ESE 605 Homework 5

Xi Cao

Apr. 27, 2022

Problems from Boyd & Vandenberghe:
9.5.

From strong convexity, we have

$$f(y) \leq f(x) + \nabla f(x)^T(y-x) + \frac{M}{2}\|y-x\|_2^2$$
$$f(x+t\Delta x) \leq f(x) + t\nabla f(x)^T\Delta x + t^2\frac{M}{2}\Delta x^T\Delta x$$

The backtracking stopping condition is:

$$f(x+\Delta x) \leq f(x) + \alpha t\nabla f(x)^T\Delta x$$

So the backtracking stopping condition holds when:

$$f(x) + t\nabla f(x)^T\Delta x + t^2\frac{M}{2}\Delta x^T\Delta x \leq f(x) + \alpha t\nabla f(x)^T\Delta x$$
$$(1-\alpha)t\nabla f(x)^T\Delta x + t^2\frac{M}{2}\Delta x^T\Delta x \leq 0$$
$$t \leq -\frac{2(1-\alpha)\nabla f(x)^T\Delta x}{M\Delta x^T\Delta x}$$

Since $0 < \alpha < 0.5$ and $t > 0$,

$$0 < t \leq -\frac{\nabla f(x)^T\Delta x}{M\|\Delta x\|_2^2}$$

9.11.

For gradient method, when minimizing f,

$$\Delta x := -\nabla f(x)$$

when minimizing g,

$$\Delta x := -\nabla g(x) = -\phi'(f(x))\nabla f(x)$$

We can see that the direction Δx for minimizing g is just Δx for minimizing f multiplied by a scalar. So for exact line search, the two approaches are exactly equivalent.

For Newton's method, when minimizing f,

$$\Delta x := -\nabla^2 f(x)^{-1} \nabla f(x)$$

when minimizing g,

$$\Delta x := -\left(\phi''(f(x))\nabla f(x)\nabla f(x)^T + \phi'(f(x))\nabla^2 f(x)\right)^{-1} \nabla f(x)$$

Use the matrix inversion lemma, set

$$A = \phi'(f(x))\nabla^2 f(x)$$
$$B = \nabla f(x)\nabla f(x)^T$$
$$C = \phi''(f(x))$$

and we have

$$\Delta x = -\nabla^2 f(x)^{-1}(1 - A^{-1}B(I + CA^{-1}B))^{-1}C\nabla f(x)$$

which is Δx for minimizing f multiplied by a factor. So for exact line search, the two approaches are exactly equivalent.

9.26.

We can solve this problem using Newton's method directly. Newton's method solves:

$$H\Delta x = -g$$

But in this case, when A_i is very sparse, we could utilize this property and use a faster solution method. Since the function is convex, Hessian is symmetric positive definite, and we could use Cholesky factorization to further simplify the second computation,

$$H_i = LL^T$$

where L is lower triangular.

The solution is found by solving first for y and then z and finally for Δx in the following three linear systems:

$$\sum_{i=1}^{N} A_i^T y_i = -g$$
$$Lz_i = y_i$$
$$L^T A_i \Delta x = z_i$$

9.30. The parameter settings are as follows, and we generate a_i from normal distribution.

```
[9] m = 150
    n = 80
    alpha = 0.01
    beta = 0.5
```

```
[10] A = np.random.normal(1, 10, size=(m, n))
```

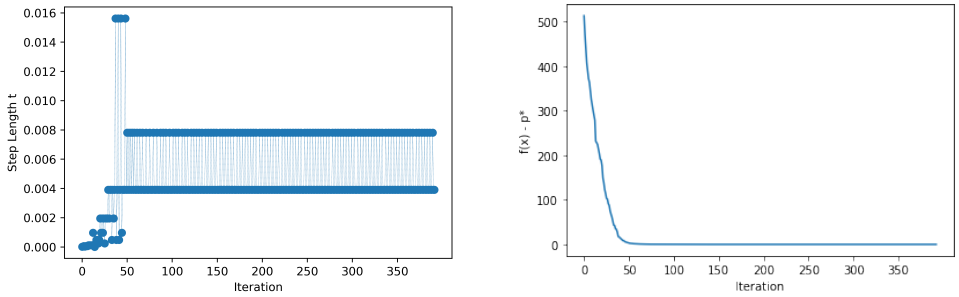
(a) For gradient method, set $x = \mathbf{0}$ as the feasible starting point. At each iteration, first calculate the gradient and determine whether the stop criterion is met, then find the descent direction Δx, check the feasibility of the new point, and use backtracking line search to determine step length.

