}\right\}$ 与 Newton 迭代次数 k 的关系图。

---

(1)对数对偶间隙 $\log\left(\frac{n}{t}\right)$ 与 Newton 迭代次数 k 之间的关系:



原对偶最优解 $x^*,\quad \lambda^*,\quad v^*$ 和最优值 p^* 如下:

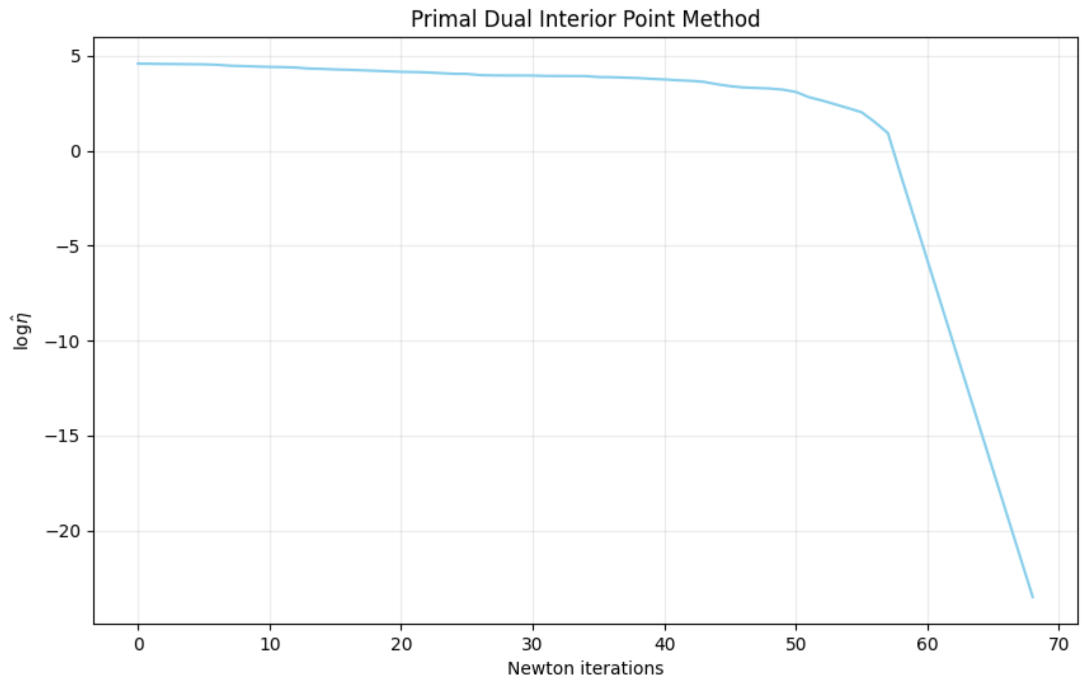
```
p* = 222682.4520036106
x* =
[8.34718428e-15 2.12590139e-01 4.53901860e-01 3.82793348e-01
 5.12300523e-01 2.61795330e-01 5.67570969e-01 1.64901573e+00
 5.02380629e-01 1.56335852e-15 1.10772380e-15 3.26293270e-15
 1.76847057e-01 4.20913996e-01 1.05296866e-15 1.50686491e+00
 9.91952174e-16 2.04599999e-15 2.20073913e+00 9.85705280e-01
 2.03692992e-15 3.32193637e-15 3.63667107e-01 2.26717904e-15
 1.16589039e-01 1.91581022e-15 9.82747809e-01 2.20029930e-14
 2.92120713e-14 3.67911296e-15 1.39413113e+00 5.01624955e-01
 7.60520473e-16 2.23413148e+00 6.44344673e-01 7.83639314e-01
 9.45214033e-01 4.18817913e-01 3.05217518e-15 1.98664088e-14
 3.27021966e-15 2.26870884e+00 2.96176061e+00 7.92559999e-15
 8.19643708e-01 3.71355331e-14 6.23578376e-15 5.17631668e-01
 8.72729658e-16 7.26166698e-15 1.82032698e-14 8.78635110e-01
 1.44467589e-15 1.19549802e-14 5.04419222e-01 2.66972955e-15
 1.99491531e-15 1.30091790e-15 6.89884991e-14 5.84236461e-01
 2.19986537e-15 6.03578860e-01 5.02374587e-15 1.27652705e+00
```

