CS 189 Introduction to Machine Learning
Fall 2017 HW6

This homework is due **Friday, October 6 at 10pm.**

1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit all code needed to reproduce your results, "HW[n] Code".
3. Submit your test set evaluation results, "HW[n] Test Set".

After you've submitted your homework, be sure to watch out for the self-grade form.

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

None. work alone
Comments : none

(b) Please copy the following statement and sign next to it:

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

I certify that all solutions are entirely my words and that I have not looked at another student's solutions. I have credited all external sources in this write up

Problem #2.

a.) $f(x) = \|x - b\|_2$

We know that $\|a\| + \|b\| \geq \|a + b\|$ for norm-2.

$$\Rightarrow \quad \lambda \|x_1 - b\| + (1-\lambda)\|x_2 - b\|$$

$$= \|\lambda x_1 - \lambda b\| + \|(1-\lambda)x_2 - (1-\lambda)b\|$$

$$\geq \|\lambda x_1 + (1-\lambda)x_2 \underbrace{- \lambda b - (1-\lambda)b}_{-b}\|$$

$$= f(\lambda x_1 + (1-\lambda)x_2) \qquad \forall x_1, x_2 \in R^d$$

i.e. $\lambda f(x_1) + (1-\lambda) f(x_2) \geq f(\lambda x_1 + (1-\lambda) x_2)$

$\forall x_1, x_2 \in R^d$

$\Rightarrow$ convex

b.) $f(x) = \|x - b\|_2 = \sqrt{(x-b)^T (x-b)}$

$$\Rightarrow \nabla f(x) = \frac{\frac{d}{dx}(x-b)^T(x-b)}{2\sqrt{(x-b)^T(x-b)}}$$

nominator $= \frac{d}{dx}(x^Tx - x^Tb - b^Tx + b^Tb) = 2(x-b)$

denominator $= 2\|x-b\|$

$$\Rightarrow \nabla f(x) = \frac{(x-b)}{\|x-b\|} \quad \text{unit vector}$$

$x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad b = \begin{bmatrix} 4.5 \\ 6 \end{bmatrix}$

$x_1 = x_0 - \nabla f(x_0) = [0.6, 0.8]^T$, $f(x_1) = 6.5$

$x_2 = x_1 - \nabla f(x_1) = [1.2, 1.6]^T$, $f(x_2) = 5.5$

$x_3 = x_2 - \nabla f(x_2) = [1.8, 2.4]^T$, $f(x_3) = 4.5$

...

$x_8 = x_7 - \nabla f(x_7) = [4.8, 6.4]^T$, $f(x_8) = 0.5$

$x_8 = x_8 - \nabla f(x_8) = [4.2, 5.6]^T$, $f(x_9) = 0.5$

$x_{10} = x_9 - \nabla f(x_9) = [4.8, 6.4]^T$, $f(x_{10}) = 0.5$

the gradient descent does not find the optimal solution
(we know that opt. sol is $x = b = [4.5, 6]^T$ where $f(x) = 0$)

Reason: step is too large

Proof:

At $(i+1)$-th iteration:

$$x_{i+1} = x_i - \nabla f(x_i) = x_i - \frac{x_i - b}{\|x_i - b\|}$$

we know that the true solution is $x = b$. Consider

$$x_{i+1} - b = x_i - b - \frac{x_i - b}{\|x_i - b\|}$$

$$= (x_i - b)\left(1 - \frac{1}{\|x_i - b\|}\right)$$

$$= (x_i - b)\,\frac{\|x_i - b\| - 1}{\|x_i - b\|}$$

we want $\|x_{i+1} - b\| \leq 0.01$

$$\|x_i - b\| \frac{|\,\|x_i - b\| - 1\,|}{\|x_i - b\|} \leq 0.01$$

$$\|x_{i+1} - b\| = \big|\|x_i - b\| - 1\big| \leq 0.01$$

Let $e_{i+1} = \|x_{i+1} - b\|$

$$\Rightarrow \quad e_{i+1} = |e_i - 1|$$

$$e_{i+1} \leq 0.01$$

$$1 - 0.01 \leq e_i \leq 1 + 0.01$$

$$i - 0.01 \leq e_1 \leq i + 0.01$$

$$i - 0.01 \leq \left\| \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 4.5 \\ 6 \end{bmatrix} \right\| \leq i + 0.01$$

$$i - 0.01 \leq 7.5 \leq i + 0.01$$

$\rightarrow$ no integer i $=$

if $b = \begin{bmatrix} 6 \\ 8 \end{bmatrix} \Rightarrow$ we can find $i = 10$

in general for $b \neq \vec{0}$, the gradient descent converges with step size 1 iff e_1 is integer. if we take $x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow e_1 = \sqrt{b_1^2 + b_2^2}$ is integer (where $b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$

iff b_1 & b_2 are two smaller numbers of Pythagorean triple.