The codes are:

```
[11] def f(x):
       return -sum(np.log(1-np.matmul(A,x)))-sum(np.log(1+x))-sum(np.log(1-x))
```

```
[12] eta = 1e-3
     x_history = []
     f_history = []
     t_history = []
     x = np.zeros((n,1))
     for _ in range(500):
       x_history.append(x)
       f_history.append(f(x))
       gradient = A.T @ (1/(1-A@x)) - 1/(1+x) + 1/(1-x)
       if np.linalg.norm(gradient, ord=2) <= eta:
         break
       delta_x = - gradient
       t = 1
       while max(np.matmul(A,x + t*delta_x)) >= 1 or max(abs(x + t*delta_x)) >=1:
         t = beta*t
       while f(x+t*delta_x) >= f(x) + alpha*t*np.matmul(gradient.T,delta_x) :
           t = beta*t
       t_history.append(t)
       x = x + t*delta_x
```

The plot results are:



(a) step length versus iteration number (b) f - p* versus iteration number

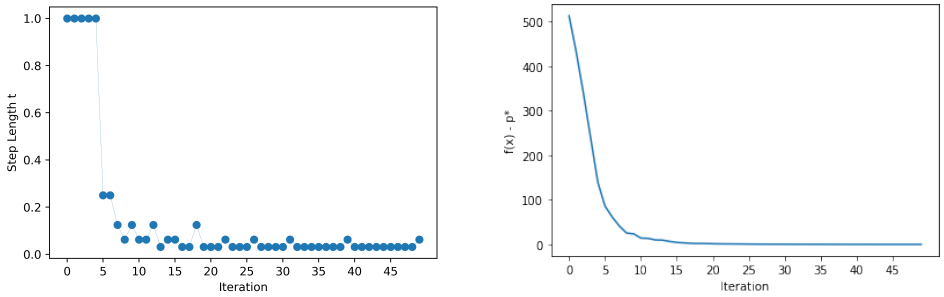
Figure 1: Result of gradient method

(b) For Newton's method, also set $x = \mathbf{0}$ as the feasible starting point. At each iteration, first calculate the gradient and hessian of the objective function. It is easier to compute in Python using the matrix form instead of the decomposed form. Second, determine whether the stop criterion is met based on the Newton decrement λ^2. Then calculate the descent direction Δx, and check the feasibility of the new point. Finally use backtracking line search to determine step length.

The codes are:

```
[25] eta = 1e-8
     x_history = []
     f_history = []
     t_history = []
     x = np.zeros((n,1))
     for _ in range(50):
       x_history.append(x)
       f_history.append(f(x))
       gradient = A.T @ (1/(1-A@x)) - 1/(1+x) + 1/(1-x)

       hessian = A.T @ np.diag((1/((1-A@x)**2)).T.tolist()[0]) @A
       + np.diag(np.diag(1/(1+x)@(1+x).T +1/(1-x)@(1-x).T))

       delta_x = - np.linalg.inv(hessian) @ gradient

       if abs(gradient.T@delta_x) <= eta:
         break

       t = 1
       while max(np.matmul(A,x + t*delta_x)) >= 1 or max(abs(x + t*delta_x)) >=1:
         t = beta*t

       while f(x+t*delta_x) >= f(x) + alpha*t*gradient.T @ delta_x and t>=1e-20:
           t = beta*t
       t_history.append(t)
       x = x + t*delta_x
```

The plot results are:



(a) step length versus iteration number (b) f - p* versus iteration number

Figure 2: Result of Newton's method

In this experiment, we can see that Newton's method convergence much quicker than gradient method.

10.2.

(a) The equation can be expressed as:

$$v + A^T w = -\nabla f(x)$$
$$Av = 0$$

Δx_{pg} is the solution of the following problem:

$$\begin{array}{ll} \text{minimize} & \| - \nabla f(x) - u\|_2 \\ \text{subject to:} & Au = 0 \end{array}$$

One of the KKT condition of this equality constraint is:

$$Au^* = 0$$

So we have,

$$\Delta x_{pg} = u^* = v$$

And therefore we can imply:

$$A^T w + \nabla f(x) = -\underset{Au=0}{\text{argmin}} \| - \nabla f(x) - u\|_2$$
$$w = \underset{y}{\text{argmin}} \|\nabla f(x) + A^T y\|_2$$