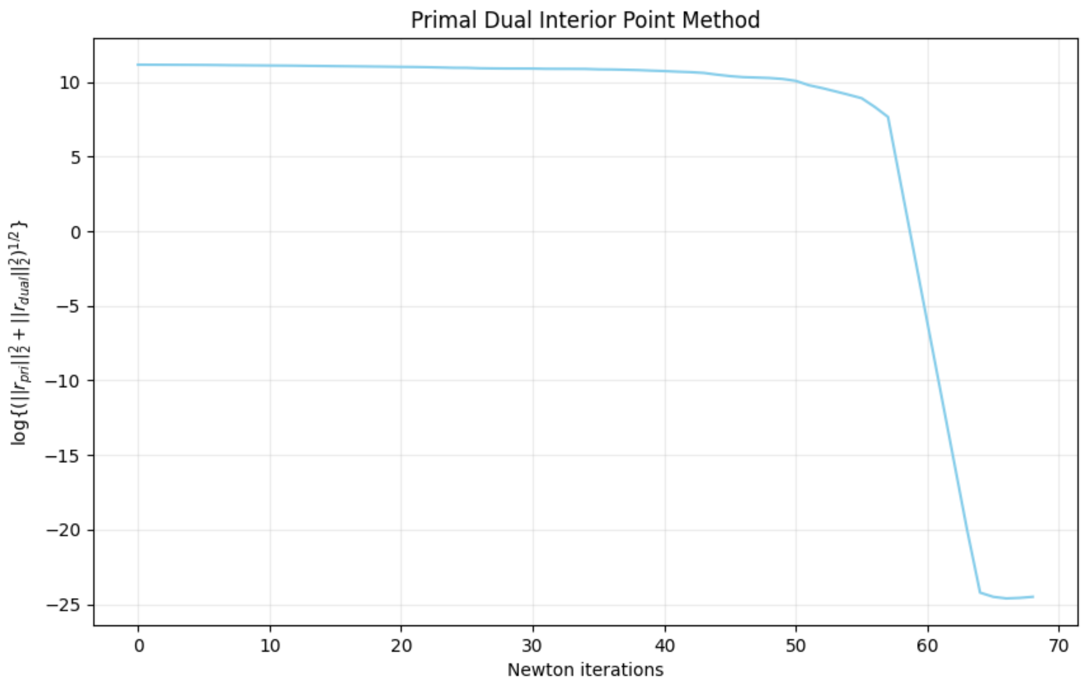
```
 1.40464270e+00 1.96017520e+00 3.59549874e-01 2.52196755e-01
 3.14525598e-15 1.98178398e-15 1.17795128e-14 2.13950164e-15
 1.61912219e-15 2.13265992e+00 3.67070483e-01 3.01442247e-15
 1.42564288e+00 5.28465485e-15 1.24424923e+00 8.01298912e-01
 1.38443005e-15 1.19133241e-15 1.67003254e+00 5.33942127e-15
 6.68792819e-01 9.47445188e-16 2.89947875e-15 1.12121475e+00
 4.36543143e-01 1.04427236e+00 1.43016510e-01 2.53721716e-15
 4.08771698e-01 6.40513067e-01 7.39820851e-01 3.49407839e-01
 1.54521101e+00 2.44334205e-15 2.79881855e-01 8.05132850e-01
 1.62922375e-15 4.43067009e-01 3.50144381e-15 3.59257604e-15
 9.15144703e-15 1.55692751e-15 1.32463380e-14 3.61298690e-01
 4.73735969e-15 6.61253478e-01 1.33092585e-15 1.25778430e-15
 1.82374327e-15 7.71607518e-15 5.91748793e-15 1.92787107e+00
 1.39297985e-15 6.83277911e-01 2.85504977e+00 2.44583053e+00
 3.05405034e-15 1.75398954e-15 1.48843137e+00 1.30167131e+00
 4.34533050e-15 1.80593637e-01 2.17281564e-15 1.76321990e+00
 1.97785601e-15 3.98848043e-01 1.99521875e-15 1.48174643e+00
 3.13930308e-14 1.99776368e-15 3.60952712e-15 1.83580529e-14
 6.36007963e-01 3.23336526e-01 1.53964423e+00 1.18848765e-15
 8.16104668e-15 8.64899184e-01 8.67185572e-01 1.07796169e-15
 7.28778693e-01 1.42244011e-15 1.80111566e-15 1.99048695e+00
 7.38402740e-15 2.28683843e-15 4.20359580e-15 2.29177722e-15
 8.81514335e-01 1.01282719e+00 6.68690165e-02 1.20968196e+00
 1.31569373e-01 3.65447609e-14 1.09685385e+00 3.39719036e-14
 2.71964295e-15 4.76419858e-15 1.74686947e+00 1.19481823e-15
 5.69542561e-01 1.14551761e-14 1.15490182e-01 4.33812054e-15
 2.96722434e-15 9.64442787e-02 2.60454272e-01 3.08604909e-01
 1.67611976e-14 4.80839729e-15 2.58530774e-15 3.96966187e-01
 2.70006456e+00 2.85140183e-01 1.78333203e-01 9.22912636e-15
 1.26344406e+00 9.54018642e-15 8.48399014e-01 1.82020276e+00
 1.49543607e-15 1.40241690e-15 9.67412166e-16 9.67608747e-15
 1.33644449e-14 3.63212894e-01 3.76925194e-02 2.52647608e-15
 1.33350641e+00 1.31788912e-15 1.82518973e+00 2.24838522e-15
 3.83910511e-01 2.77080457e+00 2.20758452e+00 3.47032357e-01]
lambda* =
[1.19800877e+02 4.70388705e-12 2.20311941e-12 2.61237559e-12
 1.95197927e-12 3.81977784e-12 1.76189420e-12 6.06422352e-13
 1.99052261e-12 6.39648542e+02 9.02752111e+02 3.06472763e+02
 5.65460358e-12 2.37578225e-12 9.49695884e+02 6.63629495e-13
 1.00811312e+03 4.88758557e+02 4.54392792e-13 1.01450202e-12
 4.90934906e+02 3.01029246e+02 2.74976753e-12 4.41076767e+02
 8.57713562e-12 5.21972368e+02 1.01755505e-12 4.54483625e+01
 3.42324237e+01 2.71804647e+02 7.17292641e-13 1.99352124e-12
 1.31488899e+03 4.47601230e-13 1.55196441e-12 1.27609728e-12
 1.05796144e-12 2.38767247e-12 3.27635192e+02 5.03362239e+01
 3.05789856e+02 4.40779346e-13 3.37637011e-13 1.26173413e+02
 1.22004231e-12 2.69283868e+01 1.60364765e+02 1.93187562e-12
 1.14583020e+03 1.37709427e+02 5.49351853e+01 1.13812889e-12
 6.92196779e+02 8.36471481e+01 1.98247798e-12 3.74569776e+02
```

```
 5.01274413e+02 7.68688017e+02 1.44951697e+01 1.71163573e-12
 4.54573273e+02 1.65678434e-12 1.99054655e+02 7.83375489e-13
 7.11924819e-13 5.10158479e-13 2.78125532e-12 3.96515808e-12
 3.17939146e+02 5.04595864e+02 8.48931546e+01 4.67398566e+02
 6.17618614e+02 4.68898014e-13 2.72427244e-12 3.31738504e+02