c.) Similar to question (b), we have:

$$e_{i+1} = \left| e_i - \left(\frac{5}{6}\right)^i \right|$$

$$e_{i+1} \leq 0.01$$

$$-0.01 \leq e_i - \left(\frac{5}{6}\right)^i \leq 0.01$$

$$\left(\frac{5}{6}\right)^i - 0.01 \leq e_i \leq \left(\frac{5}{6}\right)^i + 0.01$$

$$\left(\frac{5}{6}\right)^i - 0.01 \leq \left| e_{i-1} - \left(\frac{5}{6}\right)^{i-1} \right| \leq \left(\frac{5}{6}\right)^i + 0.01$$

$$\left(\frac{5}{6}\right)^{i-1} + \left(\frac{5}{6}\right)^i - 0.01 \leq e_{i-1} \leq \left(\frac{5}{6}\right)^{i-1} + \left(\frac{5}{6}\right)^i + 0.01$$

$$\frac{5}{6}\,\frac{1-\left(\frac{5}{6}\right)^i}{1-\frac{5}{6}} - 0.01 \leq e_1 \leq \frac{5}{6}\,\frac{1-\left(\frac{5}{6}\right)^i}{1-\frac{5}{6}} + 0.01$$

$$5 \times \left(1-\left(\frac{5}{6}\right)^i\right) - 0.01 \leq 7.5 \leq 5 \times \left|1-\left(\frac{5}{6}\right)^i\right| + 0.01$$

around 5 — around 5

→ no such i integer with $b = \begin{bmatrix} 4.5 \\ 6 \end{bmatrix}$

But if $e_1 \simeq 5$, i.e $\sqrt{b_1^2 + b_2^2} \simeq 5$

say $b_1 = 3$, $b_2 = 4$, we can find

$$5\left(1-\left(\frac{5}{6}\right)^i\right) - 0.01 \leq 5 \leq 5 \times \left(1-\left(\frac{5}{6}\right)^i\right) + 0.01$$

always ⇒ $i = 35$

(d) Again, we have:

$$e_{i+1} = \left| e_i - \frac{1}{i+1} \right|$$

$e_1 = 7.5 \Rightarrow e_2 = \left| 7.5 - \frac{1}{1+1} \right| = 7$

$e_3 = \left| 7 - \frac{1}{3} \right| = \frac{20}{3}$

...

write a function to test:

```
def test(k):
    e1 = 7.5
    for i in range(2, k+1):
        e1 -= 1/i
    return e1
```

```
>>> test(2731)
0.0107
>> test(2732)
0.0098
```

$\Rightarrow i = 2732$

For general $b \neq \vec{0}$, we have arbitrary error e_1. Since the harmonic series diverges, we can always find i such that after i iteration, the error

$e_i = \left| e_{i-1} - \frac{1}{i} \right|$ converges to 0

e.) See appendix

for step size 1 and $(\frac{5}{6})^i$ it is very easily converged.

But there are some case it converge, for example

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

step size = 1 → converge after 5 time step

step size = $(\frac{5}{6})^i$ → converge after 35 time steps

...

But if step size = $\frac{1}{i+1}$ → always converge after some finite steps.

Explanation:

For step sizes 1 & $(\frac{5}{6})^i$ it is either too big or too small step size to converge

For step size $\frac{1}{i}$ → we use harmonic series whose alternate series can conve to any number to make.

error − (alternate harmonic series at some point)

converge to 0

Problem #3

a.) $f(x) = \frac{1}{2}\|Ax - b\|_2^2$.

$b = \vec{0} \Rightarrow f(x) = \frac{1}{2}\|Ax\|_2^2 = \frac{1}{2}x^TA^TAx$

$\Rightarrow \nabla f(x) = A^TAx$

$\Rightarrow$ the state evolution of iterations of gradient descent

$x_{i+1} = x_i - \gamma \nabla f(x_i)$

$x_{i+1} = x_i - \gamma A^TAx_i = (I - \gamma A^TA)x_i$

(b) From above, we can write

$x_{i+1} = (I - \gamma A^TA)^{i+1} x_0$

The scheme does not blow up if all eigen values of $(I - \gamma A^TA)$ less than or equal 1.