(b) The reduced problem (10.5) is:

$$\text{minimize} \qquad \tilde{f}(z) = f(Fz + \hat{x})$$

And the negative gradient of the reduced problem is:

$$-\nabla \tilde{f}(z) = -F \nabla f(Fz + \hat{x})$$

Since $F^TF = I$,

$$-F\nabla\tilde{f}(z) = -\nabla f(Fz + \hat{x}) = -\nabla f(x)$$

Also, we can imply that,

$$\begin{aligned} Ax = A(Fz + \hat{x}) = 0 \\ AFz = 0 \end{aligned}$$

Therefore,

$$\begin{aligned} \Delta x_{pg} &= \operatorname{argmin} \| - \nabla f(x) - Fz\|_2 \\ &= \operatorname{argmin} \| - F\nabla\tilde{f}(z) - Fz\|_2 \\ &= -F\nabla\tilde{f}(Fz + \hat{x}) \\ &= -F\nabla\tilde{f}(x) \end{aligned}$$

(c) From (b) we know $\Delta x_{pg} = -F\nabla\tilde{f}(x)$, which means if we start from $x^{(0)}$ with $Ax^{(0)} = b$, the affects are the same. We have analyzed that applying Newton's method to the reduced problem converges when the initial sublevel set $\{x|x \in \operatorname{dom} f, f(x) \leq f(x^{(0)}), Ax = b\}$ is closed, and the objective function $f(x)$ is strongly convex.

10.13.

(a) We can write the Hessian in the form as:

$$\nabla^2 f(u,v) = \begin{bmatrix} \nabla^2_{uu} f(u,v) & \nabla^2_{vu} f(u,v) \\ \nabla^2_{uv} f(u,v) & \nabla^2_{vv} f(u,v) \end{bmatrix} = \begin{bmatrix} A & B \\ B^T & -C \end{bmatrix}$$

where $A = \nabla^2_{uu} f(u,v) \succeq mI$, $C = -\nabla^2_{vv} f(u,v) \succeq mI$.

Since the Newton step Δx_{nt} is characterized by:

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{nt} \\ w \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix}$$

To find Δx_{nt}, we have to solve:

$$\nabla^2 f(x)\Delta x_{nt} = -\nabla f(x) - A^T w$$

This equation can be written in the form of:

$$\begin{bmatrix} A & B \\ B^T & -C \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = -\begin{bmatrix} g \\ h \end{bmatrix}$$

Or,

$$\begin{aligned} Av + Bw = -g \\ B^T v - Cw = -h \end{aligned}$$

Derive v from the first equation and put back into the second one, we have:

$$v = -A^{-1}(g + Bw)$$
$$w = (B^T A^{-1} B + C)^{-1}(h - B^T A^{-1} g)$$

Since A and C are positive semidefinite, we could use Cholesky factorization to reduce computation.

The computations are:

- Compute Cholesky factorize $A = LL^T$: $\approx \frac{1}{3}p^3$ flops
- Compute $Y = L^{-1}B$, $Z = L^{-1}g$: $p^2 q$ and $2p^2$ flops
- Compute $S = (Y^T Y + C)$: pq^2 flops
- Compute $T = h - Y^T Z$: $2pq$ flpos
- Compute $Sw = T$ using Cholesky decomposition: $\frac{1}{3}q^3$ flops

To sum up, compute Newton step using Cholesky factorizations cost:

$$\frac{1}{3}p^3 + p^2 q + 2p^2 + pq^2 + 2pq + \frac{1}{3}q^3 = O\left(\frac{1}{3}(p+q)^3\right)$$

If we use LDL^T factorization of $\nabla^2 f(u, v)$,

$$\nabla^2 f(u, v) = PLDL^T P^T$$

We assume $\nabla^2 f(u, v)$ is dense, so the factorize costs $O(\frac{1}{3}p^3)$ flops, which is the same as the cost of Cholesky factorization. Since the other computation steps are also the same, the cost of Cholesky factorization and LDL^T factorization are the same.

(b) If $A = \nabla^2_{uu} f(u, v)$ is diagonal, no need to factorize it. If $A = \nabla^2_{uu} f(u, v)$ is block diagonal, factoring it is immediate, name the cost f.

