دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

تمرین شماره: ۰۲

بهینه‌سازی پیشرفته

نام و نام‌خانوادگی: سیدرضا مسلمی

شماره دانشجویی: ۸۱۰۱۰۳۳۲۶

پاییز ۱۴۰۳

فهرست مطالب

فهرست تصاویر

فهرست جداول

سوال ۱

عنوان سوال یک

۱.۱ مقدمه

تابع درجه دوم و هموار $f(x)$ به صورت زیر تعریف شده است:

$$f(x) = \frac{1}{2}x^T Qx + c^T x + r,$$

که در آن Q یک ماتریس متقارن مثبت معین است ($Q \succ 0$).

هدف این است که نشان دهیم:

$$h(\sigma) = f(x_0 + \sigma_0 p_0 + \sigma_1 p_1 + \cdots + \sigma_{k-1} p_{k-1}),$$

وقتی

$$\sigma = (\sigma_0, \sigma_1, \dots, \sigma_{k-1})$$

۲.۱ اثبات

۱.۲.۱ بازنویسی $h(\sigma)$ در قالب σ

پارامتر ورودی تابع f به صورت زیر است:

$$x = x_0 + \sum_{i=0}^{k-1} \sigma_i p_i.$$

با جایگذاری این عبارت در $f(x)$، خواهیم داشت:

$$h(\sigma) = f\left(x_0 + \sum_{i=0}^{k-1} \sigma_i p_i\right).$$

۲.۲.۱ جایگذاری در قالب درجه دوم از $f(x)$

$$h(\sigma) = \frac{1}{2}\left(x_0 + \sum_{i=0}^{k-1} \sigma_i p_i\right)^T Q \left(x_0 + \sum_{i=0}^{k-1} \sigma_i p_i\right) + c^T \left(x_0 + \sum_{i=0}^{k-1} \sigma_i p_i\right) + r.$$

۳.۲.۱ گسترش تابع $h(\sigma)$

گسترش عبارت درجه دوم به صورت زیر است:

$$\left(x_0 + \sum_{i=0}^{k-1} \sigma_i p_i\right)^T Q \left(x_0 + \sum_{i=0}^{k-1} \sigma_i p_i\right) = x_0^T Q x_0 + 2x_0^T Q \left(\sum_{i=0}^{k-1} \sigma_i p_i\right) + \left(\sum_{i=0}^{k-1} \sigma_i p_i\right)^T Q \left(\sum_{i=0}^{k-1} \sigma_i p_i\right).$$

عبارت آخر به صورت زیر گسترش می‌یابد:

$$\left(\sum_{i=0}^{k-1} \sigma_i p_i\right)^T Q \left(\sum_{j=0}^{k-1} \sigma_j p_j\right) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} \sigma_i \sigma_j p_i^T Q p_j.$$

عبارت خطی نیز به این صورت است:

$$c^T \left(x_0 + \sum_{i=0}^{k-1} \sigma_i p_i \right) = c^T x_0 + \sum_{i=0}^{k-1} \sigma_i c^T p_i.$$

بنابراین، تابع $h(\sigma)$ به شکل زیر خواهد بود:

$$h(\sigma) = \frac{1}{2} x_0^T Q x_0 + x_0^T Q \left(\sum_{i=0}^{k-1} \sigma_i p_i \right) + \frac{1}{2} \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} \sigma_i \sigma_j p_i^T Q p_j + c^T x_0 + \sum_{i=0}^{k-1} \sigma_i c^T p_i + r.$$

۴.۲.۱ گروه‌بندی عبارت‌ها در σ

عبارت‌های ثابت عبارتند از:

$$\frac{1}{2} x_0^T Q x_0 + c^T x_0 + r.$$

عبارت‌های خطی در σ به صورت زیر هستند:

$$\sum_{i=0}^{k-1} \sigma_i \left(x_0^T Q p_i + c^T p_i \right).$$

عبارت‌های درجه دوم در σ به این صورت خواهند بود:

$$\frac{1}{2} \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} \sigma_i \sigma_j p_i^T Q p_j.$$

