Nonlinear Optimization Fall 2021

HW4 Sample Solutions

Q1 (a) This follows similar to our subgradient method analysis in lecture. For fixed x_k, we have for $g_k = \mathbb{E}(g(x_k))$

$$\mathbb{E}\left[\|x_{k+1}-x^*\|^2 \,|\, x_k\right] = \mathbb{E}\left[\|x_k - x^* - \alpha_k g(x_k)\|^2 \,|\, x_k\right]$$

$$= \mathbb{E}\left[\|x_k-x^*\|^2 - 2\alpha_k g(x_k)^T(x_k-x^*) + \alpha_k^2\|g(x_k)\|^2 \,|\, x_k\right]$$

by Linearity of Expectation

$$= \mathbb{E}\left[\|x_k-x^*\|^2 \,|\, x_k\right] - 2\alpha_k \mathbb{E}\left[g(x_k)\,|\,x_k\right]^T(x_k-x^*) + \alpha_k^2 \mathbb{E}\left[\|g(x_k)\|^2 \,|\, x_k\right]$$

$$\leq \underbrace{\|x_k-x^*\|^2}_{\text{constant for fixed } x_k} - 2\alpha_k \underbrace{g_k}^{\text{by unbiased oracle}} (x_k-x^*) + \alpha_k^2 \underbrace{M^2}_{\text{by assumed bound.}}$$

by definition of subgradient at x_k.

$$\leq \|x_k-x^*\|^2 - 2\alpha_k(f(x_k)-f(x^*)) + \alpha_k^2 M^2 \qquad \square$$

(b) To inductively apply (a), we need to first take the expectation with respect to x_k (and everything before the k^{th} iterate). The Law of Total Expectation ensures

$$\mathbb{E}\left[\|x_{k+1}-x^*\|^2\right] \leq \mathbb{E}\left[\|x_k-x^*\|^2\right] - 2\alpha_k \mathbb{E}\left[f(x_k)-f(x^*)\right] + \alpha_k^2 M^2.$$

Hence after k steps we have

$$0 \le \mathbb{E}\|x_k - x^*\|^2 \le \underbrace{\|x_0 - x^*\|^2}_{\text{constant}} - \sum_{i=0}^{k-1} 2\alpha_i \mathbb{E}(f(x_i) - f(x^*)) + \sum_{i=0}^{k-1} \alpha_i^2 M^2.$$

$$\Rightarrow \mathbb{E}\left[\sum_{i=0}^{k-1} \left(\frac{\alpha_i}{\sum_{j=0}^{k-1}\alpha_j}\right)(f(x_i) - f(x^*))\right] \le \frac{\|x_0 - x^*\|^2 + \sum_{i=0}^{k-1}\alpha_i^2 M^2}{2\sum_{i=0}^{k-1}\alpha_i}$$

(rearranging, using linearity on LHS, and dividing through by $2\sum_{i=0}^{k-1}\alpha_i$ on both sides.)

Noting the LHS is a weighted average, we have the result as

$$\mathbb{E}\left[\min_{i \le k-1}\{f(x_i) - f(x^*)\}\right] \le \frac{\|x_0 - x^*\|^2 + \sum_{i=0}^{k-1}\alpha_i^2 M^2}{2\sum_{i=0}^{k-1}\alpha_i}.$$

(c) Selecting $\alpha_i = \frac{1}{\sqrt{k}}$ for any fixed k, our bound becomes

$$\le \frac{\|x_0 - x^*\|^2 + M^2}{2\sqrt{k}}$$

(since $\sum_{i=0}^{k-1}\left(\frac{1}{\sqrt{k}}\right)^2 = k \cdot \frac{1}{k} = 1$ and $\sum_{i=0}^{k-1}\frac{1}{\sqrt{k}} = k \cdot \frac{1}{\sqrt{k}} = \sqrt{k}$.)

which is $O(1/\sqrt{k})$.

(d) Selecting $\alpha_i = \frac{1}{\sqrt{i+1}}$ for each i, our bound becomes for any k ← (be careful not to divide by zero)

$$\le \frac{\|x_0 - x^*\|^2 + M^2 \cdot \log(k+1)}{2\sqrt{k}}$$

(since $\sum_{i=0}^{k-1}\left(\frac{1}{\sqrt{i+1}}\right)^2 = \sum_{i=0}^{k-1}\frac{1}{i+1} \le \log(k+1)$ and $\sum_{i=0}^{k-1}\left(\frac{1}{\sqrt{i+1}}\right) \ge k \cdot \frac{1}{\sqrt{k}} = \sqrt{k}$)

which is $O\left(\frac{\log(k)}{\sqrt{k}}\right)$.

Q2 (a) Applying the chain rule to $h(x)$, we have

$$\begin{aligned}\nabla h(x) &= \nabla F(x)^T \cdot \nabla\left(\tfrac{1}{2}(\|\cdot\|_2^2)\right)(F(x)) \\ &= \nabla F(x)^T \cdot F(x).\end{aligned}$$

