CS257 Linear and Convex Optimization Homework 7

Xiangyuan Xue (521030910387)

1. Implementation of pure Newton's method is as follow:

```
def newton(fp, fpp, x0, tol=1e-5, maxiter=100000):
    x_traces = [np.array(x0)]
    x = np.array(x0)
    for _ in range(maxiter):
        grad = fp(x)
        if np.linalg.norm(grad) < tol:
            break
        x = x - np.linalg.solve(fpp(x), grad)
        x_traces.append(np.array(x))
    return x_traces
```

Implementation of damped Newton's method is as follow:

```
def damped_newton(f, fp, fpp, x0, alpha=0.5, beta=0.5, tol=1e-5,
     maxiter=100000):
    x_traces = [np.array(x0)]
    stepsize_traces = []
    tot_num_iter = 0
    x = np.array(x0)
    for _ in range(maxiter):
        grad = fp(x)
        if np.linalg.norm(grad) < tol:
            break
        direction = np.linalg.solve(fpp(x), grad)
        stepsize = 1.0
        while f(x - stepsize * direction > f(x) + alpha *
    stepsize * (grad @ direction)):
            stepsize = beta * stepsize
        x = x - stepsize * direction
        x_traces.append(np.array(x))
        stepsize_traces.append(stepsize)
    return x_traces, stepsize_traces, tot_num_iter
```

(a) Since:

$$\nabla f(\boldsymbol{w}) = -\sum_{i=1}^{m} \left[1 - \sigma(y_i \boldsymbol{x}_i^T \boldsymbol{w})\right] y_i \boldsymbol{x}_i$$

For the j-th the component of $\nabla f(\boldsymbol{w})$:

$$\begin{aligned}\frac{\partial \left[\nabla f(\boldsymbol{w})\right]_j}{\partial w_k} &= \frac{\partial}{\partial w_k}\left\{-\sum_{i=1}^{m} \left[1 - \sigma(y_i \boldsymbol{x}_i^T \boldsymbol{w})\right] y_i x_{ij}\right\} \\ &= \sum_{i=1}^{m} \left[\sigma'(y_i \boldsymbol{x}_i^T \boldsymbol{w})\right] y_i^2 x_{ij} x_{ik} \\ &= \sum_{i=1}^{m} \sigma'(y_i \boldsymbol{x}_i^T \boldsymbol{w}) x_{ij} x_{ik}\end{aligned}$$

Thus, the Hessian of f is:

$$\nabla^2 f(\boldsymbol{w}) = \sum_{i=1}^{m} \sigma'(y_i \boldsymbol{x}_i^T \boldsymbol{w}) \boldsymbol{x}_i \boldsymbol{x}_i^T$$

Gradient function is implemented as follow:

```
def fp(w):
    return (-1 * (Xy.T @ (1 - sigmoid(Xy @ w)).reshape((-1,
   1)))).reshape(-1)
```

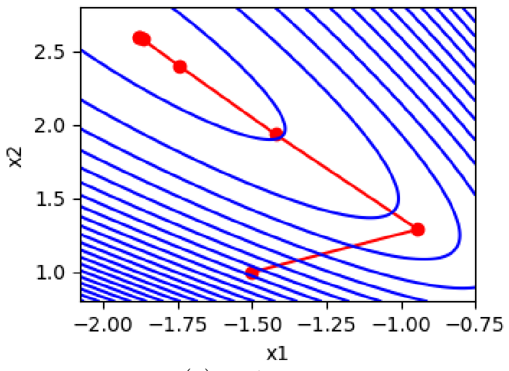
Hessian function is implemented as follow:

```
def fpp(w):
    P = sigmoid_p(Xy @ w).reshape((-1, 1)) * X
    return P.T @ X
```

