Solutions for HW5

Yunhai Han
Department of Mechanical and Aerospace Engineering
University of California, San Diego
y8han@eng.ucsd.edu

May 21, 2020

Contents

1 Problem 1

1.1 problem formulation

Suppose that action selection is changed to be greedy in SARSA. Is then Q learning the same as SARSA? Will they make the same action selections and weight updates?

1.2 solution

The answer is yes. Because Q-learning is an off-policy TD algorithm, which means it implements an ϵ greedy policy to generate the random trajectory and then a greedy policy is followed for the action selection to update Q values. On the contrary, SARSA is a on-policy algorithm and it directly uses the results from the generated trajectory for the update(no further action selection). If the action selection is changed to be greedy in SARSA, Q learning is the same as SARSA. In the sense, they will makethe same action selections and weight update(they are totally the same indeed).

2 Problem 2

2.1 problem formulation

Provide a pseudo-code for an algorithm that is similar to SARSA, but which makes use of n temporal differences, as opposed to just one.

2.2 solution

Input: A (partially known) Markov Decision Process, an exploration sequence $\{\varepsilon_k\}$

Output: An approximation of $Q_M(i,u)$ for each i and u

1: Initialize $Q_0(i,u), N(i)=0, \alpha_0(i)=1$, for each i, and all u

2: for k in $0,\ldots,M-1$

3: Obtain an ε_{k-} greedy policy from Q_k, call it π_k

4: Generate a T -long episode $\rho_0^{(k)}$ from π_k

5 : for each $(x_t, u_t, x_{t+1}, \cdots, x_{t+n+1}, u_{t+n+1})$ in $\rho_0^{(k)}$

$$
\begin{aligned}
&Q_{k+1}\left(x_t,u_t\right)=Q_k\left(x_t,u_t\right)+\\
&\qquad \alpha_k\left(x_t\right)\left[\sum_{\tau=t}^{t+n}\gamma^{\tau-t}\ell\left(x_\tau,x_{\tau+1}\right)+\gamma^{n+1}Q_k\left(x_{t+n+1},u_{t+n+1}\right)-Q_k\left(x_t,u_t\right)\right]\\
&\qquad N\left(x_t\right)=N\left(x_t\right)+1,\quad \alpha_{k+1}\left(x_t\right)=\left(N\left(x_t\right)+1\right)^{-1}
\end{aligned}
$$

As shown in the pseudo-code above, I make use of n temporal differences instead of just one.

3 Problem 3

3.1 problem formulation

Suppose that we approximate the cost to go J^π of a policy implemented over a one dimensional MDP via least squares and using two feature functions. That is, we take $J^\pi(x) \approx J(x, w) = w\phi_1(x) + \phi_0(x)$ for a state x. In order to find the best w, a training algorithm aims to solve the minimization $\frac{1}{2}E\left(J^\pi(x) - w\phi_1(x) - \phi_0(x)\right)^2$. However, if we don't have a closed form expression $J^\pi(x)$, and, more over, if we lack knowledge about state transition distributions, what we can do is consider approximations of $J^\pi(x)$ at visited points x_i, then instead minimize a function of the form:

$$\min f(w) = \frac{1}{2}\sum_{i=1}^{m}(a_i w - b_i)^2$$

where $a_i \neq 0$ and b_i are two scalars for all i, which come from the evaluation of $\phi_1(x), \phi_0(x)$, at various x_i, and approximations of $J^\pi(x)$ at the x_i. Suppose the coefficients a_i, b_i are given, and let us consider the minimization problem in w. Do the following:

3.2 a

Find the minimum w_i^* of each data-block function $f_i(w) = \frac{1}{2}\left(a_i w - b_i\right)^2$, and the minimum of the least squares problem, w^* and show that w^* lies in the range $R = [\min_i w_i^*, \max_i w_i^*]$.