(c) $\varphi(x) = x - \gamma \nabla f(x)$

$LHS = \|\varphi(x) - \varphi(x')\| = \|x - \gamma \nabla f(x) - x' + \gamma \nabla f(x')\|$

$f(x) = \frac{1}{2}\|Ax - b\|_2^2 \Rightarrow \nabla f(x) = A^TAx - b$

$\Rightarrow LHS = \|x - x' - \gamma A^TAx - \gamma b + \gamma A^TAx' + \gamma b\|$

$= \|x - x' - \gamma A^TA(x - x')\|$

$= \|(I - \gamma A^TA)(x - x')\|$

We know that $(I - \gamma A^T A)(x - x') = \lambda (I - \gamma A^T A)(x - x')$

(λ: eigen value)

$$\Rightarrow \| (I - \gamma A^T A)(x - x') \| = \| \lambda (I - \gamma A^T A)(x - x') \|$$

$$= \|x - x'\| \, | \lambda (I - \gamma A^T A) |$$

$$\leq \|x - x'\| \, | \lambda_{max} (I - \gamma A^T A) |$$

we know that $\lambda (I + A) = 1 + \lambda(A)$

(proof: $(I + A)x = \lambda (I + A) x$

$Ix + Ax = x + \lambda(A) x$

$\Rightarrow \lambda (I + A) x = x (1 + \lambda(A))$

$\Rightarrow \lambda (I + A) = 1 + \lambda(A)$)

$$\Rightarrow |\lambda_{max} (I - \gamma A^T A)| = \max \{ |\lambda (I - \gamma A^T A)| \}$$

$$= \max \{ |1 - \lambda(\gamma A^T A)| \}$$

$$= \max \{ |1 - \gamma \lambda_{max} (A^T A)|, |1 - \gamma \lambda_{min} (A^T A)| \}$$

$$= \beta$$

$$\Rightarrow \| (I - \gamma A^T A)(x - x^T) \leq \|x - x'\| \beta$$

i.e. $\| \varphi(x) - \varphi(x') \| \leq \beta \|x - x'\|$

From previous part, we do this to avoid blow up

(d) $\|\varphi(x_k) - \varphi(x^*)\| = \|x_k - \gamma\nabla f(x_k) - x^* + \gamma\nabla f(x^*)\|$

$x_{k+1} = x_k - \gamma\nabla f(x_k)$

$\nabla f(x^*) = 0$ since x^* is minimum point

$\Rightarrow \|\varphi(x_k) - \varphi(x^*)\| = \|x_{k+1} - x^*\|$

$\|x_{k+1} - x^*\| = \|\varphi(x_k) - \varphi(x^*)\| \leq \beta\|x_k - x^*\|$

$= \beta\|\varphi(x_{k-1}) - \varphi(x^*)\|$

$\leq \beta^2\|x_{k-1} - x^*\|$

$= \beta^2\|\varphi(x_{k-2}) - \varphi(x^*)\|$

$\leq \beta^3 \ldots$

$\Rightarrow \|x_{k+1} - x^*\| \leq \beta^{k+1}\|x_0 - x^*\|$

e.) $f(x) - f(x^*) = \frac{1}{2}\|Ax - b\|^2 - \frac{1}{2}\|Ax^* - b\|^2$

$= \frac{1}{2}\|Ax - b\|^2 - (Ax - b)^T(Ax^* - b) + (Ax - b)^T(Ax^* - b)$

$+ \frac{1}{2}\|Ax^* - b\|^2 - \|Ax^* - b\|^2$

$= \frac{1}{2}(Ax - b)^T(Ax - b) - (Ax - b)^T(Ax^* - b) + \frac{1}{2}(Ax^* - b)^T(Ax - b)$

$+ (Ax - b)^T(Ax^* - b) - (Ax^* - b)^T(Ax^* - b)$

$= \frac{1}{2}(Ax - b - Ax^* + b)^T(Ax - b - Ax^* + b) + (Ax - b - Ax^* + b)^T(Ax^* - b)$

$(A(x - x^*))^T \qquad A(x - x^*)$

$$= \frac{1}{2}\left(A(x-x^*)\right)^T\left(A(x-x^*)\right) + \underbrace{(Ax-b-Ax^*+b)^T(Ax^*-b)}_{0 \text{ when } f(x) \text{ converges to } f(x^*)}$$

$$= \frac{1}{2}\|A(x-x^*)\|^2$$

Thus $$f(x) - f(x^*) = \frac{1}{2}\|A(x-x^*)\|_2^2$$

when $f(x) \to f(x^*)$

(f). From (e):

$$f(x_k) - f(x^*) = \frac{1}{2}\|A(x_k - x^*)\|_2^2$$

$$= \frac{1}{2}(x_k - x^*)^T \underbrace{A^TA}_{\lambda(A^TA)}(x_k - x^*)$$

$$= \frac{1}{2}\lambda(A^TA)\underbrace{(x_k-x^*)^T(x_k-x^*)}_{\|x_k-x^*\|^2}$$

$$= \frac{1}{2}\lambda(A^TA)\|x_k - x^*\|^2$$

$$\leq \frac{1}{2}\underbrace{\lambda_{max}(A^TA)}_{\alpha}\|x_k - x^*\|^2$$

From (d): $\|x_k - x^*\| \leq \beta^k \|x_0 - x^*\|$

$$\Rightarrow \|x_k - x^*\|^2 \leq \beta^{2k}\|x_0 - x^*\|^2$$

$$\Rightarrow f(x_k) - f(x^*) \leq \frac{\alpha}{2}\beta^{2k}\|x_0 - x^*\|^2$$

$$f(x_k) - f(x^*) = \frac{\alpha}{2} \beta^{2k} \| x_0 - x^* \|_2^2$$

iff $f(x_k)$ converges to $f(x^*)$

(g) Pick γ such that

$$1 - \gamma \lambda_{max}(A^T A) = \gamma \lambda_{min}(A^T A) - 1$$

$$\Rightarrow \quad \gamma = \frac{2}{\lambda_{min}(A^T A) + \lambda_{max}(A^T A)}$$