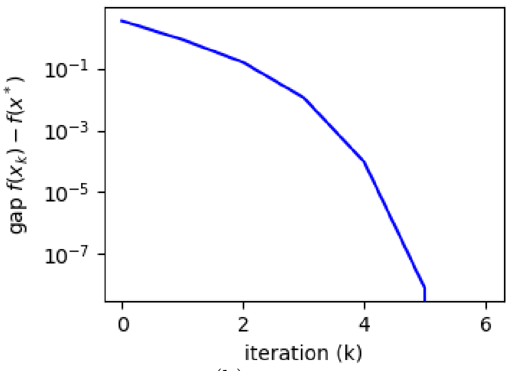
(b) When $\boldsymbol{w}_0 = (-1.5, 1)^T$, the algorithm converges. The result is as follow:

```
pure Newton's method with initial point [-1.5  1. ]
    state: converge
    iterations: 6
    solution: [-1.87973941  2.60188452]
    value: 3.3295135687527972
```

Trajectory of $\boldsymbol{x}_k$ and the gap $f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*)$ are shown in following figures:



(a) trajectory



(b) gap

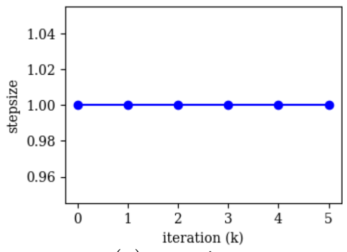
Figure 1: trajectory and gap for pure Newton's method with $\boldsymbol{w}_0 = (-1.5, 1)^T$

When $\boldsymbol{w}_0 = (1, 1)^T$, the algorithm diverges.

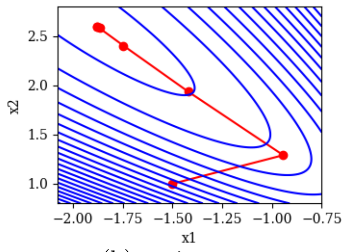
(c) When $\boldsymbol{w}_0 = (-1.5, 1)^T$, the algorithm converges. The result is as follow:

```
damped Newton's method with initial point [-1.5  1. ]
    state: converge
    iterations (outer loop): 6
    iterations (inner loop): 0
    solution: [-1.87973941  2.60188452]
    value: 3.3295135687527972
```

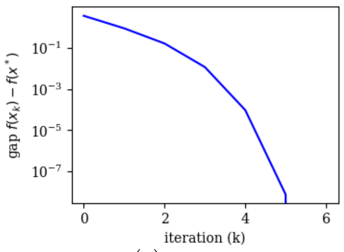
Stepsize, trajectory of $\boldsymbol{x}_k$ and the gap $f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*)$ are shown in following figures:



(a) stepsize



(b) trajectory



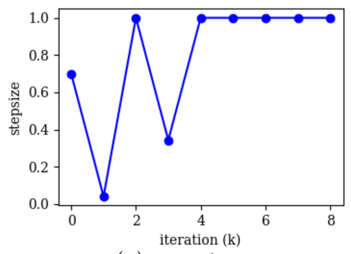
(c) gap

Figure 2: stepsize, trajectory and gap for damped Newton's method with $\boldsymbol{w}_0 = (-1.5, 1)^T$

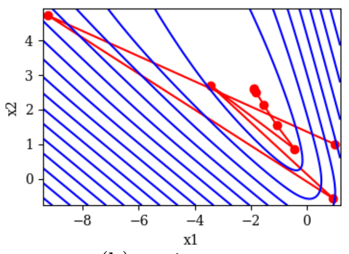
When $\boldsymbol{w}_0 = (1, 1)^T$, the algorithm converges. The result is as follow:

```
damped Newton's method with initial point [1. 1.]
    state: converge
    iterations (outer loop): 9
    iterations (inner loop): 13
    solution: [-1.87972845  2.60186892]
    value: 3.3295135688224926
```

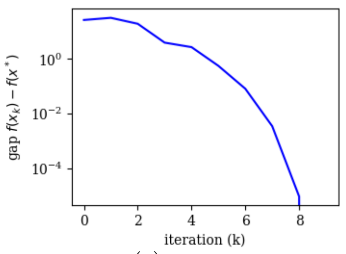
Stepsize, trajectory of $\boldsymbol{x}_k$ and the gap $f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*)$ are shown in following figures:



(a) stepsize



(b) trajectory



(c) gap

Figure 3: stepsize, trajectory and gap for damped Newton's method with $\boldsymbol{w}_0 = (1, 1)^T$

For pure Newton's method, the convergence depends on the initial point, but damped Newton's method converges regardless of the initial point. For converging case, damped Newton's method has the same behavior with pure Newton's method.