 7.01437937e-13 1.89227117e+02 8.03697506e-13 1.24797374e-12
 7.22318905e+02 8.39396282e+02 5.98790730e-13 1.87286215e+02
 1.49523137e-12 1.05547003e+03 3.44889577e+02 8.91889800e-13
 2.29072433e-12 9.57604587e-13 6.99219969e-12 3.94132601e+02
 2.44635332e-12 1.56124840e-12 1.35167858e-12 2.86198502e-12
 6.47160805e-13 4.09275483e+02 3.57293616e-12 1.24203105e-12
 6.13789234e+02 2.25699494e-12 2.85596473e+02 2.78351798e+02
 1.09272337e+02 6.42290663e+02 7.54925624e+01 2.76779304e-12
 2.11088046e+02 1.51227938e-12 7.51356660e+02 7.95048882e+02
 5.48322791e+02 1.29599567e+02 1.68990628e+02 5.18706886e-13
 7.17885473e+02 1.46353334e-12 3.50256592e-13 4.08859071e-13
 3.27434026e+02 5.70128829e+02 6.71848243e-13 7.68243096e-13
 2.30132092e+02 5.53729365e-12 4.60232328e+02 5.67144234e-13
 5.05597978e+02 2.50722052e-12 5.01198176e+02 6.74879304e-13
 3.18542038e+01 5.00559705e+02 2.77044601e+02 5.44720078e+01
 1.57230736e-12 3.09275297e-12 6.49500696e-13 8.41405462e+02
 1.22533302e+02 1.15620412e-12 1.15315572e-12 9.27676754e+02
 1.37215867e-12 7.03017295e+02 5.55211429e+02 5.02389628e-13
 1.35427450e+02 4.37284937e+02 2.37891569e+02 4.36342587e+02
 1.13441150e-12 9.87335261e-13 1.49546091e-11 8.26663564e-13
 7.60055305e-12 2.73637035e+01 9.11698491e-13 2.94360897e+01
 3.67695325e+02 2.09898891e+02 5.72452617e-13 8.36947393e+02
 1.75579503e-12 8.72967811e+01 8.65874467e-12 2.30514572e+02
 3.37015299e+02 1.03686814e-11 3.83944557e-12 3.24038915e-12
 5.96616079e+01 2.07969504e+02 3.86801148e+02 2.51910624e-12
 3.70361515e-13 3.50704692e-12 5.60748075e-12 1.08352618e+02
 7.91487359e-13 1.04819755e+02 1.17869067e-12 5.49389344e-13
 6.68701272e+02 7.13054726e+02 1.03368557e+03 1.03347557e+02
 7.48254046e+01 2.75320622e-12 2.65304632e-11 3.95808219e+02
 7.49902658e-13 7.58789175e+02 5.47888246e-13 4.44763643e+02
 2.60477369e-12 3.60906002e-13 4.52983788e-13 2.88157568e-12]
mu* =
[ -45.78680356   36.47177127 -199.54682785   80.29408776 -133.09998487
 -119.52975537   -9.49274025 -182.68322742   34.74052036  196.35278259
  -79.85053106  -27.23991717 -245.7006087  -164.20917171 -106.33616142
   59.98697152  136.08987283 -136.42022875   96.59125329 -242.4508552
 -247.52341946 -238.64526426 -108.62480172  -36.82229781   54.99386825
 -102.71576238  128.84543629  122.68792677 -138.94783326  -77.69363907
  -22.59844225 -115.19472488  -68.00841425   61.16908091  -69.39024396
 -310.93394686   50.08444061 -220.4333726   195.34453418  146.41657504
 -200.69950043  -87.05967062 -194.93827181  -43.50143318  111.33441053
 -108.61324554 -227.66686145 -156.40347888   -6.87805354 -271.61652165
  -57.2029665   -69.25774137  126.4979442   -59.77508917  -73.21931481
 -383.62324731  171.82117289 -142.83051121 -120.84811341 -121.44445182
```

```
  -12.10166684   51.17473253  139.52645234 -104.89539659  -80.70721269
 -178.04298511 -183.25751474 -152.10280155  -64.33469303 -147.62438603
    9.77053103 -126.06740991  -91.59503958 -187.94703208  -15.94616381
  177.07700176 -264.87664511  -13.93323391  -74.23105564 -116.00736755
 -121.67096519    8.35168923   -6.21800938 -159.39974794   -0.43573633
 -198.95216373   86.38748083   61.03738022  -55.80870352  104.54765585
   50.61070019 -275.32762812  -62.01724242 -309.78487705  -41.3082416
 -123.94919502 -358.70405696 -131.71016336  123.54813945   52.18096367]
```