$$\Rightarrow \quad \beta = |1 - \gamma \lambda_{max}(A^T A)|$$

$$= \left| 1 - \frac{2\lambda_{max}}{\lambda_{min} + \lambda_{max}} \right|$$

$$= \left| 1 - \frac{2\alpha}{1+\alpha} \right| = \frac{|1-\alpha|}{1+\alpha}$$

$\Rightarrow$ The convergence rate $\frac{|1-\alpha|}{1+\alpha}$

Problem #4

(a) $$L(x_1, y_1) = -\sum_{i=1}^{7} \left(\sqrt{(a_i - x_1)^2 + (b_i - y_1)^2} - d_i \right)^2$$

$$\frac{\partial L}{\partial x_1} = -\sum_{i=1}^{7} \frac{\partial}{\partial x_1} \left((a_i - x_1)^2 + (b_i - y_1)^2 - 2 d_i \sqrt{(a_i - x_1)^2 + (b_i - y_1)^2} + d_i^2 \right)$$

$$= -\sum_{i=1}^{7} \left(2x_1 - 2a_i - 2d_i \frac{2x_1 - 2a_i}{2\sqrt{(a_i - x_1)^2 + (b_i - y_1)^2}} \right)$$

$$= -\sum_{i=1}^{7} 2 \left(x_1 - a_i - d_i \frac{x_1 - a_i}{\sqrt{(a_i - x_1)^2 + (b_i - y_1)^2}} \right)$$

$$\frac{\partial}{\partial y_1} = -\sum_{i=1}^{7} \frac{\partial}{\partial y_1} \left((a_i - x_1)^2 + (b_i - y_1)^2 - 2 d_i \sqrt{(a_i - x_1)^2 + (b_i - y_1)^2} + d_i^2 \right)$$

$$= -\sum_{i=1}^{7} \left(2y_1 - 2b_i - 2d_i \frac{2y_1 - 2b_i}{2\sqrt{(a_i - x_1)^2 + (b_i - y_1)^2}} \right)$$

$$= -\sum_{i=1}^{7} 2 \left(y_1 - b_i - d_i \frac{y_1 - b_i}{\sqrt{(a_i - x_1)^2 + (b_i - y_1)^2}} \right)$$

(b) See appendix

Problem #15

done

Problem # 6

Given:

$$A = \begin{bmatrix} 3 & -1 & 1 \\ -1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 7 \\ -7 \end{bmatrix}$$

a.) Can the method gradient descent can be used to solve the equation $Ax = b$?

Solution

a.) The sufficient condition is A psd.

$\det([3]) = 3 > 0$, $\det\left(\begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}\right) = 8 > 0$

$\det\left(\begin{bmatrix} 3 & -1 & 1 \\ -1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix}\right) = 20 > 0$ (using python)

$\Rightarrow$. A is PSD $\Rightarrow$ can use gradient descent.

b.) Given a quadratic function:

$$F(x) = \frac{1}{2} x^T A x - x^T b$$

Derive the line search for this function using gradient descent

Solution

$$\nabla F(x) = Ax - b$$

Assume $x = x_0 + \alpha v$ is the line search w/ $v, x^0 \in R^n$ given

$$f(\alpha) = J(x_0 + \alpha v) = \frac{1}{2}(x_0 + \alpha v)^T A (x_0 + \alpha v) - (x_0 + \alpha v)^T b$$

$$= \frac{1}{2}\left[x_0^T A x_0 + 2\alpha v^T A x_0 + \alpha^2 v^T A v\right] - x_0^T b - \alpha v^T b$$