The computations are:

- Compute $Z = A^{-1}g$: $p^2(s/2)$ flops
- Compute $S = (Y^T Y + C)$: pq^2 flops
- Compute $T = h - Y^T Z$: $2pq$ flpos
- Compute $Sw = T$ using Cholesky decomposition: $\frac{1}{3}q^3$ flops

So the total cost is:

$$\left(\frac{1}{2}p^2 s + pq^2 + 2pq + \frac{1}{3}q^3\right)$$

10.15. The parameters setting are:

```
n = 100
p = 30
eta = 1e-8
alpha = 0.01
beta = 0.5
```

```
A = np.random.normal(5, 10, size=(p, n))
while np.linalg.matrix_rank(A) != p:
  A = np.random.normal(1, 10, size=(p, n))
x_0 = np.array([random.uniform(0,1) for i in range(n)])[np.newaxis].T
```

```
def f(x):
  return float(x.T @ np.log(x))
```

(a) Standard Newton method:

The codes are:

```
x = x_0
b = A @ x
x_history = []
w_history = []
f_history = []
t_history = []
for _ in range(100):
  x_history.append(x)
  f_history.append(f(x))
  gradient = 1 + np.log(x)
  hessian = np.diag((1/x).flatten())

  x_w = np.linalg.inv(np.block([[hessian, A.T],
    [A, np.zeros((p,p))]])) @ np.block([[-gradient],
    [np.zeros((p,1))]])
  delta_x = x_w[:n]
  w = x_w[n:]
  w_history.append(w)

  if abs(gradient.T @ delta_x) <= eta:
    print("-----converge-----")
    break

  t = 1
  while min(x + t*delta_x) <= 0:
    t = beta*t

  while f(x+t*delta_x) >= f(x) + alpha*t*gradient.T @ delta_x :
      t = beta*t
  t_history.append(t)
  x = x + t*delta_x
```

The solution is: -33.074353372849465.

(b) Infeasible start Newton method:

First we want to start with the same initial point and compare with the standard Newton method.

The codes are:

```
x = x_0
b = A @ x
v = np.zeros((p,1))

x_history = []
v_history = []
f_history = []
t_history = []
for _ in range(100):
  x_history.append(x)
  v_history.append(v)
  f_history.append(f(x))
  gradient = 1 + np.log(x)
  hessian = np.diag((1/x).flatten())

  r = np.block([[gradient+A.T @ v], [A@x-b]])

  delta_x_v = - np.linalg.inv(np.block([[hessian, A.T],
    [A, np.zeros((p,p))]])) @ r
  delta_x = delta_x_v[:n]
  delta_v = delta_x_v[n:]

  if np.linalg.norm(r) <= eta:
    print("-----converge-----")
    break

  t = 1
  while min(x + t*delta_x) <= 0:
    t = beta*t


  new_r = np.block([[1+np.log(x+t*delta_x)+A.T@(v+t*delta_v)],
                                  [A@(x+t*delta_x)-b]])
  while np.linalg.norm(new_r) >= (1-alpha*t)*np.linalg.norm(r) and t>
                                        =1e-30:
      t = beta*t
  t_history.append(t)
  x = x + t*delta_x
  v = v + t*delta_v
```

The solution is: -33.074353372849465, which is exactly the same as in (a).

Then we started with an infeasible point $x^{(0)} = 1$ and verified our method works.

(c) The dual problem is:

$$\max -b^\top v - \sum_{i=1}^{n} e^{-a_i^\top v - 1}$$

This is an unconstrained problem, so we can apply vanilla Newton method.

The codes are:

```
x = x_0
```

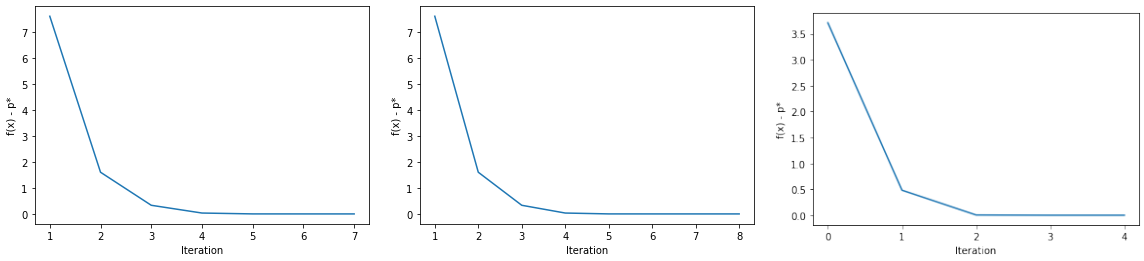
```
b = A @ x
v = np.zeros((p,1))

x_history = []
v_history = []
f_history = []
t_history = []
for _ in range(100):
  x_history.append(x)
  v_history.append(v)
  f = float(b.T @ v + sum(np.exp(-A.T@v-1)))
  f_history.append(f)
  gradient = b - A@np.exp(-A.T@v-1)
  hessian = A @ np.diag(np.exp(-A.T@v-1).flatten()) @ A.T

  delta_v = - np.linalg.inv(hessian) @ gradient

  if abs(gradient.T @ delta_v) <= eta:
    print("-----converge-----")
    break

  t = 1
  while b.T@(v+t*delta_v)+sum(np.exp(-A.T@(v+t*delta_v)-1)) > f+alpha
                                        *t*gradient.T@delta_v:
    t = beta*t
  v = v + t*delta_v
```