3.2.1 solution

The range R is $R = [\min_i w_i^*, \max_i w_i^*]$, where $\min_i w_i^* = \min \frac{b_i}{a_i}, i = 1, 2, \cdots, m$ and $\max_i w_i^* = \max \frac{b_i}{a_i}, i = 1, 2, \cdots, m$. The least-square solution w^* can be expressed as:

$$w^* = \frac{\sum_{i=1}^m a_i b_i}{\sum_{i=1}^m a_i^2} = \frac{a_1 b_1 + a_2 b_2 + \cdots + a_m b_m}{a_1^2 + a_2^2 + \cdots + a_m^2}$$

Without loss of generality, I assume $\min_i w_i^*$ is achieved at $i = 1$ and $\max_i w_i^*$ is achieved at $i = m$. Thus, I want to compare w^* with w_1^* and w_m^*.

$$\begin{aligned}
\text{To compare } w_1^* = \frac{b1}{a1} &\text{ and } w^* = \frac{a_1 b_1 + a_2 b_2 + \cdots + a_m b_m}{a_1^2 + a_2^2 + \cdots + a_m^2} \text{it is the same to compare} \\
\frac{b1}{a1}a_1^2 + a_2^2 + \cdots + a_m^2 &\text{ and} a_1 b_1 + a_2 b_2 + \cdots + a_m b_m \text{the same as} \\
b_1 a_1 + \frac{b_1 a_2^2}{a_1} + \cdots + \frac{b_1 a_m^2}{a_1} &\text{ and} a_1 b_1 + a_2 b_2 + \cdots + a_m b_m \text{the same as} \\
\frac{b_1 a_2^2}{a_1} + \cdots + \frac{b_1 a_m^2}{a_1} &\text{ and} a_2 b_2 + \cdots + a_m b_m \text{the same as} \\
\frac{b_1 a_k^2}{a_1} &\text{ and} a_k b_k \text{for each } k = 2, 3, \cdots, m. \text{the same as} \\
\frac{b_1}{a_1} &\text{ and} \frac{b_k}{a_k} \text{for each } k = 2, 3, \cdots, m.
\end{aligned}$$

Based on the assumption that $\frac{b_1}{a_1}$ is the minimum value, it is true that $\frac{b_1}{a_1} \leq \frac{b_k}{a_k}, k = 2, 3, \cdots, m$. Hence, w_1^* is smaller than w^*. Using the same method, w_m^* is larger than w^*. Hence, w^* lies in the range $R = [\min_i w_i^*, \max_i w_i^*]$.

3.3 b

Show how outside the range of R, the gradient $\nabla f_i(w)$ has the same sign as $\nabla f(w)$

3.3.1 solution

The gradient $\nabla f_i(w)$ for each data-block function $f_i(w)$ is $a_i^2 w - a_i b_i$ and the gradient $\nabla f(w)$ is $\sum_{i=1}^m a_i^2 w - \sum_{i=1}^m a_i b_i$. Since w is selected outside the range of R, for example, $w < \frac{b_1}{a_1}$(I assume it is the minimum value), they could be written as:

$$a_i^2 w - a_i b_i < a_i^2 \frac{b_1}{a_1} - a_i b_i$$

The sigh of the expression $a_i^2 \frac{b_1}{a_1} - a_i b_i$ is the same for each $i = 2, 3, \cdots, m$(0 for $i = 1$). The reason is very simple, $a_i^2 \frac{b_1}{a_1} - a_i b_i$ has the same sign as $\frac{b_1}{a_1} - \frac{b_i}{a_i}$($a_i^2 > 0$) and it is definitely smaller than zero(from the assumption). Hence, $a_i^2 w - a_i b_i$ is negative for each $i = 1, 2, \cdots, m$ and the sum of them is still negative($\nabla f(w)$).

Using the same method, if $w > \frac{b_m}{a_m}$(I assume it is the maximum value), the sign is always positive.