$f(\alpha)$ should be minimum

$$\Rightarrow \frac{\partial f(\alpha)}{\partial \alpha} = v^T A x_0 + \alpha v^T A v - v^T b = 0$$

$$\Rightarrow \alpha = \frac{v^T (b - A x_0)}{v^T A v}$$

In [70]:
```
import numpy as np
import matplotlib.pyplot as plt


# Question 2b

x0 = np.array([0, 0]).reshape(2, 1)
b = np.array([4.5, 6]).reshape(2, 1)
f = lambda x: np.sqrt( np.dot( (x - b).T, (x - b) ) )
delta_f = lambda x: (x - b) / np.sqrt( np.dot( (x - b).T, (x - b) ) )

for _ in range(10):
    x0 = x0 - delta_f(x0)
    print(x0.flatten(), f(x0).flatten())
```

```
[ 0.6  0.8] [ 6.5]
[ 1.2  1.6] [ 5.5]
[ 1.8  2.4] [ 4.5]
[ 2.4  3.2] [ 3.5]
[ 3.  4.] [ 2.5]
[ 3.6  4.8] [ 1.5]
[ 4.2  5.6] [ 0.5]
[ 4.8  6.4] [ 0.5]
[ 4.2  5.6] [ 0.5]
[ 4.8  6.4] [ 0.5]
```

In [73]:
```
# Question 2c

x0 = np.array([0, 0]).reshape(2, 1)
b = np.array([3, 4]).reshape(2, 1)
f = lambda x: np.sqrt( np.dot( (x - b).T, (x - b) ) )
delta_f = lambda x: (x - b) / np.sqrt( np.dot( (x - b).T, (x - b) ) )

for i in range(10):
    x0 = x0 - (5 / 6) ** i * delta_f(x0)
    print(x0.flatten(), f(x0).flatten())
```

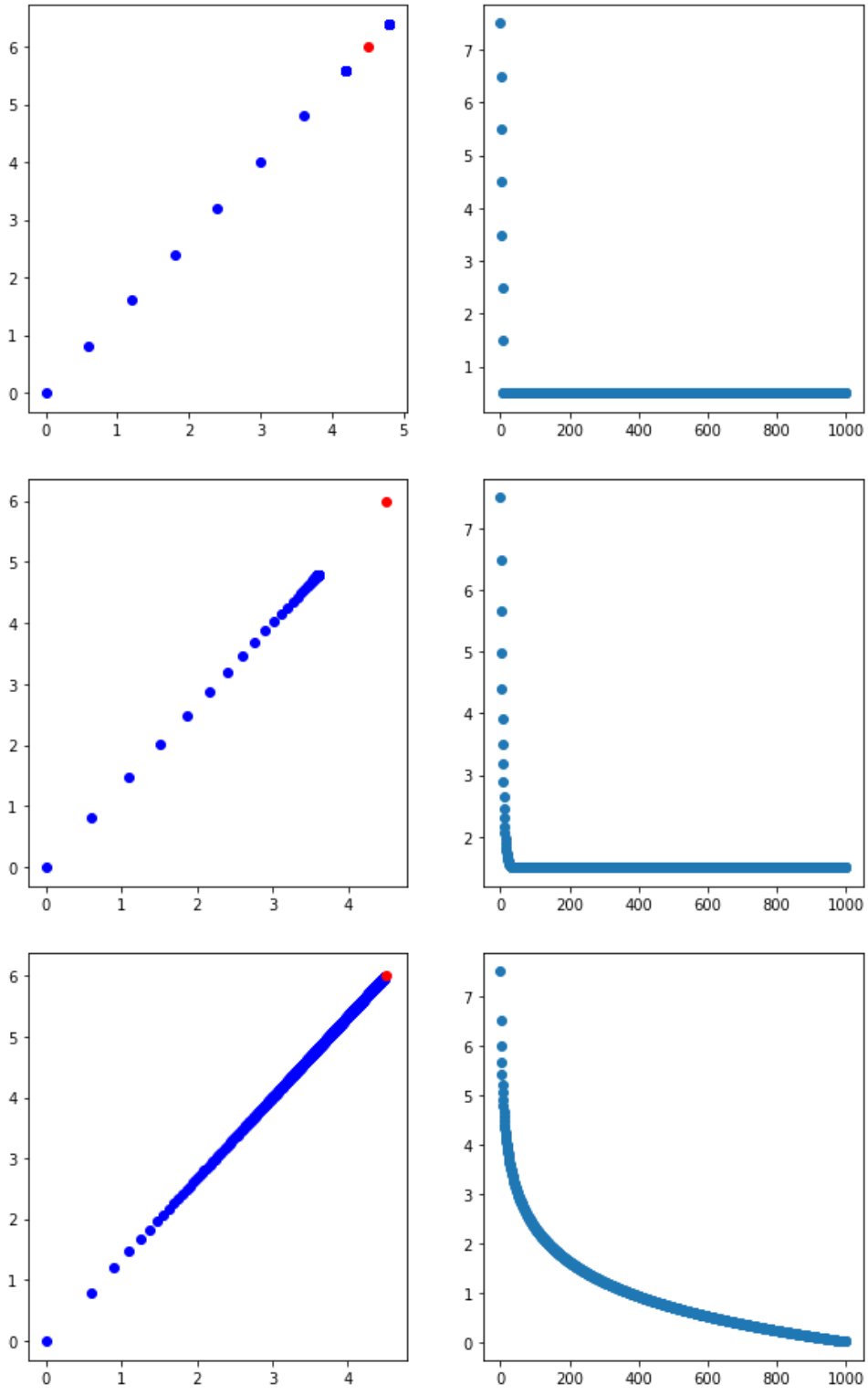
```
[ 0.6  0.8] [ 4.]
[ 1.1         1.46666667] [ 3.16666667]
[ 1.51666667  2.02222222] [ 2.47222222]
[ 1.86388889  2.48518519] [ 1.89351852]
[ 2.15324074  2.87098765] [ 1.41126543]
[ 2.39436728  3.19248971] [ 1.00938786]
[ 2.59530607  3.46040809] [ 0.67448988]
[ 2.76275506  3.68367341] [ 0.39540824]
[ 2.90229588  3.86972784] [ 0.1628402]
[ 3.0185799  4.0247732] [ 0.0309665]
```