The solution is: -33.074353372849465, which is exactly the same as the above two. Also the Lagrange multiplier of the three methods are exactly the same, which is:

```
array([[-0.00609544],
       [ 0.00110149],
       [-0.00240211],
       [-0.00577771],
       [ 0.00742805],
       [ 0.01033087],
       [-0.00450386],
       [ 0.00769715],
       [ 0.00226439],
       [ 0.00296587],
       [ 0.00612498],
       [-0.00691957],
       [-0.00455807],
       [ 0.00769236],
       [-0.01103064],
       [-0.00868194],
       [-0.00138361],
       [-0.00046549],
       [-0.00871759],
```

```
[-0.00399775],
[-0.00287929],
[-0.00200232],
[ 0.00774598],
[-0.00162215],
[-0.00538928],
[-0.01287795],
[ 0.00749998],
[-0.00084423],
[-0.00348177],
[-0.00774678]])
```

We also want to compare the computational effort per step for the three methods,



(a) Standard Newton method (b) Infeasible start Newton (c) Dual Newton method

Figure 3: computational effort per step for the three methods

We can see solving the dual problem has a faster convergence speed.

11.7.

(a) Centrality equations (11.7) is:

$$0 = t\nabla f_0\left(x^\star(t)\right) + \nabla\phi\left(x^\star(t)\right) + A^T\hat{\nu}$$

Differentiate it with respect to t,

$$\nabla f_0\left(x^\star(t)\right) + t\nabla^2 f_0\left(x^\star(t)\right)\frac{dx^*}{dt} + \nabla^2\phi\left(x^\star(t)\right)\frac{dx^*}{dt} = 0$$
$$\frac{dx^*}{dt} = -(t\nabla^2 f_0\left(x^\star(t)\right) + \nabla^2\phi\left(x^\star(t)\right))^{-1}\nabla f_0\left(x^\star(t)\right)$$

(b) We want to know the correlation between $f_0(x^*(t))$ and t,

$$\begin{aligned}\frac{\partial}{\partial t}f_0\left(x^\star(t)\right) &= \nabla f_0\left(x^\star(t)\right)^T\frac{dx^*}{dt} \\ &= -\nabla f_0\left(x^\star(t)\right)^T\left(t\nabla^2 f_0\left(x^\star(t)\right) + \nabla^2\phi\left(x^\star(t)\right)\right)^{-1}\nabla f_0\left(x^\star(t)\right)\end{aligned}$$

Since $\nabla^2 f_0\left(x^\star(t)\right)$ and $\nabla^2\phi\left(x^\star(t)\right)$ are positive definite and $t > 0$,

$$t\nabla^2 f_0\left(x^\star(t)\right) + \nabla^2\phi\left(x^\star(t)\right)$$

is positive definite. So,

$$\frac{\partial}{\partial t} f_0\left(x^\star(t)\right) < 0$$

$f_0(x^*(t))$ decreases as t increases. Thus, the objective value in the barrier method decreases, as the parameter t is increased.

11.20.

(a) The primal problem is:

$$\begin{array}{ll} \text{minimize} & -U(x) \\ \text{subject to:} & Ax \preceq c \\ & x \succeq 0 \end{array}$$

Lagrangian is:

$$\begin{aligned} L(x, \lambda_1, \lambda_2) &= -U(x) + \lambda_1^T(Ax - c) + \lambda_2^T(-x) \\ &= -U(x) + (A^T\lambda_1 - \lambda_2)^T x - c^T\lambda_1 \end{aligned}$$

Since $U(x)$ is concave and nondecreasing, $-U(x)$ is convex and decreasing. Also, $(A^T\lambda_1 - \lambda_2)^T x - c^T\lambda_1$ is affine in x. So Lagrangian is convex and we could minimize it over x.