(2) $\log\hat{\eta}$ 和 $\log\left\{\left(\|r_{\mathrm{pri}}\|_2^2+\|r_{\mathrm{dual}}\|_2^2\right)^{1/2}\right\}$ 与 Newton 迭代次数 k 的关系图如下：





最优解 $x^*, \lambda^*, \quad v^*$ 和最优值 p^* 如下:

```
iter: 68, rdual = 0.0, rpri = 0.0, eta = 6.181931481426342e-11 p* = 222682.4520036118
x* =
[2.02280655e-15 2.12590139e-01 4.53901860e-01 3.82793348e-01
 5.12300523e-01 2.61795330e-01 5.67570969e-01 1.64901573e+00
 5.02380629e-01 3.78854917e-16 2.68439134e-16 7.90719514e-16
 1.76847057e-01 4.20913996e-01 2.55170100e-16 1.50686491e+00
 2.40383733e-16 4.95815343e-16 2.20073913e+00 9.85705280e-01
 4.93617364e-16 8.05018131e-16 3.63667107e-01 5.49414551e-16
 1.16589039e-01 4.64265949e-16 9.82747809e-01 5.33207355e-15
 7.07907836e-15 8.91574125e-16 1.39413113e+00 5.01624955e-01
 1.84299963e-16 2.23413148e+00 6.44344673e-01 7.83639314e-01
 9.45214033e-01 4.18817913e-01 7.39645809e-16 4.81430636e-15
 7.92485392e-16 2.26870884e+00 2.96176061e+00 1.92064225e-15
 8.19643708e-01 8.99920318e-15 1.51114242e-15 5.17631668e-01
 2.11492063e-16 1.75974872e-15 4.41127106e-15 8.78635110e-01
 3.50094079e-16 2.89709811e-15 5.04419222e-01 6.46966225e-16
 4.83435791e-16 3.15256628e-16 1.67182623e-14 5.84236461e-01
 5.33102162e-16 6.03578860e-01 1.21742441e-15 1.27652705e+00
 1.40464270e+00 1.96017520e+00 3.59549874e-01 2.52196755e-01
 7.62202437e-16 4.80253628e-16 2.85457631e-15 5.18473974e-16
 3.92368346e-16 2.13265992e+00 3.67070483e-01 7.30497039e-16
 1.42564288e+00 1.28065152e-15 1.24424923e+00 8.01298912e-01
 3.35494464e-16 2.88700342e-16 1.67003254e+00 1.29392330e-15
 6.68792819e-01 2.29598174e-16 7.02642279e-16 1.12121475e+00
 4.36543143e-01 1.04427236e+00 1.43016510e-01 6.14853962e-16
 4.08771698e-01 6.40513067e-01 7.39820851e-01 3.49407839e-01
 1.54521101e+00 5.92104839e-16 2.79881855e-01 8.05132850e-01
 3.94816300e-16 4.43067009e-01 8.48518868e-16 8.70603308e-16
 2.21770674e-15 3.77296460e-16 3.21003833e-15 3.61298690e-01
 1.14802328e-15 6.61253478e-01 3.22528578e-16 3.04803892e-16
 4.41954986e-16 1.86986730e-15 1.43400850e-15 1.92787107e+00
 3.37566371e-16 6.83277911e-01 2.85504977e+00 2.44583053e+00
 7.40100218e-16 4.25051290e-16 1.48843137e+00 1.30167131e+00
 1.05302130e-15 1.80593637e-01 5.26547091e-16 1.76321990e+00
 4.79301745e-16 3.98848043e-01 4.83509326e-16 1.48174643e+00
 7.60759889e-15 4.84126052e-16 8.74711140e-16 4.44878003e-15
 6.36007963e-01 3.23336526e-01 1.53964423e+00 2.88010959e-16
 1.97769898e-15 8.64899184e-01 8.67185572e-01 2.61226761e-16
 7.28778693e-01 3.44705597e-16 4.36471552e-16 1.99048695e+00
 1.78940085e-15 5.54178694e-16 1.01867414e-15 5.55375513e-16
 8.81514335e-01 1.01282719e+00 6.68690165e-02 1.20968196e+00
 1.31569373e-01 8.85603846e-15 1.09685385e+00 8.23255039e-15
 6.59061929e-16 1.15452731e-15 1.74686947e+00 2.89545072e-16
 5.69542561e-01 2.77597860e-15 1.15490182e-01 1.05127411e-15
```

```
 7.19059320e-16 9.64442787e-02 2.60454272e-01 3.08604909e-01
 4.06180796e-15 1.16523811e-15 6.26507949e-16 3.96966187e-01
 2.70006456e+00 2.85140183e-01 1.78333203e-01 2.23653109e-15
 1.26344406e+00 2.31191152e-15 8.48399014e-01 1.82020276e+00
 3.62394992e-16 3.39853290e-16 2.34436854e-16 2.34484495e-15
 3.23865925e-15 3.63212894e-01 3.76925194e-02 6.12251044e-16
 1.33350641e+00 3.19369334e-16 1.82518973e+00 5.44860166e-16
 3.83910511e-01 2.77080457e+00 2.20758452e+00 3.47032357e-01]
lambda* =
[1.52805801e+02 1.45395537e-12 6.80976663e-13 8.07476346e-13
 6.03350104e-13 1.18068024e-12 5.44595462e-13 1.87443071e-13
 6.15263718e-13 8.15870561e+02 1.15145869e+03 3.90905458e+02
 1.74781859e-12 7.34346153e-13 1.21133539e+03 2.05125603e-13
 1.28584647e+03 6.23410668e+02 1.40451255e-13 3.13579099e-13
 6.26186591e+02 3.83962251e+02 8.49943721e-13 5.62592624e+02
 2.65116324e-12 6.65774810e+02 3.14522781e-13 5.79693006e+01
 4.36633921e+01 3.46686344e+02 2.21712698e-13 6.16190584e-13
 1.67713856e+03 1.38352007e-13 4.79706882e-13 3.94437299e-13