3.4 c

Now consider the incremental gradient algorithm given as follows. At each k for an iteration, starting from a given w_0, implement:

$$\begin{aligned} \psi_0 &= w_k \\ \psi_i &= \psi_{i-1} - \beta_k \nabla f_i(\psi_{i-1}), \quad i = 1 \dots m \\ w_{k+1} &= w_k - \beta_k \sum_{i=1}^{m} \nabla f_i(\psi_{i-1}) \end{aligned}$$

What is the difference with the standard gradient method? In practice, what does that mean in practice regarding the computational cost of processing the data set?

3.4.1 solution

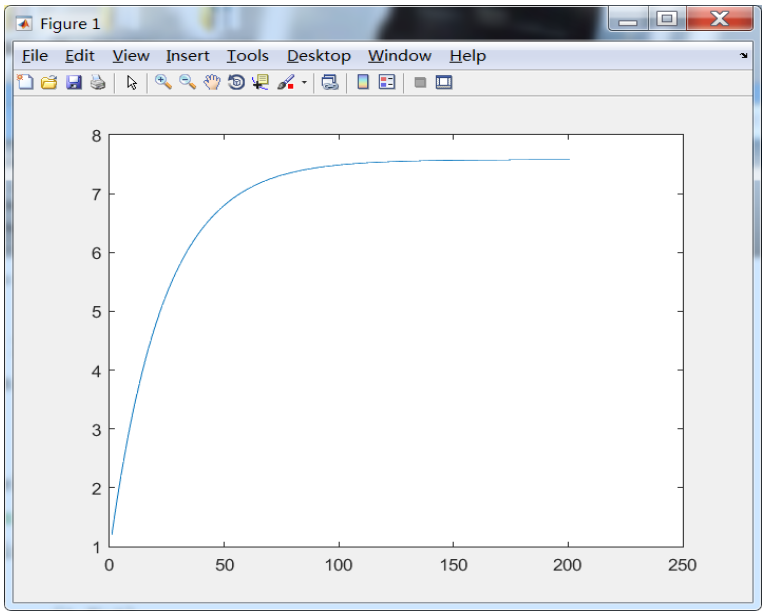
This method is similar to the standard gradient method with the key difference that at each iteration, there are m inner iterations(along each sub-function) for each outer iteration. Hence, the the decision vector is updated incrementally by taking sequential steps in a cyclic order starting from the results of the previous outer iteration. I think in practice, this would lead to computational coefficiency especially in the case of very large datasets. If we update the weights based on every single training sample, it may converge to the local minimum in a zig-zag way instead of a direct way. However, it could converge much faster.

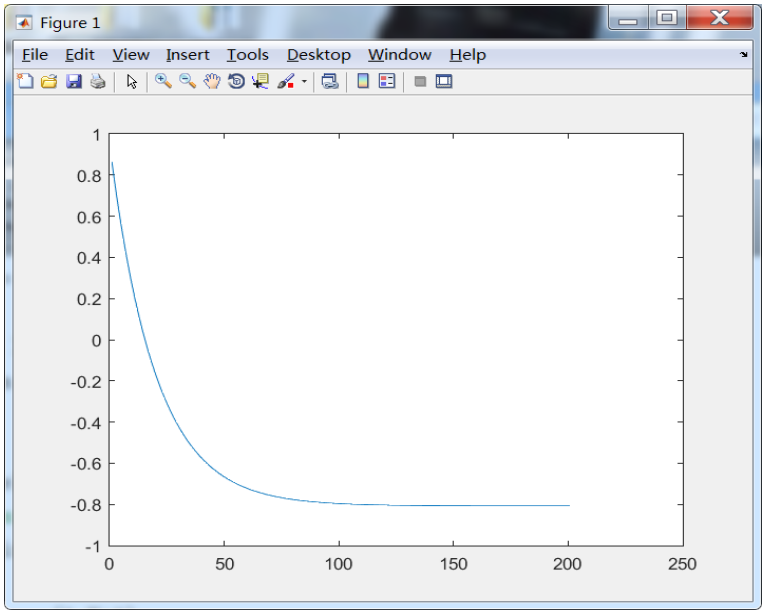
3.5 d

Take a fixed stepsize $\beta_k = \beta < \min_i \frac{1}{a_i^2}$, and implement the previous incremental gradient method for values $m = 10, a_i$ generated randomly over [-10,10] and b_i generated randomly over [-5,5] What do you observe if w_0 is outside R versus if w_0 is inside R? Connect your observations with your answer in part (b). What can you do to improve convergence? Include plots of the trajectories starting from different initial conditions in your solution.