The dual function is:

$$\begin{aligned} g(\lambda_1, \lambda_2) &= \inf_x L(x, \lambda_1, \lambda_2) \\ &= \inf_x(-U(x) + (A^T\lambda_1 - \lambda_2)^T x - c^T\lambda_1) \\ &= \inf_x \sum_{i=1}^n(-U_i(x_i) + (A^T\lambda_1)_i x_i - \lambda_{2i}x_i) - c^T\lambda_1 \\ &= -\sum_{i=1}^n \sup_x \left(U_i(x_i) - (A^T\lambda_1)_i x_i + \lambda_{2i}x_i\right) - c^T\lambda_1 \end{aligned}$$

Define the conjugate utility functions as,

$$V_i(\lambda) = \sup_{x>0}\left(\lambda x + U_i(x)\right)$$

If $\lambda \geq 0$, since $U_i(x)$ is nondecreasing, $V_i(\lambda) \to \infty$ when $x \to \infty$.

So $\mathbf{dom} V_i \in -R_{++}$, and $\nabla V_i \to \infty$ as $x \to 0$, $\nabla V_i \to 0$ as $x \to \infty$. So that for each $\lambda < 0$ there is a unique x with $U_i^{'}(x) = -\lambda$.

So we can write the dual function as:

$$g(\lambda_1, \lambda_2) = -\sum_{i=1}^{n} V_i(-(A^T\lambda_1)_i + \lambda_{2i}) - c^T\lambda_1$$

The dual problem is:

$$\begin{array}{ll} \text{maximize} & -\sum_{i=1}^{n} V_i(-(A^T\lambda_1)_i + \lambda_{2i}) - c^T\lambda_1 \\ \text{subject to:} & \lambda_1 \succeq 0 \\ & \lambda_2 \succeq 0 \end{array}$$

Or we can flip the objective function:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} V_i(-(A^T\lambda_1)_i + \lambda_{2i}) + c^T\lambda_1 \\ \text{subject to:} & \lambda_1 \succeq 0 \\ & \lambda_2 \succeq 0 \end{array}$$

Since the $V_i(-(A^T\lambda_1)_i + \lambda_{2i})$ has minimum at $\lambda_2 = 0$, we could further simplify the problem as:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} V_i(-(A^T\lambda)_i) + c^T\lambda \\ \text{subject to:} & \lambda \succeq 0 \end{array}$$

(b) We would love to implement barrier method for the dual problem. The logarithmic barrier function is:

$$\Phi(x) = -\sum_{i=1}^{m} \log(\lambda_i)$$

And the objective is:

$$\text{minimize} \;\; t\left(\sum_{i=1}^{n} V_i(-(A^T\lambda)_i) + c^T\lambda\right) - \sum_{i=1}^{m} \log(\lambda_i)$$

Its hessian is:

$$H = tA\,\text{diag}\left(\left(-A^T\lambda\right)\nabla^2 V_i\right)^{-2} A^T + \text{diag}(\lambda)^{-2}$$

If AA^T is sparse, then $tA\,\text{diag}\left(\left(-A^T\lambda\right)\nabla^2 V_i\right)^{-2} A^T$ is sparse. For Newton's method, we solve,

$$H\Delta\lambda = -g$$

So a sparse matrix method can be used to compute the Newton step.

If A^TA is sparse, we want to solve,

$$(D_1 + A^T D_2 A)x = b$$

where D_1 and D_2 are diagonal. We could use the matrix inversion lemma to simplify the above equation and compute the Newton step.

11.21.

(a) The fractional Chebyshev approximation problem is:

$$\begin{array}{ll} \text{minimize} & \max_{i=1,\ldots,n} \mid \log a_i^T x - \log b_i \mid \\ \text{subject to:} & l \preceq x \preceq u \\ & Ax \succ 0 \end{array}$$

Since $log(x)$ is strictly increasing, we could simplify the objective function as:

$$\max_{i=1,\ldots,n} \mid a_i^T x - b_i \mid$$

We can formulate the above problem to be a convex problem:

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to:} & (a_i^T x)/b_i - s \leq 0 \\ & b_i/(a_i^T x) - s \leq 0 \\ & l \preceq x \preceq u \\ & Ax \succ 0 \end{array}$$

We could see that the objective and the constraint functions are all convex and twice differentiable.