 3.27012257e-13 7.38021380e-13 4.17898094e+02 6.42037608e+01
 3.90034412e+02 1.36243386e-13 1.04362443e-13 1.60933966e+02
 3.77110897e-13 3.43471047e+01 2.04544965e+02 5.97136136e-13
 1.46150437e+03 1.75648132e+02 7.00697304e+01 3.51791740e-13
 8.82895748e+02 1.06691787e+02 6.12777152e-13 4.77763077e+02
 6.39374617e+02 9.80460194e+02 1.84885587e+01 5.29060739e-13
 5.79807392e+02 5.12106362e-13 2.53893854e+02 2.42138680e-13
 2.20053523e-13 1.57688238e-13 8.59676490e-13 1.22561678e-12
 4.05530813e+02 6.43611118e+02 1.08281069e+02 5.96166037e+02
 7.87771432e+02 1.44934770e-13 8.42063278e-13 4.23131864e+02
 2.16812063e-13 2.41358847e+02 2.48420146e-13 3.85744408e-13
 9.21316466e+02 1.07064845e+03 1.85084163e-13 2.38883228e+02
 4.62170893e-13 1.34625014e+03 4.39906028e+02 2.75680081e-13
 7.08055044e-13 2.95992297e-13 2.16126497e-12 5.02715430e+02
 7.56159431e-13 4.82576531e-13 4.17799219e-13 8.84629764e-13
 2.00035188e-13 5.22030143e+02 1.10438233e-12 3.83907543e-13
 7.82887063e+02 6.97629405e-13 3.64277785e+02 3.55037215e+02
 1.39376667e+02 8.19240588e+02 9.62906177e+01 8.55515347e-13
 2.69242427e+02 4.67440375e-13 9.58354066e+02 1.01408342e+03
 6.99384742e+02 1.65304016e+02 2.15547239e+02 1.60330521e-13
 9.15661633e+02 4.52373140e-13 1.08263113e-13 1.26376938e-13
 4.17641512e+02 7.27198297e+02 2.07665990e-13 2.37461309e-13
 2.93533069e+02 1.71155850e-12 5.87025509e+02 1.75302340e-13
 6.44889315e+02 7.74973274e-13 6.39277378e+02 2.08602881e-13
 4.06299778e+01 6.38463005e+02 3.53369884e+02 6.94789519e+01
 4.85994817e-13 9.55959347e-13 2.00758440e-13 1.07321116e+03
 1.56291011e+02 3.57378732e-13 3.56436482e-13 1.18325004e+03
 4.24129543e-13 8.96697288e+02 7.08171180e+02 1.55286913e-13
 1.72737470e+02 5.57756149e+02 3.03430274e+02 5.56554199e+02
 3.50642709e-13 3.05181947e-13 4.62241843e-12 2.55518876e-13
 2.34930491e-12 3.49023523e+01 2.81802880e-13 3.75456644e+01
```

```
 4.68994734e+02 2.67725650e+02 1.76943143e-13 1.06752490e+03
 5.42710230e-13 1.11346886e+02 2.67638831e-12 2.94020915e+02
 4.29862412e+02 3.20492390e-12 1.18675947e-12 1.00159319e-12
 7.60982737e+01 2.65264731e+02 4.93364170e+02 7.78647109e-13
 1.14477475e-13 1.08401619e-12 1.73325309e-12 1.38203567e+02
 2.44646031e-13 1.33697406e+02 3.64329247e-13 1.69814364e-13
 8.52927279e+02 9.09500019e+02 1.31846409e+03 1.31819621e+02
 9.54396711e+01 8.51006612e-13 8.20047529e-12 5.04852670e+02
 2.31792343e-13 9.67834231e+02 1.69350380e-13 5.67295232e+02
 8.05126625e-13 1.11554809e-13 1.40015737e-13 8.90685171e-13]
mu* =
[ -86.02023026   19.77639117 -255.65851714   62.09791979 -199.34404049
 -136.417844    -32.88401416 -208.01157108   50.05220642  259.50577871
  -77.07137326  -39.78622134 -300.73847398 -216.43879421 -181.34392075
   60.53792319  152.84399412 -175.77870603  105.2644989  -315.62638489
 -321.87741421 -322.4980743  -157.75724321  -52.49748811   67.46306936
 -128.05780538  155.70057828  155.04088322 -178.32558174  -88.09920627
  -41.5205585  -134.78181307  -92.14200563   58.49973475  -70.26715986
 -370.39171331   66.8495956  -276.26519423  238.87361474  170.59009071
 -264.53877866 -103.4455381  -281.22875829  -55.64354431  149.30969334
 -164.62479617 -292.9867745  -183.94051631   -8.4506149  -343.8628887
  -76.11012785  -81.09973987  149.85619471  -79.77574666  -70.40705342
 -465.06868413  217.19029734 -193.56030897 -139.0994289  -139.02550712
  -26.88609787   29.27017739  168.35828853 -114.31825814 -116.27567003
 -243.99196217 -208.65717306 -194.68363278  -59.22565295 -188.88340142
   -4.79087819 -157.95100521 -141.29165952 -177.74728032   -1.99535186
  220.10941285 -333.40454155   -8.67052831  -98.43638197 -148.63001008
 -183.18034613   22.92440535  -41.69690553 -187.09673509   13.86785517
 -296.94135664   98.49294767   94.14329233  -73.3503076   130.97930676
   78.18347969 -360.55604788 -120.21166275 -379.16334449  -39.20498087
 -143.01013044 -431.69849829 -175.04790157  153.45153417   71.80421005]
```

