VE485 HomeWork 6

Pan, Chongdan ID:516370910121

July 31, 2020

1 LS Problem

To $\min_x ||Ax - b||$, it's equal to minimize $f(x) = (Ax - b)^T(Ax - b)$
It's gradient is $\bigtriangledown f(x) = 2A^T(Ax - b)$
When $f(x) = 0$, we get the pseudo inverse $\hat{x} = A^T(AA^T)^{-1}b$
I set the A, X, b randomly, stopping critera $\epsilon = 0.01$
For Backtracking search $\alpha = 0.2, \beta = 0.25$
After a lot of test, I get results similar to following table

Input	Final Error	Iterations
$\hat{x}$	31.57	0
$0.5\hat{x}$	31.57	30
$0.75x$	7.892	28
$0.25x$	23.67	31
Random	44.61	33
x+Random	30.81	30
1	43.64	33
0	31.57	31
0.5	54.87	31

2 Analysis

For $X_{input} = \hat{x}$, I think it may be a saddle point since it has 0 iterations but still have error between X. It may also because A is not symmetric. What's more, if $X_{input} = k\hat{x}$, it will lead to same error after some iterations.

For $X_{input} = kX$, it's surprising that kX's error is always lower when $X_{input} = \hat{x}$, hence, they can also lead to similar saddle point, but it's closer to the optimal value, hence there are multiple saddle points. When $k \to 1$, the error is smaller because it's saddle point is closer to the original solution.

For $X_{input} = X_{random}$, we can find it always achieves the largest error with most iterations, it may because there are too much saddle points away from X and the random X will lead to it. But I also found that when $X_{input} = X_{random} + X$, error will be smaller.

For constant matrix, we find it will lead to results similar to $\hat{x}$, this may have relation with sparsity.

3 With Noise

3.1 Model for $Ax = b$

If we use the old model for $Ax = b$, we will get similar result when the noise σ^2 is small, such as $\sigma \in (0, 10)$

Input	Final Error	Iterations
$\hat{x}$	30.52	0
$0.5\hat{x}$	30.52	29
$0.75x$	7.839	29
$0.25x$	22.93	31
Random	43.33	32
x+Random	31.26	31
1	43.24	32
0	30.52	31
0.5	34.17	30

Table 1: When $\sigma = 10$

However, when σ increase bigger, all error will approach the same value.

Input	Final Error	Iterations
$\hat{x}$	2230	0
$0.5\hat{x}$	2230	43
$0.75x$	2230	44
$0.25x$	2230	44
Random	2230	44
x+Random	2230	44
1	2230	44
0	2230	44
0.5	2230	44

Table 2: When $\sigma = 100$

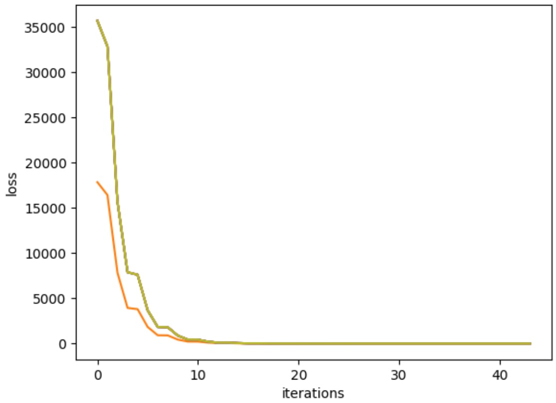

Figure 1: Loss vs. iteration for GD model without considering $\sigma = 100$

The iteration for $\hat{x}$ is still 0, which means it's still at the saddle point. The error will also increase with the increase of σ. It also takes more iterations to get the optimal value.

3.2 Model for $Ax + \varepsilon = b$

In this case, our objective is to minimize $||Ax+\varepsilon-b||$, hence the gradient $\bigtriangledown f(x) = 2A^T(Ax+\varepsilon-b)$, $\hat{x}$ is also $\hat{x} = A^T(AA^T)^{-1}(b-\varepsilon)$ now.