In [56]:
```
# Question 2d

x0 = np.array([0, 0]).reshape(2, 1)
b = np.array([4.5, 6]).reshape(2, 1)
f = lambda x: np.sqrt( np.dot( (x - b).T, (x - b) ) )
delta_f = lambda x: (x - b) / np.sqrt( np.dot( (x - b).T, (x - b) ) )

for i in range(10):
    x0 = x0 - 1 / (i + 1) * delta_f(x0)
    print(x0.flatten(), f(x0).flatten())
```

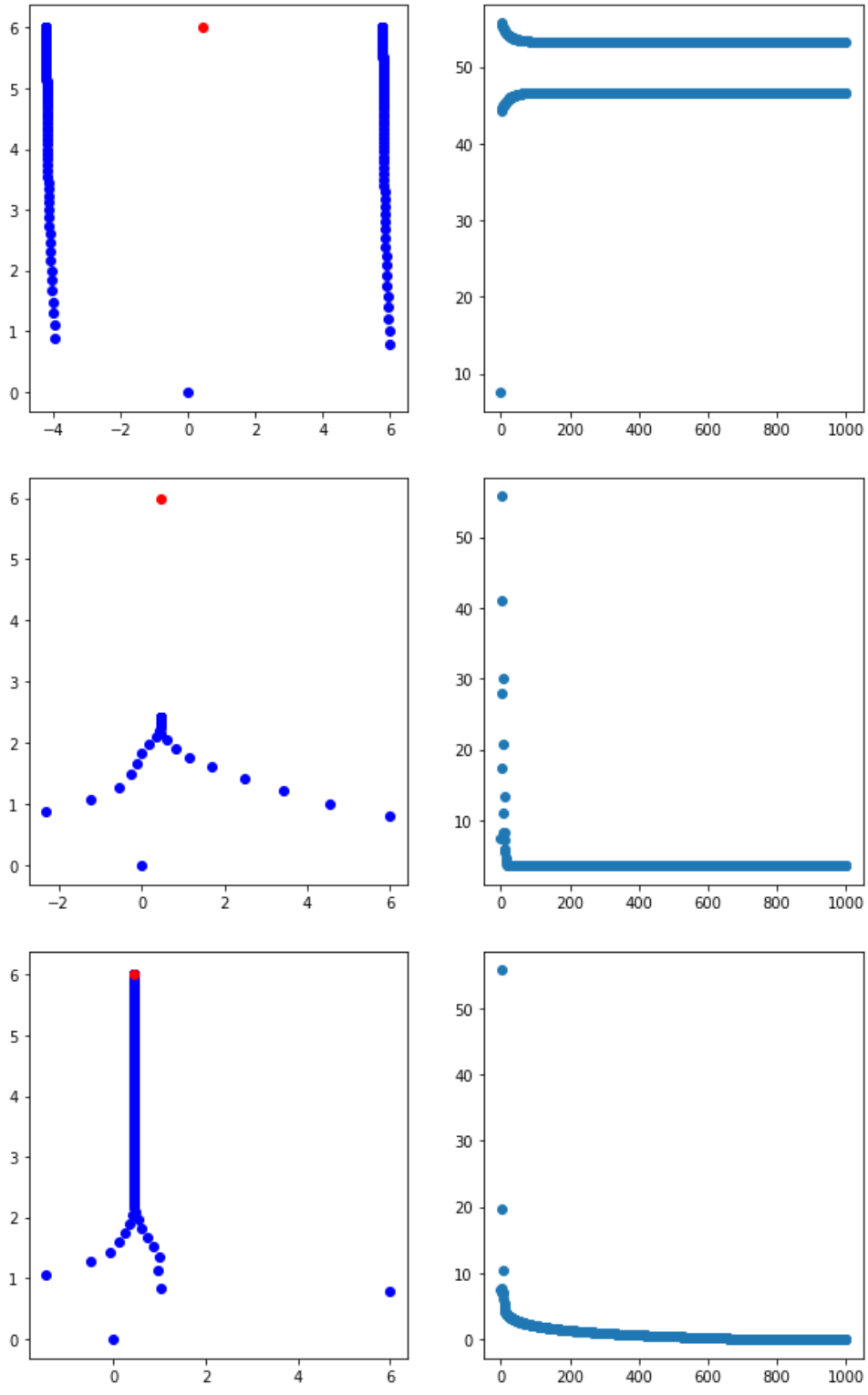
```
[ 0.6  0.8] [ 6.5]
[ 0.9  1.2] [ 6.]
[ 1.1         1.46666667] [ 5.66666667]
[ 1.25        1.66666667] [ 5.41666667]
[ 1.37        1.82666667] [ 5.21666667]
[ 1.47  1.96] [ 5.05]
[ 1.55571429  2.07428571] [ 4.90714286]
[ 1.63071429  2.17428571] [ 4.78214286]
[ 1.69738095  2.2631746 ] [ 4.67103175]
[ 1.75738095  2.3431746 ] [ 4.57103175]
```