三、附录

Code1:

```python
import numpy as np
import matplotlib.pyplot as plt
import mat4py
import math

class obj_func(object):
    def __init__(self,a):
        self.a=np.array(a)

```

```
    def check(self,x):
        if np.any(x<=0):
            return False
        else:
            return  True

    def call_func(self,x):
        return np.dot(x,np.log(x))

    def call_grad(self,x):
        return 1+np.log(x)

    def call_hessian(self,x):
        return np.diag(1/x)

    def call_direction(self,hessian,grad):
        zero1=np.zeros(self.a.shape[0])
        tmp1=np.concatenate((-grad,zero1))
        zero2 = np.zeros((self.a.shape[0], self.a.shape[0]))
        l=np.concatenate((hessian,self.a))
        r=np.concatenate((self.a.T,zero2))
        tmp2=np.concatenate((l,r),axis=1)
        return -np.matmul(np.linalg.inv(tmp2),tmp1)[:self.a.shape[1]]


class solve(object):
    def __init__(self, obj_func):
        self.obj_func = obj_func
        self.process = []
        self.stepsize = []
        self.current_x=[]

    def backtrack(self, x, direction, gradient, alpha=0.2, beta=0.8):
        t = 1.0
        while True:
            left = x - t * direction
            if self.obj_func.check(left):
                if self.obj_func.call_func(left) <= self.obj_func.call_func(x) - alpha * t * np.dot(gradient, direction):
                    break
            t *= beta
        print("backtrack finished")
        return t

    def search(self, initial):
        self.process = []
        if not self.obj_func.check(initial):
            raise ValueError("Initial point is not correct")
        else:
```

```
            ctr = 0
            current_x = initial
            self.current_x.append(current_x)
            eta = float('inf')
            while True:
                ctr += 1
                print("iteration {:d}".format(ctr))
                gradient = self.obj_func.call_grad(current_x)
                self.process.append(self.obj_func.call_func(current_x))
                if eta <= 1e-10:
                    break
                hessian = self.obj_func.call_hessian(current_x)
                direction = self.obj_func.call_direction(hessian, gradient)
                eta = np.matmul(np.matmul(direction, hessian), direction) / 2
                t = self.backtrack(current_x, direction, gradient)
                self.stepsize.append(t)
                current_x -= t * direction
                self.current_x.append(current_x)
                print("stepsize:", t)
                print("eta", eta)
        print("total number of iterations {:d}".format(ctr))
        print("optimum", self.process[-1])
        print("current_x", current_x)
        return current_x

    def plot(self, name):
        optimum = self.process[-1]
        y = [math.log(f - optimum) for f in self.process[:-1]]
        plt.figure(figsize=(10, 6))
        plt.plot(y, color="gray", linestyle='--', marker='+')
        plt.xlabel("number of iterations k")
        plt.ylabel("log(f-p*)")
        plt.title("log error v.s. number of iterations")
        plt.savefig(name+".png")

if __name__ == "__main__":
    A = mat4py.loadmat('./A.mat')['A']
    x_0 = mat4py.loadmat('./x_0.mat')['x_0']
    x_0 = np.array([x[0] for x in x_0])
    obj_func = obj_func(A)
    solver = solve(obj_func)
    solver.search(initial=x_0)
    solver.plot(name="p1")
```

Code2:

```python
import numpy as np
import matplotlib.pyplot as plt
import mat4py
import math

class obj_func(object):
    def __init__(self,a,b):
        self.a=np.array(a)
        self.b=np.array(b)

    def check(self,x):
        if np.any(x<=0):
            return False
        else:
            return  True

    def call_func(self,x):
        return np.dot(x,np.log(x))

    def call_grad(self,x):
        return 1+np.log(x)

    def call_hessian(self,x):
        return np.diag(1/x)

    def call_direction(self,hessian,grad,x,v):
        zero = np.zeros((self.a.shape[0], self.a.shape[0]))
        l = np.concatenate((hessian,self.a))
        r = np.concatenate((self.a.T,zero))
        tmp = np.concatenate((l,r),axis=1)

        r_pri = np.matmul(self.a, x) - self.b
        rhs = - np.concatenate((grad, r_pri))
        tmp2 = np.matmul(np.linalg.inv(tmp), rhs)

        direction_x = - tmp2[: self.a.shape[1]]
        next_v = tmp2[self.a.shape[1]:]
        direction_v = v - next_v
        return direction_x, direction_v

    def residual_norm(self,x,v,grad):
        r_pri = np.matmul(self.a, x) - self.b
        r_dual = grad + np.matmul(self.a.T, v)
        r = np.concatenate((r_pri, r_dual))
        return np.linalg.norm(r_pri, 2), np.linalg.norm(r_dual, 2),
np.linalg.norm(r, 2)


```

```
class solve(object):
    def __init__(self, obj_func):
        self.obj_func = obj_func
        self.process = []
        self.stepsize = []
        self.r_dual = []
        self.r_pri = []

    def backtrack(self, x, v, direction_x, direction_v, gradient, alpha=0.1, beta=0.5):
        t = 1.0
        while True:
            left_x = x - t * direction_x
            left_v = v - t*direction_v
            if self.obj_func.check(left_x):
                _,_,left = self.obj_func.residual_norm(left_x,left_v,gradient)
                _, _, right = self.obj_func.residual_norm(x, v, gradient)
                if left <= (1-alpha*t)*right:
                    break
            t *= beta
        print("backtrack finish")
        return t

    def search(self, initial,v):
        self.process = []
        self.stepsize = []
        self.r_dual = []
        self.r_pri = []
        if not self.obj_func.check(initial):
            raise ValueError("initial point is not correct")
        else:
            ctr = 0
            current_x = initial
            current_v = v
            eta = float('inf')
            while True:
                ctr += 1
                print("iteration {:d}".format(ctr))
                gradient = self.obj_func.call_grad(current_x)
                self.process.append(self.obj_func.call_func(current_x))
                if eta <= 1e-10:
                    break
                hessian = self.obj_func.call_hessian(current_x)
                direction_x,direction_v = self.obj_func.call_direction(hessian, gradient,current_x,current_v)
                eta_pri,eta_dual,eta = self.obj_func.residual_norm(current_x,current_v,gradient)
                self.r_dual.append(eta_dual)
```

```
                self.r_pri.append(eta_pri)
                t = self.backtrack(current_x, current_v, direction_x, direction_v, gradient)
                self.stepsize.append(t)
                current_x -= t * direction_x
                current_v -= t * direction_v
                print("stepsize:", t)

        print("total number of iterations {:d}".format(ctr))
        print("optimum", self.process[-1])
        print("current_x", current_x)
        print("current_v", current_v)
        return current_x

    def plot(self, name):
        optimum = self.process[-1]
        y = [(f - optimum) for f in self.process[:-1]]
        print('y ',y)
        plt.figure(figsize=(10, 6))
        plt.plot(y, color="gray", linestyle='--', marker='+')
        plt.xlabel("number of iterations k")
        plt.ylabel("log(f-p*)")
        plt.title("log error v.s. number of iterations")
        plt.savefig(name + "_1.png")

        plt.figure(figsize=(10, 6))
        plt.plot(self.r_pri, color="gray", linestyle='--', marker='+',label='primal residual')
        plt.plot(self.r_dual, color="blue", linestyle='--', marker='o', label='dual residual')
        plt.legend()
        plt.xlabel("number of iterations k")
        plt.ylabel("l_2 norm of primal and dual residuals")
        plt.title("l_2 norm of residuals v.s. number of iterations")
        plt.savefig(name+"_2.png")

if __name__ == "__main__":
    A = mat4py.loadmat('./A.mat')['A']
    b = mat4py.loadmat('./b.mat')['b']
    b = [temp[0] for temp in b]
    x_1 = mat4py.loadmat('./x_1.mat')['x_1']
    x_1 = np.array([x[0] for x in x_1])
    obj_func = obj_func(A, b)
    solver = solve(obj_func)
    zeros = np.zeros(len(A))
    solver.search(initial=x_1, v=zeros)
    solver.plot("p2")
```

