Imperial College
London

COURSEWORK1

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Computational Optimization

Author:
Ruizhe Zhao (CID: 01151703)

Date: November 12, 2016

1 Part 1

1.1 Log-Sum-Exp

$\boldsymbol{B}_{jk}$ is the class k's feature j, which is the same as the kth value in $\boldsymbol{\beta}_j$, thus it is convenient to transform the original expression of the Log-Sum-Exp into a function f: [1]

$$f(\boldsymbol{x}) = \log \sum_{k=1}^{10} \exp(\boldsymbol{x}_k) \tag{1}$$

Here $f : \mathbb{R}^{1\times 10} \to \mathbb{R}$ is a function that maps a **row vector** to a scalar value. It is necessary to note that f is just a generalization of the original Log-Sum-Exp expression, as it is easy to illustrate that expression in f:

$$f(\boldsymbol{B}_j) = \log \sum_{k=1}^{10} \exp(\boldsymbol{B}_{jk}) = \log \sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(j)) \tag{2}$$

As mentioned in the problem description, $\boldsymbol{B} \in \mathbb{R}^{n\times 10}$, and $\boldsymbol{B}_j \in \mathbb{R}^{1\times 10}$ represents the j-th row vector in $\boldsymbol{B}$ for $\forall j \in 1, \ldots, n$.
It is a little bit more complicated to prove by line segment definition of convex function. As f is a well defined function and $f \in \mathcal{C}^2$, it is easier to directly use the **second derivative test** for proving convexity.
The first derivative for f over the i-th element in $\boldsymbol{\beta}_j$ is:

$$\begin{aligned}
\frac{\partial f}{\partial \boldsymbol{\beta}_i(j)} &= \frac{\partial \log \sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(j))}{\partial \boldsymbol{\beta}_i(j)} \\
&= \frac{\partial \log \sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(j))}{\partial \sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(j))} \frac{\partial \sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(j))}{\partial \boldsymbol{\beta}_i(j)} \\
&= \frac{\exp(\boldsymbol{\beta}_i(j))}{\sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(j))}
\end{aligned} \tag{3}$$

Thus, the (j,l) element at the Hessian matrix $\boldsymbol{H}(f) \in \mathbb{R}^{10\times 10}$ with given $i \in 1, \ldots, n$ will be:

$$\boldsymbol{H}(f)_{j,l} = \begin{cases} \dfrac{\exp(\boldsymbol{\beta}_j(i)) \exp(\boldsymbol{\beta}_l(i))}{[\sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(i))]^2} & j \neq l \\ \dfrac{\exp(\boldsymbol{\beta}_j(i)) \sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(i)) - \exp(\boldsymbol{\beta}_j(i))^2}{[\sum_{k=1}^{10} \exp(\boldsymbol{\beta}_k(i))]^2} & j = l \end{cases} \tag{4}$$

It is obvious that this Hessian matrix is a real and symmetric matrix, all elements in the matrix is positive (due to the range of exp is $(0, \infty)$). It is also easy to prove that for any vector $\boldsymbol{d} \in \mathbb{R}^{10}$, there is $\boldsymbol{d}^\top \boldsymbol{H}(f) \boldsymbol{d} \geq 0$. Thus $\boldsymbol{H}(f)$ is **positive semi-definite**, and $\boldsymbol{H}(f) \succeq 0$. Hence, by using the second derivative test for convexity, it can be directly proved that f is convex.

[1] f mentioned here is different from the one mentioned in part 2.