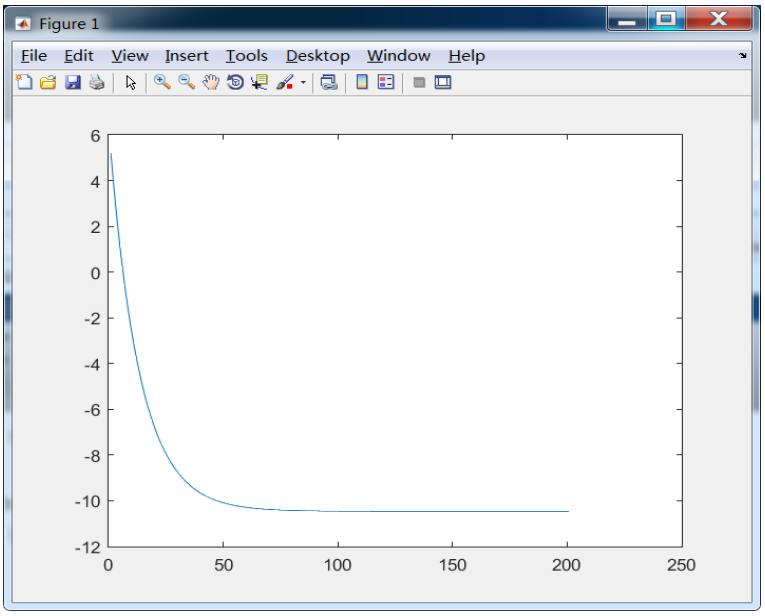
3.5.1 solution

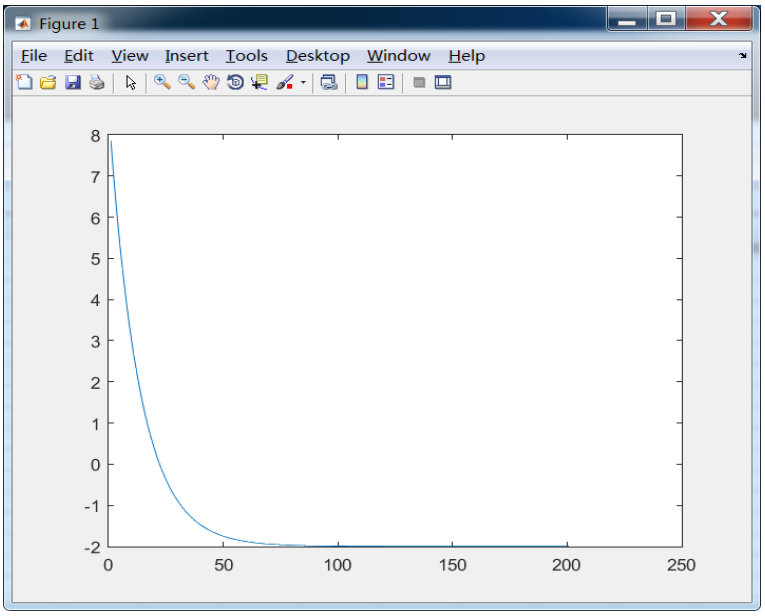
Here I draw two plots of trajectories when w_0 is generated inside the range R.





Then, I draw two plots of trajectories when w_0 is generated outside the range R.





I think when w_0 is generated outside the range R, the gradient $\nabla f_i(w)$ has the same sign as $\nabla f(w)$(during each inner iteration, the changing directions are the same), so the trajectories tends to be steeper at the beginning. As a result, it could converge faster. From the above four figures, my observation is proved.