بنابراین، $h(\sigma)$ به شکل زیر نوشته می‌شود:

$$h(\sigma) = \frac{1}{2} \sigma^T A \sigma + b^T \sigma + \text{const},$$

که در آن:

$$A_{ij} = p_i^T Q p_j, \quad b_i = x_0^T Q p_i + c^T p_i.$$

۵.۲.۱ نشان دادن هموار و محدب بودن $h(\sigma)$

- ماتریس A متقارن است زیرا Q متقارن است:

$$A_{ij} = p_i^T Q p_j = p_j^T Q p_i = A_{ji}.$$

- ماتریس A مثبت معین است زیرا $Q \succ 0$. برای هر σ غیرصفر،

$$\sigma^T A \sigma = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} \sigma_i \sigma_j p_i^T Q p_j = \left(\sum_{i=0}^{k-1} \sigma_i p_i \right)^T Q \left(\sum_{j=0}^{k-1} \sigma_j p_j \right).$$

از آنجا که $Q \succ 0$، این فرم درجه دوم برای هر $\sum_{i=0}^{k-1} \sigma_i p_i$ غیرصفر به صورت مثبت خواهد بود. بنابراین، $A \succ 0$ است که به این معناست که تابع $h(\sigma)$ هموار و محدب است.

سوال ۲

عنوان سوال دو

۱.۲ مقدمه

۱.۱.۲ Conjugate Vectors:

مجموعه‌ای از بردارهای $\{p_0, p_1, \dots, p_l\}$ با توجه به ماتریس A، conjugate است اگر:

$$p_i^T A p_j = 0, \quad \text{برای تمام } i \neq j.$$

۲.۱.۲ استقلال خطی (Linear Independence):

مجموعه $\{p_0, p_1, \dots, p_l\}$ مستقل خطی است اگر پاسخ معادله:

$$\sum_{i=0}^{l} \alpha_i p_i = 0,$$

این باشد که:

$$\alpha_0 = \alpha_1 = \cdots = \alpha_l = 0$$

۲.۲ اثبات

۱.۲.۲ فرض کردن مستقل بردارها

فرض میکنیم که مجموعه $\{p_0, p_1, \ldots, p_l\}$ مستقل خطی نیست. در این صورت، مقادیر اسکالر $\{\alpha_0, \alpha_1, \ldots, \alpha_l\}$ وجود دارند که همه‌ی آن‌ها صفر نیست و ترکیب خطی زیر برابر صفر است:

$$\sum_{i=0}^{l} \alpha_i p_i = 0.$$

فرض می‌کنیم $v = \sum_{i=0}^{l} \alpha_i p_i$. طبق فرض $v = 0$ و هیچ‌کدام از α_i صفر نیستند.

۲.۲.۲ محاسبه $v^T A v$

از آنجا که A یک ماتریس مثبت معین و متقارن است، فرم درجه دوم $v^T A v$ همیشه غیرمنفی و برای $v \neq 0$ همیشه مثبت است. با قرار دادن $v = \sum_{i=0}^{l} \alpha_i p_i$، خواهیم داشت:

$$v^T A v = \left(\sum_{i=0}^{l} \alpha_i p_i \right)^T A \left(\sum_{j=0}^{l} \alpha_j p_j \right).$$

این را گسترش می‌دهیم:

$$v^T A v = \sum_{i=0}^{l} \sum_{j=0}^{l} \alpha_i \alpha_j p_i^T A p_j.$$

۳.۲.۲ استفاده از ویژگی conjugate بودن

طبق تعریف بردارهای conjugate، $p_i^T A p_j = 0$ برای $i \neq j$. بنابراین، عبارت به این صورت ساده می‌شود:

$$v^T A v = \sum_{i=0}^{l} \alpha_i^2 p_i^T A p_i.$$

از آنجا که A مثبت معین است و $p_i \neq 0$، می‌دانیم که $p_i^T A p_i > 0$ برای هر i. بنابراین:

$$v^T A v = \sum_{i=0}^{l} \alpha_i^2 p_i^T A p_i > 0,$$

به شرطی که حداقل یکی از α_iها صفر نباشد.

۴.۲.۲ تناقض

اگر $v = 0$، آنگاه $v^T A v = 0$. اما از محاسبات قبلی داریم که $v^T A v > 0$ چون حداقل یکی از α_iها صفر نیست و $p_i^T A p_i > 0$. این یک تناقض است.

سوال ۳

عنوان سوال سه

۱.۳ مقدمه

برای روش گرادیان مزدوج (CG) که معادله $Ax = b$ را حل می‌کند، نابرابری زیر را باید اثبات کنیم:

$$f(x_k) \leq \left(\frac{b-a}{b+a}\right)^k f(x_0),$$

تحت شرایطی که A یک ماتریس تقارن‌دار مثبت معین است و مقادیر ویژه آن در بازه $[a, b]$ قرار دارد .($a, b > 0$)

۲.۳ اثبات

۱.۲.۳ رفتار بقایای روش گرادیان مزدوج

روش گرادیان مزدوج برای کمینه‌سازی تابع مربعی زیر طراحی شده است:

$$f(x) = \frac{1}{2}x^T Ax - b^T x,$$

که در آن داریم:

$$f(x_k) = \frac{1}{2} r_k^T A^{-1} r_k,$$

که در آن $r_k = b - Ax_k$ باقی‌مانده در گام k است.