1.2 Affine Transformation

Still, before proving the convexity of the expression, it is worthwhile to get its generalization expression, which can be expressed in function h:

$$h_i(\boldsymbol{Y}) = f(\boldsymbol{x}_i^\top \boldsymbol{Y}) \tag{5}$$

Here $\boldsymbol{Y} \in \mathbb{R}^{n\times 10}$ is a fixed size matrix, $\boldsymbol{x}_i \in \mathbb{R}^{1\times n}$ is a constant column vector, and $h_i : \mathbb{R}^{n\times 10} \to \mathbb{R}$ is a mapping from a matrix to a scalar. It is clear that h_i applies an **affine transformation** upon $\boldsymbol{Y}$. If we put $\boldsymbol{B}$ into the domain of h_i, we could prove that $h_i(\boldsymbol{B})$ is consistent with the expression in the problem.

$$h_i(\boldsymbol{B}) = f(\boldsymbol{x}_i^\top \boldsymbol{B}) = \log \sum_{k=1}^{10} \exp(\boldsymbol{x}_i^\top \boldsymbol{\beta}_k) \tag{6}$$

To prove the convexity of this function, it is better to directly apply the defintion of function convexity. For any point in the **line segment** between $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$: [2]

$$\{\alpha \boldsymbol{B}_1 + (1-\alpha)\boldsymbol{B}_2 \mid \forall \alpha \in [0,1]\} \tag{7}$$

With the convexity of f, the convexity of h can be derived as follows:

$$\begin{aligned} h_i(\alpha \boldsymbol{B}_1 + (1-\alpha)\boldsymbol{B}_2) &= f\left(\boldsymbol{x}_i^\top(\alpha \boldsymbol{B}_1 + (1-\alpha)\boldsymbol{B}_2)\right) \\ &= f(\alpha \boldsymbol{x}_i^\top \boldsymbol{B}_1 + (1-\alpha)\boldsymbol{x}_i^\top \boldsymbol{B}_2) \\ &\le \alpha f(\boldsymbol{x}_i^\top \boldsymbol{B}_1) + (1-\alpha) f(\boldsymbol{x}_i^\top \boldsymbol{B}_2) \\ &= \alpha h_i(\boldsymbol{B}_1) + (1-\alpha) h_i(\boldsymbol{B}_2) \end{aligned} \tag{8}$$

Thus, $h_i(\boldsymbol{B}) = \log \sum_{k=1}^{10} \exp(\boldsymbol{x}_i^\top \boldsymbol{\beta}_k)$ is a convex function.

1.3 Another Affine Function

To prove the convexity of function, as $\boldsymbol{B}$ is the single parameter we have and $\boldsymbol{X}, \boldsymbol{Y}$ are fixed as constant, we could still derive the function form of $-\boldsymbol{x}_i^\top \boldsymbol{\beta}_{y_i+1}$ as:

$$g_i(\boldsymbol{B}) = -\boldsymbol{x}_i^\top \boldsymbol{\beta}_{y_i+1} \tag{9}$$

As $i, \boldsymbol{x}_i, \boldsymbol{y}_i$ are all predefined parameters, g_i is indeed the function mentioned in the question. Thus g_i is a generalization of function $-\boldsymbol{x}_i^\top \boldsymbol{\beta}_{y_i+1}$.

The convexity of g_i is pretty easy to be derived. Suppose two matrices $\boldsymbol{B}_1, \boldsymbol{B}_2 \in \mathbb{R}^{n\times 10}$, for a point in the line segment between $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$, $\alpha \in [0,1]$, we have:

$$\begin{aligned} g_i(\alpha \boldsymbol{B}_1 + (1-\alpha)\boldsymbol{B}_2) &= -\boldsymbol{x}_i^\top(\alpha \boldsymbol{\beta}_{y_i+1}^{(1)} + (1-\alpha)\boldsymbol{\beta}_{y_i+1}^{(2)}) \\ &= -\alpha \boldsymbol{x}_i^\top \boldsymbol{\beta}_{y_i+1}^{(1)} - (1-\alpha)\boldsymbol{x}_i^\top \boldsymbol{\beta}_{y_i+1}^{(2)} \\ &\le \alpha g_i(\boldsymbol{B}_1) + (1-\alpha) g_i(\boldsymbol{B}_2) \end{aligned} \tag{10}$$

[2]It is a little bit weird to define the line segment between two matrices rather than two vectors, but as this line segment expression is a valid algebra expression, and we can always reshape a matrix into a vector, it is sensible to still use the term ”line segment”.