In order to improve convergence rate, I could adjust the value of β_k. In the previous example, it is set as $0.75 \min_i \frac{1}{a_i^2}$, if I set it as $0.55 \min_i \frac{1}{a_i^2}$, I think it would converge faster.

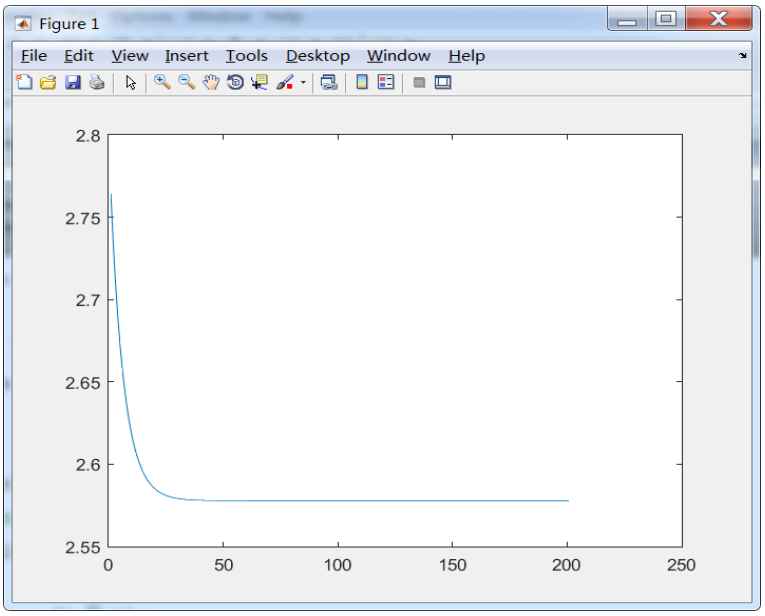


Figure 1
File Edit View Insert Tools Desktop Window Help
2.8
2.75
2.7
2.65
2.6
2.55
0
50
100
150
200
250

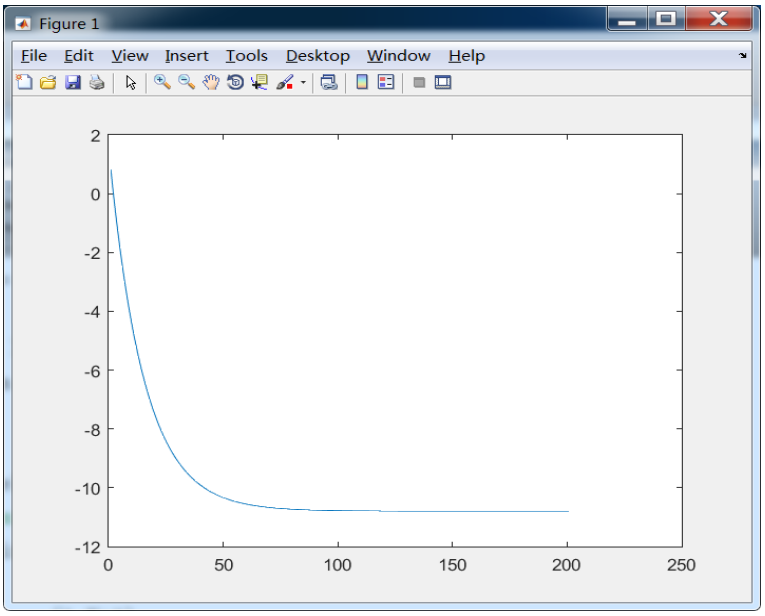


Figure 1
File Edit View Insert Tools Desktop Window Help
2
0
-2
-4
-6
-8
-10
-12
0
50
100
150
200
250