(b) Without loss of generality, we could assume $b_i = 1$, so the problem becomes:

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to:} & (a_i^T x) - s \leq 0 \\ & 1/(a_i^T x) - s \leq 0 \\ & l \preceq x \preceq u \\ & Ax \succ 0 \end{array}$$

The logarithmic barrier function is:

$$\begin{aligned} \Phi(x) &= -\sum_{i=1}^{m} \{\log(-(a_i^T x)/ + s) + \log(-1/(a_i^T x) + s) \\ &\quad + \log(x_i - l_i) + \log(-x_i + u_i) + \log(a_i^T x)\} \\ &= -\sum_{i=1}^{m} \{\log(-(a_i^T x) + s) + \log(s(a_i^T x) - 1) \\ &\quad - \log(a_i^T x) + \log(x_i - l_i) + \log(-x_i + u_i) + \log(a_i^T x)\} \\ &= -\sum_{i=1}^{m} \{\log(s - (a_i^T x)) + \log(s(a_i^T x) - 1) + \log(x_i - l_i) + \log(-x_i + u_i)\} \end{aligned}$$

And the objective is :

$$f(s, x) = ts - \sum_{i=1}^{m} \{\log(s - (a_i^T x)) + \log(s(a_i^T x) - 1) + \log(x_i - l_i) + \log(-x_i + u_i)\}$$

The barrier method is:

---

feasible$x = x^{(0)}, t^{(0)}, \mu, \epsilon$

repeat

1. Centering step.

   repeat

   1. Compute the Newton step and decrement.

      $\Delta x_{nt} := -\nabla^2 f(x)^{-1}\nabla f(x); \quad \lambda^2 := \nabla f(x)^T \nabla^2 f(x)^{-1}\nabla f(x)$

      2. Stopping criterion. quit if $\lambda^2/2 \leq \epsilon$

      3. Line search. Choose step size t_{nt} by backtracking line search.

      4. Update. $x := x + t_{nt}\Delta x_{\text{nt}}$

   return $x^*(t)$

2. Update. $x := x^*(t)$

3. Stopping criterion. quit if $m/t < \epsilon$

4. Increase $t.t := \mu t$

Now calculate the first and second order derivative of $f(x, s)$. The gradient is:

$$\nabla f(s,x) = \begin{bmatrix} t \\ \operatorname{diag}(u-x)^{-1}\mathbf{1} - \operatorname{diag}(x-l)^{-1}\mathbf{1} \end{bmatrix} + \begin{bmatrix} -\mathbf{1}^T \\ A^T \end{bmatrix} \operatorname{diag}(s-Ax)^{-1}\mathbf{1} + \left(- \begin{bmatrix} (Ax)^T \\ sA^T \end{bmatrix} \operatorname{diag}(sAx-1)^{-1}\mathbf{1} \right)$$

The hessian is:

$$\nabla^2 f(s,x) = \begin{bmatrix} 0 & 0 \\ 0 & \operatorname{diag}(u-x)^{-2} + \operatorname{diag}(x-l)^{-2} \end{bmatrix} + \begin{bmatrix} -\mathbf{1}^T \\ A^T \end{bmatrix} \operatorname{diag}(s-Ax)^{-2} \begin{bmatrix} -\mathbf{1} & A \end{bmatrix} + \begin{bmatrix} x^T A \operatorname{diag}(sAx-\mathbf{1})^{-2}Ax & \mathbf{1}^T \operatorname{diag}(sAx-1)^{-2}A \\ A^T \operatorname{diag}(sAx-\mathbf{1})^{-2}\mathbf{1} & s^2 A^T \operatorname{diag}(sAx-\mathbf{1})^{-2}A \end{bmatrix}$$