Thus, g_i is convex, and further we have $-\boldsymbol{x}_i^\top \boldsymbol{\beta}_{y_i+1}$ is convex.

1.4 Regularisation

For any vector $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \in \mathbb{R}^{n\times 1}$, and value $\alpha \in [0,1]$, we have:

$$\begin{aligned} \left\|\alpha\boldsymbol{\beta}_1 + (1-\alpha)\boldsymbol{\beta}_2\right\|_1 &= \sum_{i=1}^{n} |\alpha\boldsymbol{\beta}_1(i) + (1-\alpha)\boldsymbol{\beta}_2(i)| \\ &\leq \sum_{i=1}^{n} \alpha|\boldsymbol{\beta}_1(i)| + \sum_{i=1}^{n}(1-\alpha)|\boldsymbol{\beta}_2(i)| \\ &= \alpha\left\|\boldsymbol{\beta}_1\right\|_1 + (1-\alpha)\left\|\boldsymbol{\beta}_2\right\|_1 \end{aligned} \tag{11}$$

We have applied the triangle inequality to derive the equation above. According to the definition of convexity, this regularisation function is a convex function.

1.5 Entire problem

The entire optimisation problem is defined with an objective function $g(\boldsymbol{B}) + h(\boldsymbol{B})$, and a constraint $\mathbb{R}^{n\times 10}$ for parameter $\boldsymbol{B}$. As $\mathbb{R}^{n\times 10}$ is indeed a convex set, only the convexity of the objective function is required to be proved.
$g(\boldsymbol{B}) + h(\boldsymbol{B})$ is a convex function is merely a combination of 3 convex functions:

$$\begin{aligned} g(\boldsymbol{B}) + h(\boldsymbol{B}) &= \sum_{i=1}^{m}\left(\log\sum_{k=1}^{10}\exp(\boldsymbol{x}_i^\top\boldsymbol{\beta}_k) - \boldsymbol{x}_i^\top\boldsymbol{\beta}_{y_i+1}\right) + \lambda\sum_{k=1}^{10}\left\|\boldsymbol{\beta}_k\right\|_1 \\ &= \sum_{i=1}^{m}(h_i(\boldsymbol{B}) + g_i(\boldsymbol{B})) + \lambda\sum_{k=1}^{10}\left\|\boldsymbol{\beta}_k\right\|_1 \end{aligned} \tag{12}$$

It has been proved that for $\forall i \in \{1,...,m\}$, h_i and g_i are both convex. Also each regularisation term is also a convex function. Thus $g(\boldsymbol{B}) + h(\boldsymbol{B})$ is sum of convex functions, and itself is absolutely a convex function. And this entire problem is a convex program.

2 Part 2

2.1 L1 Regularisation

According to the definition of the L1 regularisation function, it is reasonable to discuss the continuity and differentiability of h on the element level of $\boldsymbol{B}$, as h is merely a linear combination of all $\boldsymbol{B}$ elements' absolute value.
For $\forall i \in \{1,...,n\}, j \in \{1,...,10\}$, it is obvious that the **partial derivative** of h over $\boldsymbol{B}_{i,j}$ is 1 when $\boldsymbol{B}_{i,j} > 0$, and -1 when $\boldsymbol{B}_{i,j} < 0$. When $\boldsymbol{B}_{i,j}$ takes 0, the left and right partial derivatives of h over $\boldsymbol{B}_{i,j}$ are not equal:

$$\begin{aligned}
\frac{\overleftarrow{\partial} h}{\partial \boldsymbol{B}_{i,j}}(\boldsymbol{B}) &= \lim_{\alpha\to 0^+} \frac{h(\ldots,\boldsymbol{B}_{i,j}+\alpha,\ldots)-h(\boldsymbol{B})}{\alpha} \overset{\boldsymbol{B}_{i,j}=0}{=\!=\!=} \lim_{\alpha\to 0^+}\frac{\alpha}{\alpha} = 1 \\
\frac{\overrightarrow{\partial} h}{\partial \boldsymbol{B}_{i,j}}(\boldsymbol{B}) &= \lim_{\alpha\to 0^-} \frac{h(\ldots,\boldsymbol{B}_{i,j}+\alpha,\ldots)-h(\boldsymbol{B})}{\alpha} \overset{\boldsymbol{B}_{i,j}=0}{=\!=\!=} \lim_{\alpha\to 0^-}\frac{-\alpha}{\alpha} = -1
\end{aligned} \tag{13}$$

However, the continuity keeps at $\boldsymbol{B}_{i,j} = 0$ as the left and right limits are equal:

$$\lim_{\boldsymbol{B}_{i,j}\to 0^+} h(\boldsymbol{B}) = \lim_{\boldsymbol{B}_{i,j}\to 0^-} h(\boldsymbol{B}) = \sum_{k=1}^{10}\sum_{l=1}^{n} |\boldsymbol{B}_{l,k}| \qquad \text{when } l \neq i \text{ and } k \neq j$$

In a word, h is **continuously partial differentiable** for all $\boldsymbol{B}$ elements when they don't take value at 0, and h is **only** continuous at 0. So $h \notin \mathcal{C}^1$.

2.2 L2 Regularisation

Convexity

According to the discussion in part 1, that the first term of f is a convex function. For the new L2 regularisation term, it is also a convex function which will be proved in the following.
Let h denote the L2 regularisation function, $\forall \boldsymbol{B}_1, \boldsymbol{B}_2 \in \mathbb{R}^{n,10}$, and for any point within the line segment between $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ - $\alpha\boldsymbol{B}_1 + (1-\alpha)\boldsymbol{B}_2$ for $\forall\alpha \in [0,1]$, we have:

$$\begin{aligned}
h(\alpha\boldsymbol{B}_1 + (1-\alpha)\boldsymbol{B}_2) &= \lambda\sum_{k=1}^{10}\left\|\alpha\beta_k^{(1)} + (1-\alpha)\beta_k^{(2)}\right\|_2^2 \\
&\leq \lambda\sum_{k=1}^{10}\left\|\alpha\beta_k^{(1)}\right\|_2^2 + \lambda\sum_{k=1}^{10}\left\|(1-\alpha)\beta_k^{(2)}\right\|_2^2 && \text{(Minkowski Inequality)} \\
&= \lambda\sum_{k=1}^{10}\alpha^2\left\|\beta_k^{(1)}\right\|_2^2 + \lambda\sum_{k=1}^{10}(1-\alpha)^2\left\|\beta_k^{(2)}\right\|_2^2 \\
&\leq \alpha\lambda\sum_{k=1}^{10}\left\|\beta_k^{(1)}\right\|_2^2 + (1-\alpha)\lambda\sum_{k=1}^{10}\left\|\beta_k^{(2)}\right\|_2^2 && (\alpha \leq 1) \\
&= \alpha h(\boldsymbol{B}_1) + (1-\alpha)h(\boldsymbol{B}_2)
\end{aligned}$$

During the derivation, we utilise the **Minkowski Inequality** ($\left\|\boldsymbol{x}+\boldsymbol{y}\right\|_q \leq \|\boldsymbol{x}\|_q + \left\|\boldsymbol{y}\right\|_q$ where q is the degree of the normalization function), and also the $\alpha \leq 1$ condition. Thus, we finally have $h(\alpha\boldsymbol{B}_1 + (1-\alpha)\boldsymbol{B}_2) \leq \alpha h(\boldsymbol{B}_1) + (1-\alpha)h(\boldsymbol{B}_2)$, which can further prove that h is a convex function.
As mentioned above, g is a convex function, hence f, which is the sum of two convex functions g and h, is indeed a convex function.