Code3:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import math
from scipy import linalg
from sklearn.linear_model import LinearRegression

def f(x, P, q):
    return (0.5 * (x.reshape(-1, 1)).T @ P @ x.reshape(-1, 1) + np.sum(q * x))[0][0]


def df(x, P, q):
    return P @ x + q


def H(x, P, q):
    return P


def F(x, P, q, t):
    return (t * (0.5 * (x.reshape(-1, 1)).T @ P @ x.reshape(-1, 1) +
                 np.sum(q * x)) - np.sum(np.log(x)))[0][0]


def dF(x, P, q, t):
    return t * (P @ x + q) - 1 / x


def HF(x, P, q, t):
    return t * P + np.diag(1 / x**2)

def back_search(func, df, x, d, alpha, beta, P, q, t):
    ans = 1
    fx = func(x, P, q, t)
    dfx = df(x, P, q, t)
    while func(x + ans * d, P, q, t) > fx + alpha * ans * np.sum(dfx * d) or \
            np.isnan(func(x + ans * d, P, q, t)):
        ans = beta * ans
    return ans

def feasible_newton(f, df, H, x0, A, b, P, q, t, alpha, beta, ttrack, eta=1e-10):
    m = A.shape[0]
    n = A.shape[1]
    dfx0 = df(x0, P, q, t)
```

```python
    Hx0 = H(x0, P, q, t)
    x = x0.copy()
    Alarge = np.r_[np.c_[Hx0, A.T], np.c_[A, np.zeros((m, m))]]
    blarge = np.r_[-dfx0, np.zeros(m)]
    deltax = linalg.solve(Alarge, blarge)[:n]
    lambdax = (deltax.reshape(-1, 1).T @ Hx0 @ deltax.reshape(-1, 1))**0.5
    ttrack.append(t)
    indi = 0
    while lambdax**2 / 2 > eta and indi < 50:
        indi += 1
        tt = back_search(f, df, x, deltax, alpha, beta, P, q, t)
        x += tt * deltax
        dfx = df(x, P, q, t)
        Hx = H(x, P, q, t)
        Alarge = np.r_[np.c_[Hx, A.T], np.c_[A, np.zeros((m, m))]]
        blarge = np.r_[-dfx, np.zeros(m)]
        deltax = linalg.solve(Alarge, blarge)[:n]
        lambdax = ((deltax.reshape(-1, 1).T @ Hx @ deltax.reshape(-1, 1))**0.5)[0][0]
        ttrack.append(t)
        print('\r' + 'iter: ' + str(indi) + ', error = ' + str(round(lambdax**2 / 2, 10)) +
              ', t = ' + str(t), end='', flush=True)
    return x, ttrack


def barrier(f, df, H, x0, A, b, P, q, alpha, beta, t, mu=10, epsilon=1e-10):
    m = x0.shape[0]
    x = x0.copy()
    ttrack = []
    i = 0
    while m / t >= epsilon and i < 15:
        i += 1
        xnew, ttrack = feasible_newton(f, df, H, x, A, b, P, q, t, alpha, beta, ttrack)
        x = xnew
        t = mu * t
    return x, ttrack


if __name__ == "__main__":
    P = loadmat('./P.mat')['P']  # shape = (200, 200)
    q = loadmat('./q.mat')['q'].reshape(-1)  # shape = (200,)
    A = loadmat('./A.mat')['A']  # shape = (100, 200)
    b = loadmat('./b.mat')['b'].reshape(-1)  # shape = (100,)
    x0 = loadmat('./x_0.mat')['x_0'].reshape(-1)  # shape = (200,)
```

```
    lambda0 = loadmat('./lambda.mat')['lambda'].reshape(-1)  # shape = (200,)
    mu0 = loadmat('./mu.mat')['mu'].reshape(-1)  # shape = (100,)