After we change $f(x), \bigtriangledown f(x)$, and $\hat{x}$, we can get same result as $\varepsilon = 0$ no matter how large ε

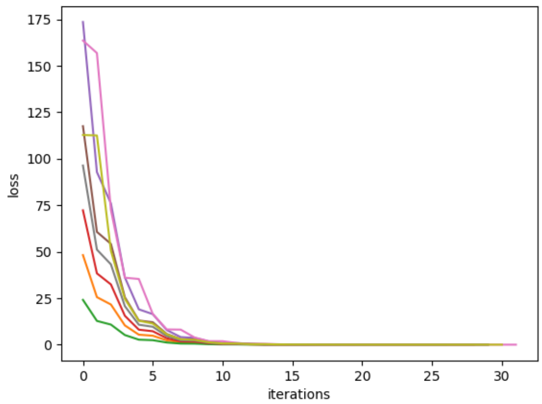

Figure 2: Loss vs. iteration for GD model considering $\sigma = 100$

Input	Final Error	Iterations
$\hat{x}$	31.77	0
$0.5\hat{x}$	31.77	29
$0.75x$	7.94	27
$0.25x$	23.83	30
Random	45.17	31
x+Random	32.37	30
1	43.60	32
0	31.77	30
0.5	34.91	30

Table 3: When $\sigma = 100$

Python Code

```
import numpy as np
import matplotlib.pyplot as plt
# loss function
def loss(A, b, X, delta):
    y = np.dot(A, X) + delta - b # 50*1
    return np.dot(y.T, y)

# GD algorithm
def GD(A, b, X0, delta):
    e = 1e-2
    a = 0.2
    beta=0.25
    k = 0
    X = X0
    g = 2*np.dot(A.T,np.dot(A,X) + delta - b)
    n = []
    m = []
    while np.linalg.norm(g)>e: # When the gradient is not small enough
        g=2*np.dot(A.T,np.dot(A,X) + delta - b) # Calculate the negative gradient 1000*1
        t=1
        while 1: # Backtracking
            Xt = X-t*g # 1000*1
            if loss(A, b, Xt, delta) < loss(A, b, X, delta) + a*t*np.dot(g.T, (Xt - X)):
                break
            t *= beta
        X = Xt
        n.append(k)
        m.append(np.linalg.norm(np.dot(A,X) + delta - b))
        k += 1
    plt.plot(n, m)
    return [X,k]

# generate the dataset
```

```
delta = 1 * (100**2)*np.random.randn(50, 1)
X = np.random.randn(1000, 1)
A = np.random.randn(50, 1000)
b = np.dot(A, X) + delta


# find the optimal solution through pseudo inverse directly
xhat = np.linalg.inv(np.dot(A, A.T))
xhat = np.dot(A.T, xhat)
xhat = np.dot(xhat, b-delta)
# print the rank of A
print('The rank of A is', np.linalg.matrix_rank(A))
# find the solution through GD
Result = GD(A, b, xhat, delta)
print('error for x0=xhat is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'iterations')

# find the solution through GD
Result = GD(A, b, 0.5*xhat, delta)
print('error for x0=0.5xhat is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'iteration

# find the solution through GD
Result = GD(A, b, 0.75*X, delta)
print('error for x0=0.75x is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'iterations'

# find the solution through GD
Result = GD(A, b, 0.25*X, delta)
print('error for x0=0.25x is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'iterations'

# find the solution through GD
X0 = np.random.randn(1000, 1)
Result = GD(A, b, X0, delta)
print('error for x0=random matrix is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'ite

# find the solution through GD
Result = GD(A, b, X+X0, delta)
print('error for x0=x+random matrix is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'i

# find the solution through GD
X0 = np.ones((1000, 1))
Result = GD(A, b, X0, delta)
print('error for x0=1 is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'iterations')

# find the solution through GD
Result = GD(A, b, 0*X0, delta)
print('error for x0=0 is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'iterations')

# find the solution through GD
Result = GD(A, b, 0.5*X0, delta)
print('error for x0=0.5 is', np.linalg.norm(X-Result[0]), ' with ', Result[1], 'iterations')
```

```
plt.xlabel('iterations')
plt.ylabel('loss')
plt.axis([0,10,0,200])
plt.show()
```