Continuously Differentiable

As mentioned in the problem description, and it is easy to prove that $g(\boldsymbol{B}) \in \mathcal{C}^1$, so the remaining task to prove $f(\boldsymbol{B}) \in \mathcal{C}^1$ is proving $h(\boldsymbol{B}) \in \mathcal{C}^1$, as a function is continuously differentiable if it is a sum of two continuous differentiable functions.
$h(\boldsymbol{B})$ is indeed a continuously differentiable function, as it is merely a squared sum of all elements in $\boldsymbol{B}$. It is obvious that the gradient for h is:

$$\frac{\partial h(\boldsymbol{B})}{\partial \boldsymbol{B}} = 2\lambda \boldsymbol{B} \tag{14}$$

It is also a continuous function. Thus $h(\boldsymbol{B}) \in \mathcal{C}^1$, and then $f(\boldsymbol{B}) \in \mathcal{C}^1$.

2.3 Conditions Implementation

Remember to fix these line numbers once the code is freezed.

Please see the code at line 60, 65 & 70 in `SolveMNIST_Gradient.m`.

2.4 Conditions Explanation

Among the 3 optimality conditions, **only** FONC is relevant to the tolerant conditions, SONC and SOSC are irrelevant.

Why FONC is related?

The first tolerant condition (line 60) corresponds to the FONC condition. As the problem is an unconstrained optimisation problem ($\boldsymbol{B} \in \mathbb{R}^{n\times 10}$), only the **interior point** version of FONC is necessary to be stated here, which requires the **first derivative** $\nabla_{\boldsymbol{B}} f(\boldsymbol{B})$ equals $\mathbf{0}$.
All the 3 conditions are related to FONC, as they are all targeting at checking whether the gradient is close enough to $\mathbf{0}$.

1. The first condition $\left\|\nabla_{\boldsymbol{B}} f(\boldsymbol{B}^{(j)})\right\|_2 < \epsilon$ checks whether the **norm** of the current step (j) gradient $\nabla_{\boldsymbol{B}} f(\boldsymbol{B})$ is close enough to zero, if so, then all the elements in the gradient are all close enough to zero.

2. The second condition $\left\|\boldsymbol{B}^{(j+1)} - \boldsymbol{B}^{(j)}\right\|_2 < \epsilon$ checks whether two adjacent points are close enough to each other. If so, then the gradient should also be close enough to $\mathbf{0}$, which can be easily derived from the steepest descent **iterative algorithm** definition: $\boldsymbol{B}^{(j+1)} - \boldsymbol{B}^{(j)} = \alpha^{(j)} \nabla_{\boldsymbol{B}} f(\boldsymbol{B}^{(j)})$.

3. The third condition $\left|f(\boldsymbol{B}^{(j+1)}) - f(\boldsymbol{B}^{(j)})\right| < \epsilon$ checks whether two adjacent steps has close enough function values. If so, then the gradient should also be close enough to $\mathbf{0}$, because the gradient **measures the difference** between two function values on adjacent points (matrices).

Why SONC and SOSC are irrelevant?

For a given interior point $\boldsymbol{B}$ that already satisfies FONC, if it further satisfies SONC, then f must be a *second continuously differentiable* function according to the SONC definition, and the Hessian matrix $\nabla^2_{\boldsymbol{B}} f(\boldsymbol{B})$ must be **positive semi-definite**; If it satisfies SOSC, then the Hessian matrix $\nabla^2_{\boldsymbol{B}} f(\boldsymbol{B})$ must be **positive definite**. Unfortunately, the last two conditions have **no second order information**: The Hessian matrix, which is the key part of SONC and SOSC, is not calculated in either condition.