خطای گام k در روش CG به صورت زیر محدود می‌شود:

$$\|e_k\|_A \leq \|e_0\|_A \cdot \frac{\kappa(A) - 1}{\kappa(A) + 1},$$

که در آن:

- $e_k = x_k - x^*$ است،
- $\|e_k\|_A = \sqrt{e_k^T A e_k}$ است،
- $\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$ عدد شرطی ماتریس A است.

در اینجا، $\lambda_{\max} \leq b$ و $\lambda_{\min} \geq a$.

۲.۲.۳ محدود کردن نرمال انرژی

برای روش CG، نرمال انرژی خطا به شکل زیر محدود می‌شود:

$$\|e_k\|_A^2 \leq \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^{2k} \|e_0\|_A^2.$$

با استفاده از $\kappa(A) = \frac{b}{a}$، داریم:

$$\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} = \frac{\sqrt{b/a} - 1}{\sqrt{b/a} + 1}.$$

این کسر را ساده می‌کنیم:

$$\frac{\sqrt{b/a} - 1}{\sqrt{b/a} + 1} = \frac{b - a}{b + a}.$$

بنابراین:

$$\|e_k\|_A^2 \leq \left(\frac{b-a}{b+a}\right)^{2k} \|e_0\|_A^2.$$

۳.۲.۳ کاهش مقدار تابع

چون $f(x_k)$ متناسب با $\|e_k\|_A^2$ است، نتیجه می‌گیریم که:

$$f(x_k) \leq \left(\frac{b-a}{b+a}\right)^{k} f(x_0).$$

سوال ۴

عنوان سوال چهار

۱.۴ مقدمه

معادله زیر به ما داده شده است:

$$(J^T J + \lambda I)p = -J^T r$$

که در آن:

- $J \in R^{m\times n}$ یک ماتریس ژاکوبین است،
- $r \in R^m$ یک بردار است،
- $p \in R^n$ بردار جهت است،
- λ پارامتر تنظیم‌سازی اسکالر است،
- I ماتریس همانی با اندازه $n \times n$ است.

ما باید جواب p را به طور کامل در قالب **تجزیه مقدار منفرد (SVD)** ماتریس J، و همچنین معیاری از اندازه quadratic $||p||^2$ بیان کنیم، و سپس رفتار آن را هنگامی که $\lambda \to 0$ می‌شود تحلیل کنیم.

۲.۴ اثبات

۱.۲.۴ تجزیه مقدار منفرد J

SVD ماتریس J به صورت زیر است:

$$J = U\Sigma V^T$$

که در آن:

- $U \in R^{m\times m}$ یک ماتریس اورتوگونال است که ستون‌های آن بردارهای منفرد چپ J هستند،
- $\Sigma \in R^{m\times n}$ یک ماتریس قطری است (با مقادیر منفرد σ_i در قطر ماتریس J)،
- $V \in R^{n\times n}$ یک ماتریس اورتوگونال است که ستون‌های آن بردارهای منفرد راست J هستند.

بگذارید مقادیر منفرد J را به صورت $\sigma_1, \sigma_2, \ldots, \sigma_n$ بنامیم.

۲.۲.۴ بازنویسی دستگاه به کمک SVD

تجزیه مقدار منفرد J را در معادله داده شده جایگذاری می‌کنیم:

$$(J^T J + \lambda I)p = -J^T r$$

با استفاده از $J = U\Sigma V^T$، داریم:

$$J^T J = (V\Sigma^T U^T)(U\Sigma V^T) = V\Sigma^T \Sigma V^T$$

بنابراین معادله به صورت زیر در می‌آید:

$$(V\Sigma^T \Sigma V^T + \lambda I)p = -V\Sigma^T U^T r$$

حالا با ضرب کردن هر دو طرف معادله در V^T برای ساده‌سازی داریم:

$$(\Sigma^T \Sigma + \lambda I)V^T p = -\Sigma^T U^T r$$

اگر $p' = V^T p$ تا معادله به صورت زیر در می‌آید:

$$(\Sigma^T\Sigma + \lambda I)p' = -\Sigma^T U^T r$$

۳.۲.۴ حل برای p'