In [65]:
```
def main(A, step_size):
    ##########################################################################
#########
    # TODO(student): Input Variables
    A  # do not change this until the last part
    b = np.array([4.5, 6])  # b in the equation ||Ax-b||
    initial_position = np.array([0, 0])  # position at iteration 0
    total_step_count = 1000  # number of GD steps to take
    step_size  # step size at iteration i
    ##########################################################################
#########

    # computes desired number of steps of gradient descent
    positions = compute_updates(A, b, initial_position, total_step_count, s
tep_size)

    # print out the values of the x_i
#     print(positions)
#     print(np.dot(np.linalg.inv(A), b))

    fig = plt.figure(figsize=(10, 5))
    # plot the values of the x_i
    ax1 = fig.add_subplot(121)
    ax1.scatter(positions[:, 0], positions[:, 1], c='blue')
    ax1.scatter(np.dot(np.linalg.inv(A), b)[0],
                np.dot(np.linalg.inv(A), b)[1], c='red')
    ax1.plot()

    # plot the values of f
    f = []
    for x in positions:
        f.append( np.linalg.norm(A.dot(x) - b) )
    ax2 = fig.add_subplot(122)
    ax2.plot(f, "o")
    plt.show()


def compute_gradient(A, b, x):
    """Computes the gradient of ||Ax-b|| with respect to x."""
    return np.dot(A.T, (np.dot(A, x) - b)) / np.linalg.norm(np.dot(A, x) -
b)


def compute_update(A, b, x, step_count, step_size):
    """Computes the new point after the update at x."""
    return x - step_size(step_count) * compute_gradient(A, b, x)


def compute_updates(A, b, p, total_step_count, step_size):
    """Computes several updates towards the minimum of ||Ax-b|| from p.

    Params:
        b: in the equation ||Ax-b||
        p: initialization point
        total_step_count: number of iterations to calculate
        step_size: function for determining the step size at step i
    """
    positions = [np.array(p)]
    for k in range(total_step_count):
        positions.append(compute_update(A, b, positions[-1], k, step_size))
    return np.array(positions)
```