2.5 Constant Step Size Strategy

Here is the gradient descent plot (Fig. 1). The configuration parameters for the algorithm remains the same as the question description.

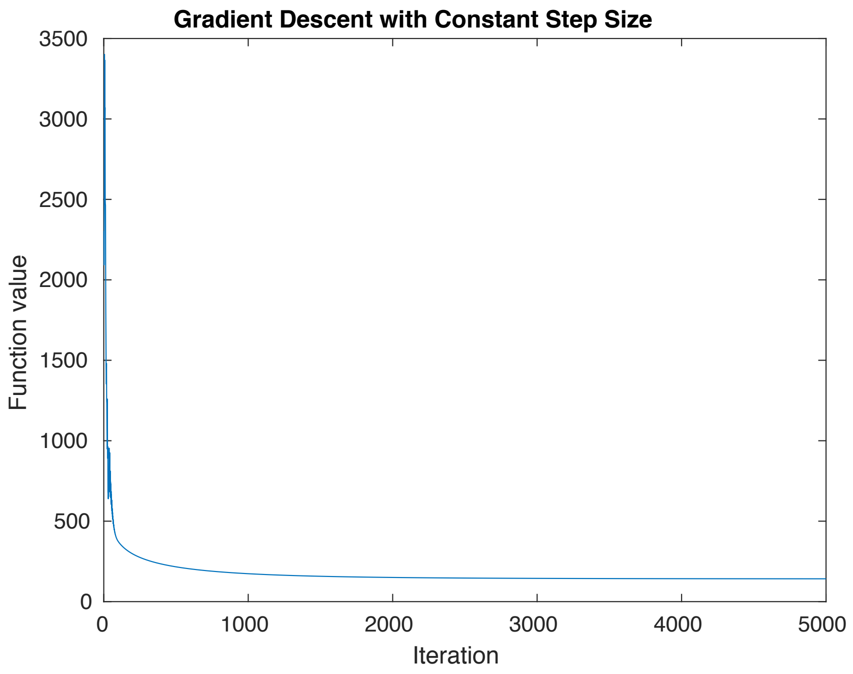

Figure 1: Plot for gradient descent of the given function f with constant step size strategy. Step size is initiated as 0.0001 and the maximum number of iteration is 5000.

It is worth to note that at the ending iteration step 5000, neither of the 3 tolerance conditions is satisfied. So the final step is still not the convergence step.

2.6 Exact Line Search Strategy

The exact line search problem lies in this problem is the optimisation of the **step size** variable $\alpha^{(k)}$ for any step k. With given k-th step's value $x^{(k)}$, the target of the optimisation is to **minimise** the next step's value $x^{(k+1)}$ w.r.t $\alpha^{(k)}$, which is equivalent to **maximising** the improvement of the current step. As $\alpha^{(k)}$ is a **scalar** (one dimen-

sional variable), and so for the function value, it is possible for us to use the exact line search strategy to find the minimiser.
The line search optimisation problem can be formalised in the following. For any step k, and with $\phi_k(\alpha) \overset{\Delta}{=} f(\boldsymbol{B}^{(k)} - \alpha \nabla f(\boldsymbol{B}^{(k)}))$, we have:

$$\begin{aligned} \arg\min_{\alpha} \ & \phi_k(\alpha) \\ \textbf{s.t.} \ & \alpha \geq 0 \end{aligned} \tag{15}$$

Here $\nabla f(\boldsymbol{B}^{(k)})$ is the **search direction** of the line search algorithm.
There are several exact line search method (or one dimensional method) can be applied, here we briefly justify whether each method is suitable for the given function $\phi_k(\alpha)$:

1. **Golden Section Search** is *suitable* for ϕ_k, as $\phi_k : \mathbb{R} \to \mathbb{R}$, and it is **unimodal**. The unimodal property is easy to prove: As f is a convex function and ϕ_k takes along one single line of f, ϕ_k is definitely a convex function, which further indicates that ϕ_k is unimodal as the local minimiser is equivalent to the global minimiser.