This is Lipschitz with the constant $NL + QM$ as for any $x, y \in \mathbb{R}^d$, we have

$$\begin{aligned}\|\nabla h(x) - \nabla h(y)\| &= \|\nabla F(x)^T F(x) - \nabla F(y)^T F(y)\| \\ \text{adding \& subtracting} &= \|\nabla F(x)^T F(x) - \nabla F(x)^T F(y) + \nabla F(x)^T F(y) - \nabla F(y)^T F(y)\| \\ \text{by } \Delta\text{-inequality} &\le \|\nabla F(x)^T F(x) - \nabla F(x)^T F(y)\| \\ &\quad + \|\nabla F(x)^T F(y) - \nabla F(y)^T F(y)\| \\ &= \|\nabla F(x)^T (F(x) - F(y))\| \\ &\quad + \|(\nabla F(x) - \nabla F(y))^T F(y)\| \\ &\le \|\nabla F(x)\| \cdot \|F(x) - F(y)\| \\ &\quad + \|\nabla F(x) - \nabla F(y)\| \cdot \|F(y)\| \\ &\le N \cdot L \|x - y\| \\ &\quad + Q\|x - y\| \cdot M \qquad = (NL + QM)\|x - y\|.\end{aligned}$$

□

(b) This iteration is cheaper than Newton's Method. Both require computing $F(x_k)$ and $\nabla F(x_k)$, but then this method only needs their matrix-vector product, whereas Newton needs to solve a linear system based on these.

To see this cheaper iteration fail to converge to a zero of F, consider $F(x) = -x^2 + 1$, which has zeroes at ± 1.

Then $h(x) = (1-x^2)^2$ has gradient $\nabla h(x) = -4x(1-x^2)$

and so the GD iteration is $x_{k+1} = x_k - 4x_k(1-x_k^2)\alpha_k$.

If $x_0 = 0$, this iteration is constant: $x_k = x_0 = 0$, but $F(0) = 1 \neq 0$

despite ± 1 both having $F(\cdot) = 0$.

(c) These bad points exist because gradient descent only finds local minima of $h(\cdot)$, whereas the target points with $F(x) = 0$ are the global minimizers of $h(\cdot)$. How can we resolve this?

One approach would be to require $h(\cdot)$ is convex, but this would amount to $\nabla^2 h(\cdot) \succeq 0$, which involves $\nabla^3 f(\cdot)$. We want a condition just on ∇f and $\nabla^2 f$.

Instead, we need a condition under which having $\nabla h(\cdot)$ small implies $\nabla f(\cdot)$ is small. The issue is that $\nabla h(x) = \nabla^2 f(x)^T \nabla f(x)$ can be small for two reasons: $\nabla f(x)$ is small or $\nabla f(x)$ is the null space of $\nabla^2 f(x)$ (or at least in the span of small eigenvalues & vectors).

The following condition rules out this second case:

$$\boxed{\text{There exists } \gamma > 0 \text{ s.t. } \forall x,\ \left\|\nabla^2 f(x)^T \nabla f(x)\right\| \geq \gamma \|\nabla f(x)\|.}$$

(This condition does not imply h is convex.)

Under this condition, our GD analysis applies to how

$$\|\nabla h(x_k)\| \to 0 \qquad \text{(also using that } \inf h(x) > -\infty\text{)}$$

and then our condition allows us to extend this to

$$\|\nabla f(x_k)\| \to 0.$$

Note this condition holds whenever our Hessian is full rank (as there is no null space) and so there is still hope for this iteration being useful (in structured settings).

Q3 (a) This system of $n+1$ nonlinear equations is given by

$$F(x,\lambda) = \begin{bmatrix} (A-\lambda I)x \\ x^T x - 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

This has Jacobian given by

$$\nabla F(x,\lambda) = \begin{bmatrix} A-\lambda I & -x \\ 2x^T & 0 \end{bmatrix}.$$

Hence Newton's Method iterates

$$(x_{k+1}, \lambda_{k+1}) = (x_k, \lambda_k) - \begin{bmatrix} A-\lambda I & -x \\ 2x^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} (A-\lambda I)x \\ x^T x - 1 \end{bmatrix}.$$

(Omitting the matrix inverse gives the algorithm for Q2.)
and adding a transpose

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

A = np.array([[4,2,1],
              [2,3,0],
              [1,0,1]])
```

In [2]:
```
def F(x,lam):
    return np.append(np.dot(A-lam*np.eye(3),x), np.dot(x,x)-1)

def Jacob(x,lam):
    return np.block([[A-lam*np.eye(3),   -1*np.array([x]).transpose()],
                     [2*x,    0]])
```

In [3]:
```
x = np.array([1.0/5.0, -1.0/5.0, 4.0/5.0])
lam = 1
for i in range(100):
    p = -1.0*np.dot(np.linalg.inv(Jacob(x,lam)),F(x,lam)) #Newton direction to move in
    x = x + p[0:3]
    lam = lam + p[3]
print(x, lam)
```

```
[ 0.43184431 -0.73306142  0.52548211] 1.8218059199792438
```

In [4]:
```
def backtrack(x,lam,p, alpha0, eta, tau):
    alpha = alpha0
    while (0.5*np.linalg.norm(F(x+alpha*p[0:3], lam + alpha*p[3]))**2
            >= 0.5*np.linalg.norm(F(x, lam))**2 - alpha*eta*np.linalg.norm(p)**2):
        alpha = alpha*tau
    return alpha
```

In [5]:
```
x = np.array([1.0/5.0, -1.0/5.0, 4.0/5.0])
lam = 1
for i in range(100):
    p = -1.0*np.dot(Jacob(x,lam).transpose(),F(x,lam)) #Gradient direction to move in
    alpha = backtrack(x,lam,p,100.0,0.1,0.9)
    x = x + alpha*p[0:3]
    lam = lam + alpha*p[3]
print(x, lam)
```

```
[ 0.43212805 -0.73282039  0.52558097] 1.8197296365817155
```