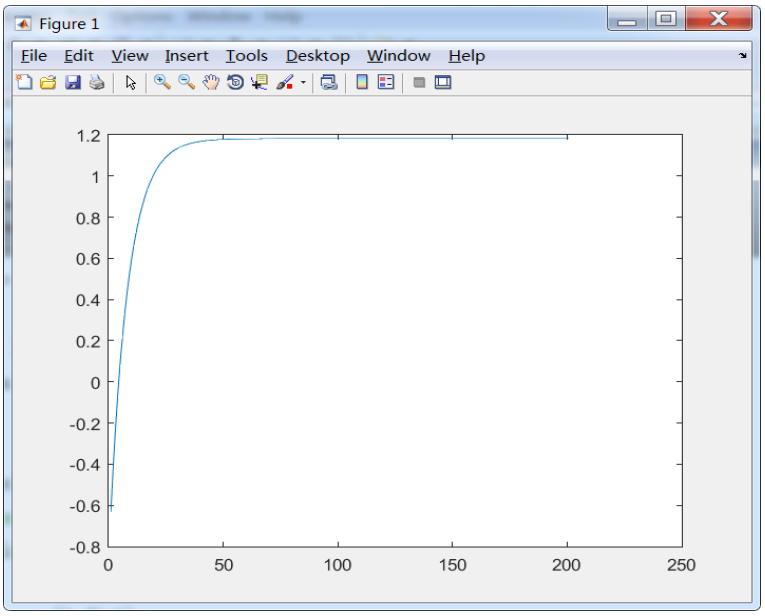


Figure 1
File Edit View Insert Tools Desktop Window Help
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
50
100
150
200
250

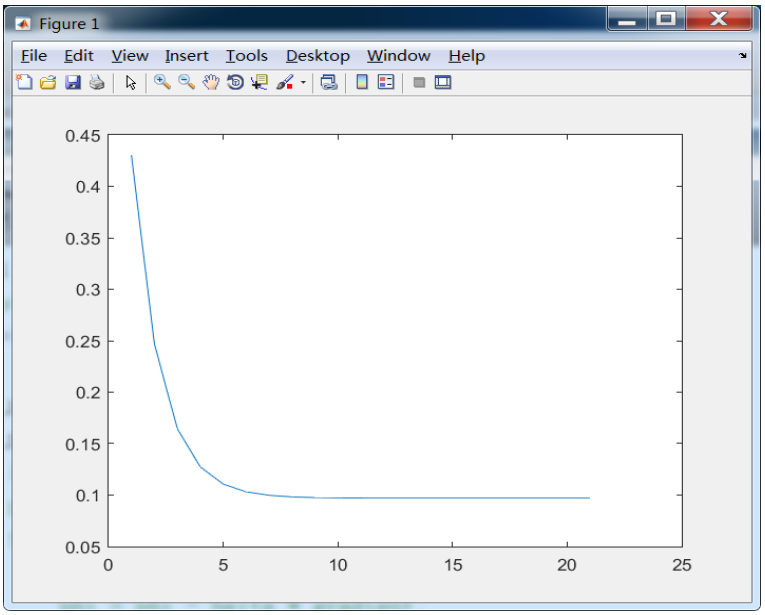
It is obvious that it converges much faster.