2. **Newton Method** is also a proper choice here. $\phi_k \in \mathcal{C}^2$ and [3]:

$$\phi_k'' = \nabla f(\boldsymbol{B}^{(k)})^\top \nabla^2 f(\boldsymbol{B}^{(k+1)}) \nabla f(\boldsymbol{B}^{(k)}) \tag{16}$$

   As f is a convex function, the Hessian matrix $\nabla^2 f(\boldsymbol{B}^{(k+1)})$ is positive semi-definite, which means ϕ_k'' is always non-negative, which can make sure that the newton method is able to work well.

3. **Secant Method** could also work well according to similar reason mentioned in the **Newton Method**.

2.7 Implementation of Line Search

In this section, we will see 3 line search methods implementation, including: **Golden Section Search**, **Secant**, and **Backtracking**[4]. With experiment data and plotting, we could claim that these line search methods are better than the constant step size method in terms of **convergence rate**, but suffer from **longer run time per iteration**.[5] We will justify this first by observing descent curves for all methods in a small region, and then enlarge the maximum iteration number until all methods converge to find out their general performance.
Fig. 2 shows different converge curves by using different line search strategies, with a maximum number of iterations 100. From this figure, we could find out that the

[3]For the dimensions in this equation, it would be easier to reason if we reshape the matrix to a vector.

[4]There are 3 MATLAB files related to these methods: `golden_section_search.m`, `secant.m`, and `backtracking.m`.

[5]The code to run the experiment lies in `run_mnist_gradient.m`, and please refer to the `README.md` file for instructions to **reproduce** the experiments.

constant step size method acts quite ”unstable” before iteration number 70. The Golden Section Search and Secant Method show almost the same curves, as they both give the exact line search result for each iteration, so they will have almost the same descent path. The backtracking method, however, *seems* to perform the best among all these methods (We will challenge this argument when we show the general data), as it has the **steepest decrease** on function value.

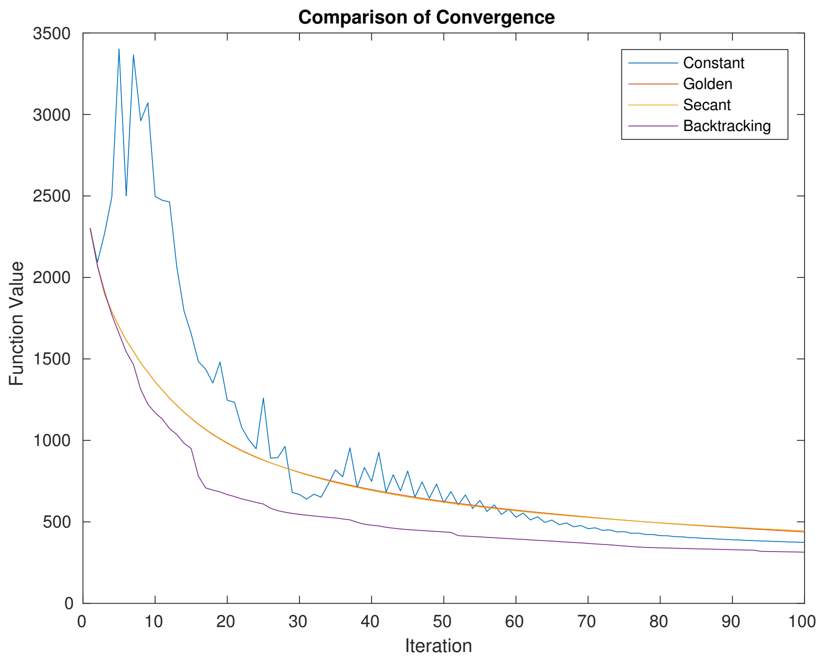

Figure 2: Plot for gradient descent of the given function by using different line search methods, the maximum iteration number is 100.