    x_star, ttrack = barrier(F, dF, HF, x0, A, b, P, q, alpha=0.01, beta=0.5, t=1)
    p_star = f(x_star, P, q)
    lambda_star = 1 / ttrack[-1] / x_star
    model = LinearRegression()
    model.fit(A.T, -df(x_star, P, q) + lambda_star)
    mu_star = model.coef_
    print(' p* = ' + str(round(p_star, 10)))
    print('x* = ')
    print(x_star)
    print('lambda* = ')
    print(lambda_star)
    print('mu* = ')
    print(mu_star)
    plt.figure(figsize=(10, 6))
    plt.plot(np.log(x0.shape[0] / np.array(ttrack)), color='skyblue')
    plt.xlabel('Newton iterations')
    plt.ylabel('Duality Gap ($\log(n/t)$)')
    plt.grid(alpha=0.25)
    plt.title('Barrier Method')
    plt.savefig('1.png')
```

Code4:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import math
from scipy import linalg

def f(x, P, q):
    return (0.5 * (x.reshape(-1, 1)).T @ P @ x.reshape(-1, 1) + np.sum(q * x))[0][0]


def df(x, P, q):
    return P @ x + q


def H(x, P, q):
    return P


def F(x, P, q, t):
```

```
    return (t * (0.5 * (x.reshape(-1, 1)).T @ P @ x.reshape(-1, 1) +
                 np.sum(q * x)) - np.sum(np.log(x)))[0][0]


def dF(x, P, q, t):
    return t * (P @ x + q) - 1 / x


def HF(x, P, q, t):
    return t * P + np.diag(1 / x**2)
def r(x, lamb, mu, t, A, b, P, q):
    m = x.shape[0]
    rdual = df(x, P, q) - lamb + A.T @ mu
    rcent = np.diag(lamb) @ x - np.ones(m) / t
    rpri = A @ x - b
    return np.r_[rdual, rcent, rpri]


def dual_search(x, lamb, mu, deltax, deltalambda, deltamu, t, r, A, b,
alpha, beta):
    w = -lamb / deltalambda
    w[w < 0] = 1
    w = np.append(w, [1])
    s = 0.99 * np.min(w)
    while (x + s * deltax < 0).any() or linalg.norm(r(x + s * deltax,
                 lamb + s * deltalambda, mu + s * deltamu, t, A, b, P,
q)) > \
                (1 - alpha * s) * linalg.norm(r(x, lamb, mu, t, A, b, P,
q)):
        s = s * beta
    return s


def dual(f, df, H, x0, lambda0, mu0, P, q, A, b, u, alpha, beta,
         eps_pri=1e-10, eps_dual=1e-10, eps_eta=1e-10):
    x = x0.copy()
    lamb = lambda0.copy()
    mu = mu0.copy()
    m = x0.shape[0]
    eta = np.sum(x0 * lambda0)
    etatrack = [eta]
    t = u * m / eta
    rdual = df(x0, P, q) - lambda0 + A.T @ mu0
    rcent = np.diag(lambda0) @ x0 - np.ones(m) / t
    rpri = A @ x - b
    rtrack = [(linalg.norm(rpri)**2 + linalg.norm(rdual)**2)**0.5]
    indi = 0
    while linalg.norm(rdual) > eps_dual or linalg.norm(rpri) > eps_pri or
eta > eps_eta:
```

```python
        line1 = np.c_[H(x, P, q), -np.diag(np.ones(m)), A.T]
        line2 = np.c_[np.diag(lamb), np.diag(x), np.zeros((m, A.shape[0]))]
        line3 = np.c_[A, np.zeros((A.shape[0], (A.shape[0] + m)))]
        AAmat = np.r_[line1, line2, line3]
        bbmat = -np.r_[rdual, rcent, rpri]
        sol = linalg.solve(AAmat, bbmat)
        deltax = sol[:m]
        deltalambda = sol[m: 2 * m]
        deltamu = sol[2 * m:]
        s = dual_search(x, lamb, mu, deltax, deltalambda, deltamu, t, r, A, b, alpha, beta)
        x = x + s * deltax
        lamb = lamb + s * deltalambda
        mu = mu + s * deltamu
        eta = np.sum(x * lamb)
        etatrack.append(eta)
        t = u * m / eta
        rdual = df(x, P, q) - lamb + A.T @ mu
        rcent = np.diag(lamb) @ x - np.ones(m) / t
        rpri = A @ x - b
        rtrack.append((linalg.norm(rpri)**2 + linalg.norm(rdual)**2)**0.5)
        indi += 1
        print('\r' + 'iter: ' + str(indi) + ', rdual = ' + str(round(linalg.norm(rdual), 10)) +
              ', rpri = ' + str(round(linalg.norm(rpri), 10)) +
              ', eta = ' + str(eta), end='', flush=True)
    return x, lamb, mu, etatrack, rtrack



if __name__ == "__main__":
    P = loadmat('./P.mat')['P']  # shape = (200, 200)
    q = loadmat('./q.mat')['q'].reshape(-1)  # shape = (200,)
    A = loadmat('./A.mat')['A']  # shape = (100, 200)
    b = loadmat('./b.mat')['b'].reshape(-1)  # shape = (100,)
    x0 = loadmat('./x_0.mat')['x_0'].reshape(-1)  # shape = (200,)
    lambda0 = loadmat('./lambda.mat')['lambda'].reshape(-1)  # shape = (200,)
    mu0 = loadmat('./mu.mat')['mu'].reshape(-1)  # shape = (100,)
    xdual, lambdual, mudual, etatrack, rtrack = dual(f, df, H, x0, lambda0, mu0, P, q,A, b, u=10, alpha=0.01, beta=0.5)
    pdual = f(xdual, P, q)
    print(' p* = ' + str(round(pdual, 10)))
    print('x* = ')
    print(xdual)
    print('lambda* = ')
    print(lambdual)
```

```
    print('mu* = ')
    print(mudual)
    plt.figure(figsize=(10, 6))
    plt.plot(np.log(etatrack), color='skyblue')
    plt.xlabel('Newton iterations')
    plt.ylabel('$\log \hat{\eta}$')
    plt.grid(alpha=0.25)
    plt.title('Primal Dual Interior Point Method')
    plt.savefig('2.png', )

    plt.figure(figsize=(10, 6))
    plt.plot(np.log(rtrack), color='skyblue')
    plt.xlabel('Newton iterations')
    plt.ylabel('$\log\{(||r_{pri}||_{2}^{2} + ||r_{dual}||_{2}^{2})^{1/2}\}$')
    plt.grid(alpha=0.25)
    plt.title('Primal Dual Interior Point Method')
    plt.savefig('3.png')

```