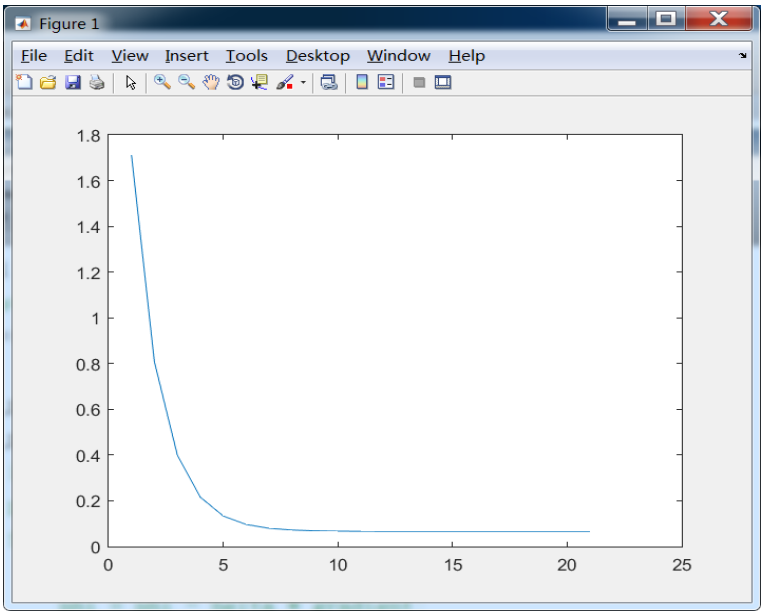
3.6 e

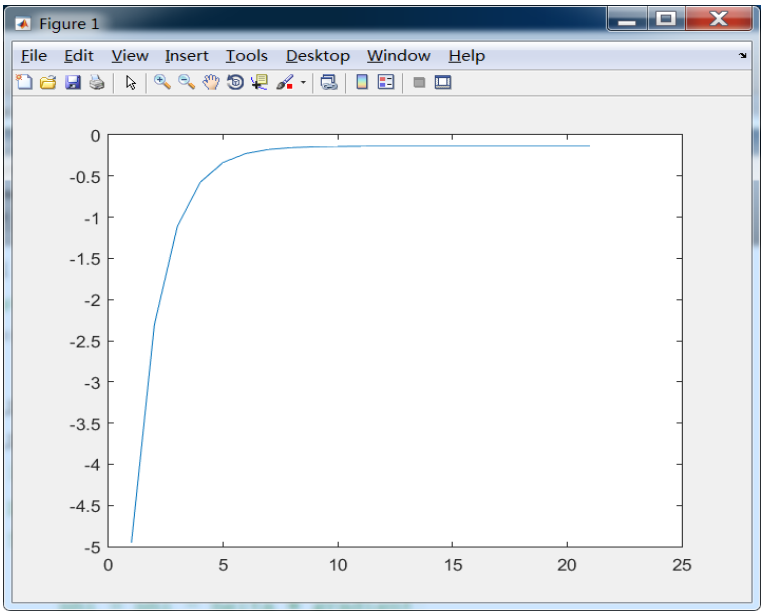
Now implement the standard gradient algorithm for a constant stepsize $\beta_k = \beta < \frac{1}{\sum_i a_i^2}$. What do you observe for trajectories starting at w_0 outside R versus those starting inside R? Include some plots of the trajectories in your solution.

3.6.1 solution

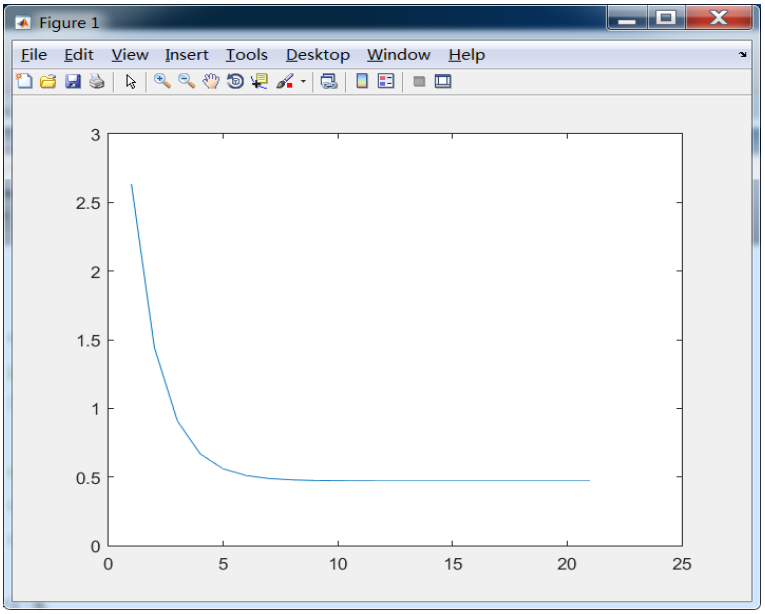
I implement the standard gradient algorithm with a constant setpsize $\beta_k = 0.55\frac{1}{\sum_i a_i^2}$ and I draw some plots here(w_0 generated inside the range R):