The codes are:

```
m = 80
n = 50
alpha = 0.01
beta = 0.5
mu = 20
NTTOL = 1e-8
TOL = 1e-4
```

```
A = np.random.rand(m,n)
l = np.random.rand(n,1)
x = l + np.random.rand(n,1)
x_0 = x
u = x + np.random.rand(n,1)
```

```
b = A @ x
s = float(1.1*max(np.block([[max(b)], [max(1/b)]])))
t = 1
```

```
def f(x,t=t,s=s,l=l,u=u,b=b):
  return float(t*s - sum(np.log(u-x)) - sum(np.log(x-l)) - sum(np.log
                                          (s-b)) - sum(np.log(s*b-1)))
```

```
def newton_method(t, s, x, b, A=A, l=l, u=u):
  for k in range(500):
    obj = f(x,t)
    gradient = np.block([[t-sum(1./(s-b))-sum(b/(s*b-1))],
    [1/(u-x)-1/(x-l)+A.T@(1/(s-b)-s/(s*b-1))]])
    hessian = np.block([[sum((s-b)**(-2)+(b/(s*b-1))**(-2)),
                          (-(s-b)**(-2) + (s*b-1)**(-2)).T@A],
                        [A.T@(-(np.ones((m,1))*s-b)**(-2)
                        + (s*b-1)**(-2)),
                        np.diag((u-x).flatten()**(-2)+
                        (x-l).flatten()**(-2))+
                        A.T@(np.diag((s-b).flatten()**(-2)+
                        (s/(s*b-1)).flatten()**(-2)))@A]])
    delta_t_s = - np.linalg.inv(hessian) @ gradient
    if abs(gradient.T @ delta_t_s)  <= NTTOL:
      print("-----Newton method converge-----")
      return(s,x,b)
    delta_s = delta_t_s[0]
    delta_x = delta_t_s[1:]
    delta_b = A@delta_x

    t_newton = 1
    s_new = s+t_newton*delta_s
    x_new = x+t_newton*delta_x
    b_new = b+t_newton*delta_b
    while min(np.block([[np.ones((m,1))*s_new-b_new],
                        [np.ones((m,1))*s_new -1/b_new],
                        [b_new],
                        [u-x_new],
                        [x_new - l]])) <= 0 and t_newton > 1e-20:
      t_newton = beta * t_newton

    s_new = s+t_newton*delta_s
    x_new = x+t_newton*delta_x
    b_new = b+t_newton*delta_b

    obj_new = f(x_new,t)
    while obj_new  >= obj+ t_newton*alpha*gradient.T @ delta_t_s and
                                          t_newton > 1e-20:
      t_newton = beta * t_newton


    s = s+t_newton*delta_s
    x = x+t_newton*delta_x
    b = A@x
```

```
  return(s,x,b)
```

```
def barrier_method(t=t, x=x, A=A, l=l, u=u, b=b, s=s):
  for _ in range(100):
    if (3*m+2*n)/t < TOL:
      print("-----barrier method converge-----")
      return x
    t = mu * t
    s,x,b = newton_method(t,s,x,b)
  return x
```

```
    x = barrier_method()
```

Bonus questions:
9.18.

(a) From 9.17(c), we know that,

$$(1-t\alpha)^2\nabla^2 f(x)v \preceq \nabla^2 f(x+tv) \preceq \frac{1}{(1-t\alpha)^2}\nabla^2 f(x)$$

for $x+tv \in \operatorname{dom} f$, $0 \leq t < \alpha$, $\alpha = (v^T\nabla^2 f(x)v)^{1/2}$, v is the descent direction $-\nabla^2 f(x)^{-1}\nabla f(x)$.

For strictly convex self-concordant functions, we can obtain similar bounds in terms of the Newton decrement,

$$\lambda(x) = (\nabla f(x)^T\nabla^2 f(x)^{-1}\nabla f(x))^{1/2} = \alpha$$

So we can write,

$$(1-t\lambda(x))^2\nabla^2 f(x) \preceq \nabla^2 f(x-t\nabla^2 f(x)^{-1}\nabla f(x)) \preceq \frac{1}{(1-t\lambda(x))^2}\nabla^2 f(x)$$

Since $\lambda(x) < 1$, we could let $t = 1$ and define $x^+ = x - \nabla^2 f(x)^{-1}\nabla f(x)$

$$(1-\lambda(x))^2\nabla^2 f(x) \preceq \nabla^2 f(x-\nabla^2 f(x)^{-1}\nabla f(x)) \preceq \frac{1}{(1-\lambda(x))^2}\nabla^2 f(x)$$

$$(1-\lambda(x))^2\nabla^2 f(x) \preceq \nabla^2 f(x^+) \preceq \frac{1}{(1-\lambda(x))^2}\nabla^2 f(x)$$

We know that,

$$\lambda(x^+) = (\nabla f(x^+)^T\nabla^2 f(x^+)^{-1}\nabla f(x^+))^{1/2}$$

So we now have,

$$\begin{aligned}\lambda(x^+) &\leq (\nabla f(x^+)^T \frac{\nabla^2 f(x)}{(1-\lambda(x))^2}\nabla f(x^+))^{1/2} \\ &= \frac{1}{1-\lambda(x)}(\nabla f(x^+)^T\nabla^2 f(x)\nabla f(x^+))^{1/2}\end{aligned}$$

Without loss of generality, we could assume that $\nabla^2 f(x) = I$, so

$$\begin{aligned}\lambda(x) &= (\nabla f(x)^T \nabla^2 f(x) \nabla f(x))^{1/2} \\ &\geq (\nabla f(x^+)^T \nabla^2 f(x) \nabla f(x^+))^{1/2}\end{aligned}$$

So we have,

$$\lambda(x^+) \leq \frac{\lambda(x)}{1-\lambda(x)} \preceq 1$$

$$\lambda(x^+) \leq \frac{\lambda(x)^2}{(1-\lambda(x))^2}$$