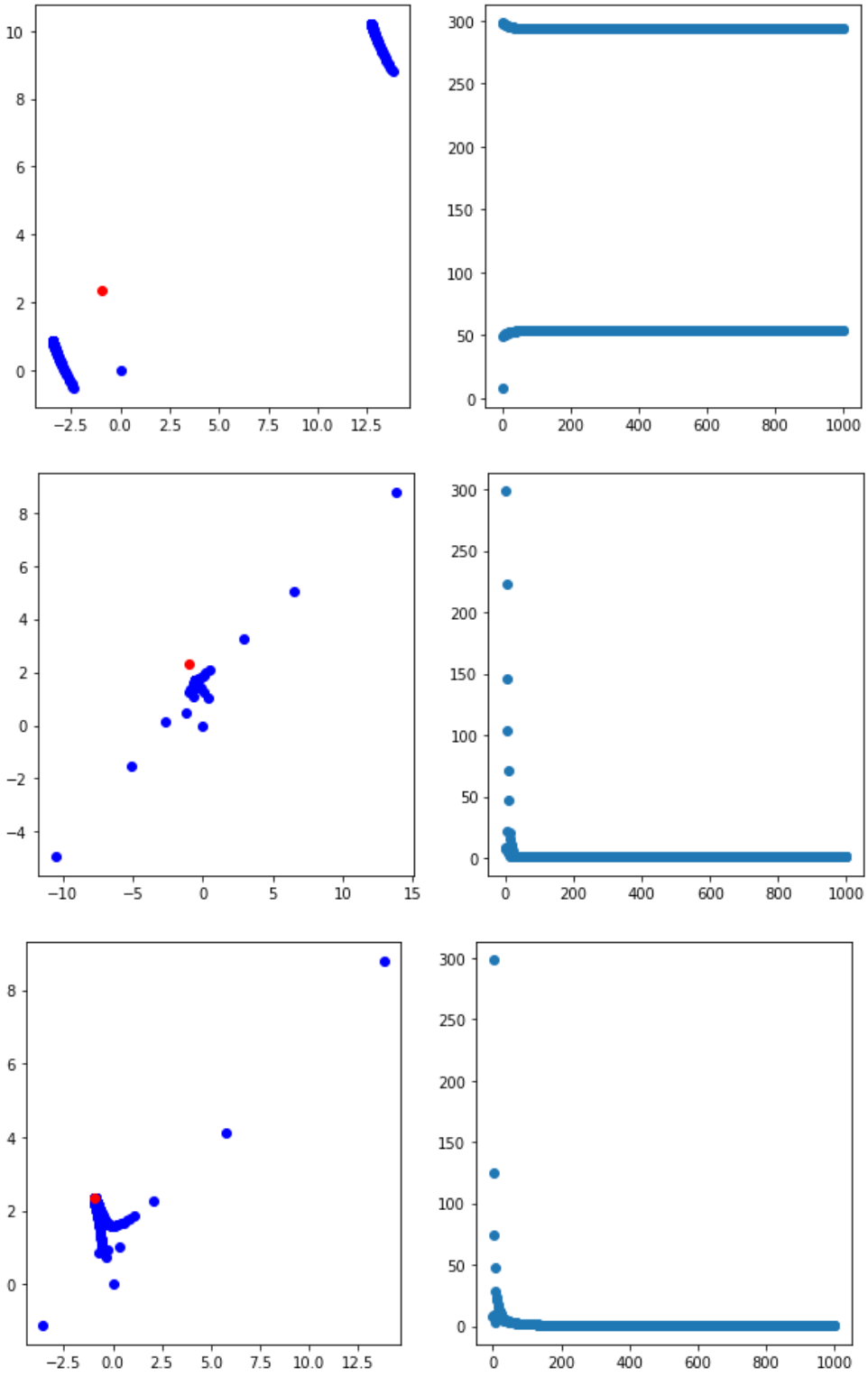
```
[[1 0]
 [0 1]]
```



```
[[10  0]
 [ 0  1]]
```



```
[[15  8]
 [ 6  5]]
```


10
8
6
4
2
0
−2.5 0.0 2.5 5.0 7.5 10.0 12.5
300
250
200
150
100
50
0
0 200 400 600 800 1000
8
6
4
2
0
−2
−4
−10 −5 0 5 10 15
300
250
200
150
100
50
0
0 200 400 600 800 1000
8
6
4
2
0
−2.5 0.0 2.5 5.0 7.5 10.0 12.5
300
250
200
150
100
50
0
0 200 400 600 800 1000

In [106]:
```
# Question 4

import scipy.spatial

def gradient(position, locations, distances):
    df_dx1_vector = 2 * (position[0] - locations[:, 0] - \
                  distances * (position[0] - locations[:, 0]) \
                  / np.sqrt( (locations[:, 0] - position[0]) ** 2 + (locati
ons[:, 1] - position[1]) ** 2 ))
    df_dx1 = np.sum(df_dx1_vector)

    df_dy1_vector = 2 * (position[1] - locations[:, 1] - \
                  distances * (position[1] - locations[:, 1]) \
                  / np.sqrt( (locations[:, 0] - position[0]) ** 2 + (locati
ons[:, 1] - position[1]) ** 2))
    df_dy1 = np.sum(df_dy1_vector)
    return np.array([df_dx1, df_dy1])


##########################################################################
#########  Data Generating Functions ######################################
##########################################################################
def generate_sensors(k = 7, d = 2):
        """
        Generate sensor locations.
        Input:
        k: The number of sensors.
        d: The spatial dimension.
        Output:
        sensor_loc: k * d numpy array.
        """
        sensor_loc = 100*np.random.randn(k,d)
        return sensor_loc

def generate_data(sensor_loc, k = 7, d = 2,
                                  n = 1, original_dist = True):
        """
        Generate the locations of n points.

        Input:
        sensor_loc: k * d numpy array. Location of sensor.
        k: The number of sensors.
        d: The spatial dimension.
        n: The number of points.
        original_dist: Whether the data are generated from the original
        distribution.

        Output:
        obj_loc: n * d numpy array. The location of the n objects.
        distance: n * k numpy array. The distance between object and
        the k sensors.
        """
        assert k, d == sensor_loc.shape

        obj_loc = 100*np.random.randn(n, d)
        if not original_dist:
           obj_loc += 1000

        distance = scipy.spatial.distance.cdist(obj_loc,

           sensor_loc,

           metric='euclidean')
```

```
Generated location:
[ 44.38632327  33.36743274]
Gradient Descent result:
[ 43.07188433  32.71217817]
```