Although backtracking looks good in the figure above, we might have different opinion based on a more general study with maximum number of iterations as 10000 (See Table. 1. There are three key observations from this table:

1. Golden section search has the least number of iterations, and constant step size strategy has the most. Secant and backtracking has similar number of iterations. In terms of number of iterations and the **convergence rate**, golden section search is the best.

2. Backtracking is the slowest strategy in terms of total run time, and still the golden section search strategy is the fastest.

3. In terms of time per iteration, constant strategy is the fastest as there is no additional computation to decide the best step size. Other three methods have similar time per iteration. However, this metric is not quite important if the constant strategy has much slower convergence rate than others.

As a conclusion, there is no doubt that the constant strategy is the worst among all strategies, as it converges really slow in total. Other adaptive method performs much better than the constant strategy in terms of convergence rate, however, as different method has different computation cost in deciding the best step size, the total run time might be slower than the constant strategy.

	Total Iterations	Total Time (s)	Time Per Iteration (s)
Constant	7084	1442.4	0.20362
Golden	1505	1138.5	0.75645
Secant	2375	1492.4	0.62839
Backtracking	2342	1636.1	0.69857

Table 1: Comparison on total number of iterations, total time, and time per iteration for these 4 different step size strategies.

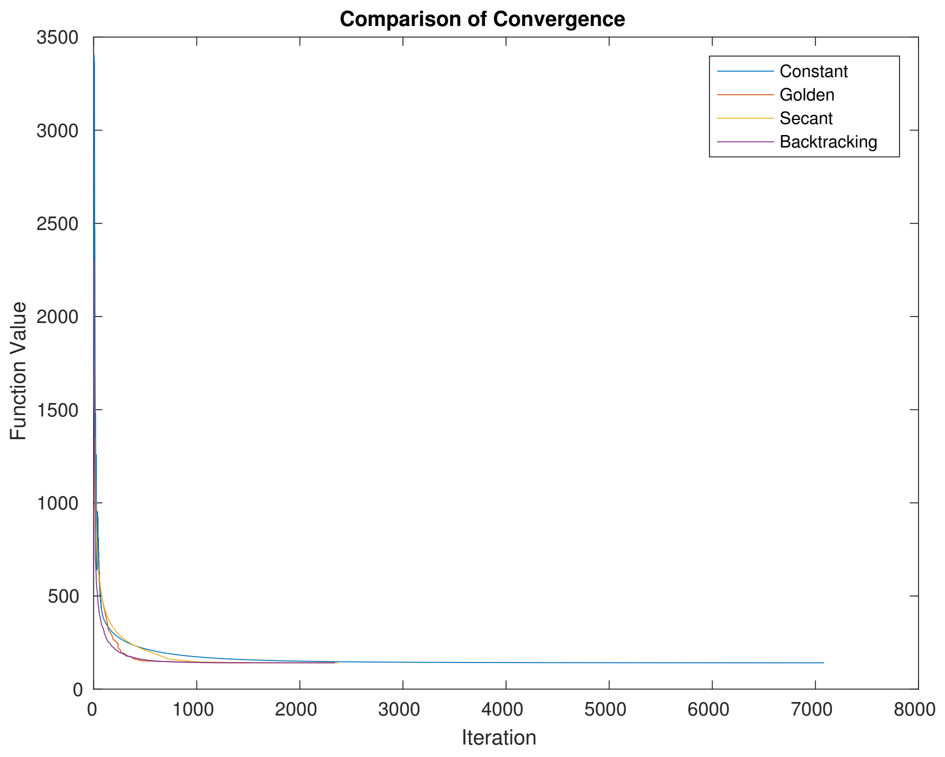

Figure 3: Plot for gradient descent of the given function by using different line search methods, the maximum iteration number is 10000.