2. (a) Since $f(x) = (x-a)^6$, we have:

$$f'(x) = 6(x-a)^5, f''(x) = 30(x-a)^4$$

Then explicit expression for the Newton step is:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} = x_k - \frac{1}{5}(x_k - a) = \frac{4}{5}x_k + \frac{1}{5}a$$

(b) Since:

$$x_{k+1} = x_k - \frac{1}{5}(x_k - a)$$

We have:

$$x_{k+1} - a = (x_k - a) - \frac{1}{5}(x_k - a) = \frac{4}{5}(x_k - a)$$

Therefore:

$$y_{k+1} = |x_{k+1} - a| = \frac{4}{5}|x_k - a| = \frac{4}{5}y_k$$

(c) From the relationship above we have:

$$|x_k - a| = \left(\frac{4}{5}\right)^k |x_0 - a|$$

For any $\varepsilon > 0$, to make:

$$|x_k - a| < \varepsilon$$

We require:

$$k > \frac{\log \frac{1}{\varepsilon} + \log |x_0 - a|}{\log \frac{5}{4}}$$

Thus, we claim that:

$$k = O\left(\log \frac{1}{\varepsilon}\right)$$

Therefore, we conclude that $|x_k - a|$ decays to zero exponentially.

3. Implementation of soft-thresholding operator is as follow:

```
def soft_th(w, th):
    return np.sign(w) * np.maximum(np.abs(w) - th, 0)
```

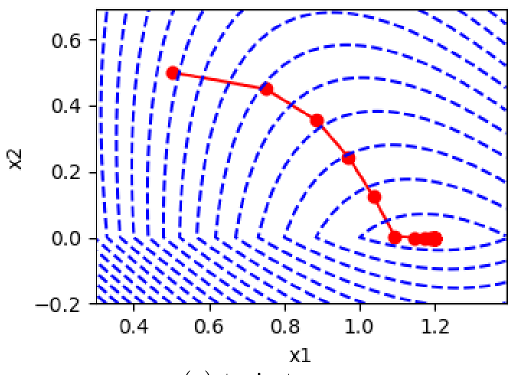
Implementation of ISTA algorithm is as follow:

```
def ista(X, y, lambd, w0, stepsize, tol=1e-9, maxiter=100000):
    w_traces = [np.array(w0)]
    w = np.array(w0)
    for _ in range(maxiter):
        grad = X.T @ (X @ w - y)
        w_next = soft_th(w - stepsize * grad, lambd * stepsize)
        if np.linalg.norm(w_next - w) < tol:
            break
        w = w_next
        w_traces.append(np.array(w))
    return w_traces
```

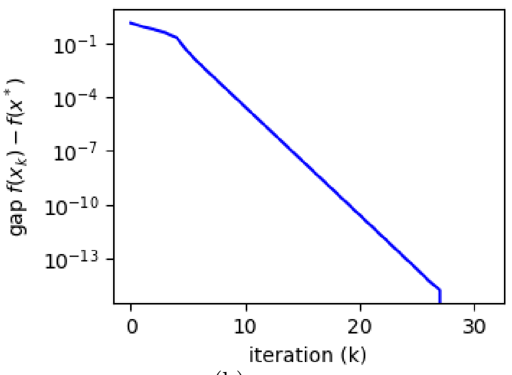
(a) The result is as follow:

```
ISTA algorithm with lambda = 2.0
    state: converge
    iterations: 31
    solution: [1.2 0. ]
    value: 4.9
```

Ignoring numerical errors, the solution is $\boldsymbol{w} = (1.2, 0)^T$, which contains a zero. Trajectory of $\boldsymbol{w}_k$ and the gap $f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*)$ are shown in following figures:



(a) trajectory



(b) gap

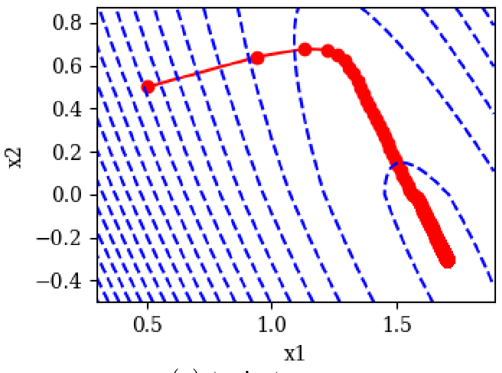
Figure 4: trajectory and gap for ISTA algorithm with $\lambda = 2$

(b) The result is as follow:

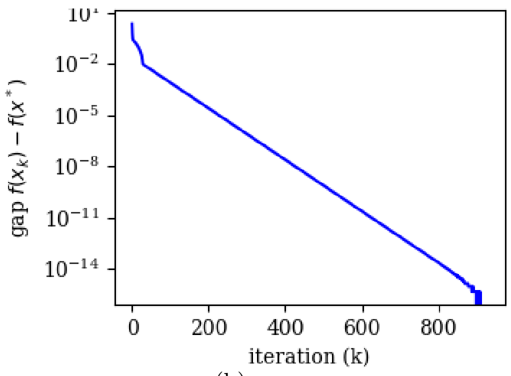
```
ISTA algorithm with lambda = 0.1
    state: converge
```

```
        iterations: 927
        solution: [ 1.69999998 -0.29999995]
        value: 2.2500000000000004
```

Ignoring numerical errors, the solution is $\boldsymbol{w} = (1.7, -0.3)^T$, which contains no zeros. Trajectory of $\boldsymbol{w}_k$ and the gap $f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*)$ are shown in following figures:



(a) trajectory



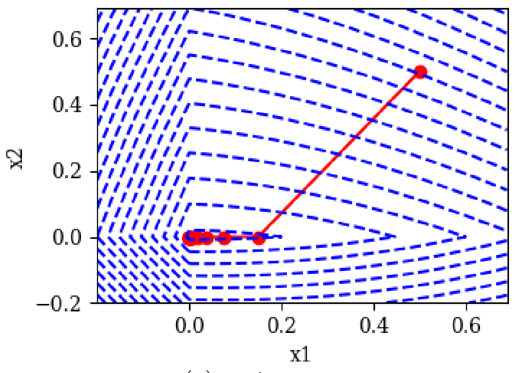
(b) gap

Figure 5: trajectory and gap for ISTA algorithm with $\lambda = 0.1$

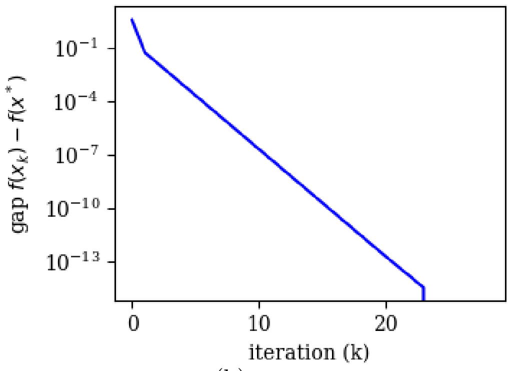
(c) The result is as follow:

```
    ISTA algorithm with lambda = 8.0
        state: converge
        iterations: 28
        solution: [1.11758702e-09 0.00000000e+00]
        value: 8.5
```

Ignoring numerical errors, the solution is $\boldsymbol{w} = (0, 0)^T$, which contains two zeros. Trajectory of $\boldsymbol{w}_k$ and the gap $f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*)$ are shown in following figures:



(a) trajectory



(b) gap

Figure 6: trajectory and gap for ISTA algorithm with $\lambda = 8$

We can conclude that larger λ brings more zeros, which effectively promotes sparsity.