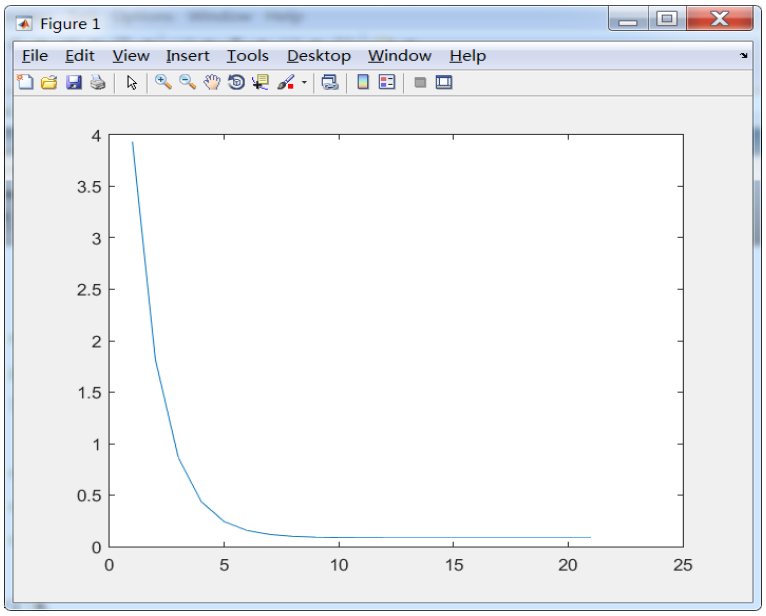


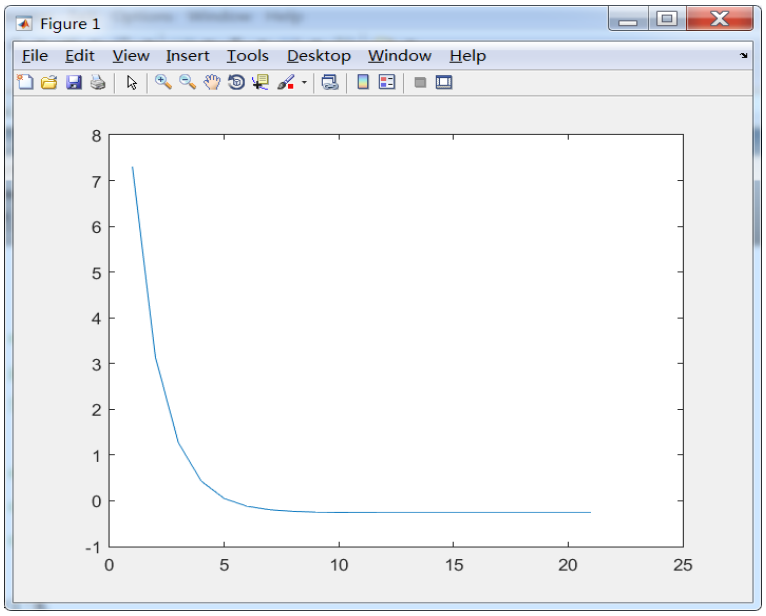




Then, if it is generated outside the range:







It seems that there is no big difference between the convergence rates. I think the reason is that during each iteration, we use all the data in the training set(in this case, $m = 10$), so whether the signs of the gradient of each data-block functions are the same or not does not matter. We combine all the information and could obtain the best approximation. However, in the case of very large dataset, it would result in great computation resources.

3.7 f

From the discussion above, what conclusions can you draw regarding the relative effectiveness of the incremental gradient algorithm versus the standard one?

3.7.1 solution

When the size of training set is not very large, the standard gradient method is preferred for the best approximation results. However, if the dataset is really large(of course, $m = 10$ can not

be considered as ”really large”), we could try the incremental gradient algorithm or stochastic iterative algorithm for the computation efficiency. Based on some online resources, all of these algorithms could guarantee convergence if the objective function has some properties. In the field of big data analysis, incremental gradient algorithms are very popular for these benefits. Besides, there are lots of variants of such algorithm.