حالا می‌توانیم برای p' حل کنیم:

$$p' = (\Sigma^T\Sigma + \lambda I)^{-1}\Sigma^T U^T r$$

از آنجایی که $\Sigma^T\Sigma$ یک ماتریس قطری از مقادیر منفرد مربع شده است، این معادله به صورت زیر در می‌آید:

$$p' = \sum_{i=1}^{n} \frac{\sigma_i^2}{\sigma_i^2 + \lambda}(u_i^T r)v_i$$

۴.۲.۴ تحلیل اندازه quadratic p

برای محاسبه $\|p\|^2$، اندازه quadratic جواب را می‌گیریم:

$$\|p\|^2 = \sum_{i=1}^{n} \left(\frac{\sigma_i^2}{\sigma_i^2 + \lambda}\right)^2 \|u_i^T r\|^2$$

بنابراین، اندازه quadratic p به پارامتر تنظیم سازی λ و مقادیر منفرد σ_i بستگی دارد.

۵.۲.۴ رفتار هنگامی که $\lambda \to 0$

هنگامی که $\lambda \to 0$، جواب تمایل دارد بیشتر بر مقادیر منفرد **غیر صفر** تمرکز کند. به طور خاص، ضریب $\frac{\sigma_i^2}{\sigma_i^2+\lambda}$ برای تمامی $\sigma_i \neq 0$ به ۱ نزدیک می‌شود و برای $\sigma_i = 0$ به ۰ میل می‌کند. بنابراین، جواب به صورت زیر در می‌آید:

$$p \to \sum_{\sigma_i \neq 0} \frac{u_i^T r}{\sigma_i} v_i$$

این همان **حل معکوس شبه‌ای** دستگاه است، که جواب مسئله **کمترین مربعات** بدون تنظیم سازی

است، جایی که تنها مقادیر منفرد غیر صفر سهم دارند.

۳.۴ جواب نهایی

جواب p که به کمک SVD بیان شده است، به صورت زیر است:

$$p = \sum_{i=1}^{n} \frac{\sigma_i^2}{\sigma_i^2 + \lambda} u_i^T r v_i$$

اندازه quadratic p به صورت زیر است:

$$\|p\|^2 = \sum_{i=1}^{n} \left(\frac{\sigma_i^2}{\sigma_i^2 + \lambda} \right)^2 \|u_i^T r\|^2$$

و هنگامی که $\lambda \to 0$، جواب p به صورت زیر می‌شود:

$$\lim_{\lambda \to 0} p = \sum_{\sigma_i \neq 0} \frac{u_i^T r}{\sigma_i} v_i$$

سوال ۵

کمینه‌سازی تابع روزن براک

۱.۵ مقدمه

این کد برای بهینه‌سازی تابع روزن براک ($f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$) از دو روش BFGS و DFP همراه با جستجوی خط بک‌ترکینگ استفاده می‌کند. هدف این کد مقایسه دو روش در عملکرد بهینه‌سازی، سرعت همگرایی، و رفتار الگوریتم‌ها است.

۲.۵ شرح بخش‌های اصلی کد

در نمونه کد ۱ و نمونه کد ۲ توابع استفاده شده آمده است و در ادامه به تشریح آن‌ها می‌پردازیم.

```python
def rosenbrock(x):
    return 100 * (x[1] - x[0]**2)**2 + (1 - x[0])**2

def rosenbrock_grad(x):
    grad = np.zeros_like(x)
    grad[0] = -400 * x[0] * (x[1] - x[0]**2) - 2 * (1 - x[0])
    grad[1] = 200 * (x[1] - x[0]**2)
    return grad

def backtracking_line_search(func, grad_func, xk,
                             pk, alpha=1.0, rho=0.8, c=1e-4):
    while func(xk+alpha*pk) > func(xk) + c*alpha*np.dot(
                                        grad_func(xk), pk):
        alpha *= rho
    return alpha

# BFGS implementation
def bfgs(func, grad_func, x0, max_iter=100, tol=1e-5):
    n = len(x0)
    Hk = np.eye(n)
    xk = x0
    hist_bfgs = {"x": [], "f": [], "alpha": []}
    for _ in range(max_iter):
        grad = grad_func(xk)
        if np.linalg.norm(grad) < tol:
            break
        pk = -np.dot(Hk, grad)
        alpha = backtracking_line_search(func, grad_func, xk, pk)
        sk = alpha * pk
        xk_next = xk + sk
        yk = grad_func(xk_next) - grad
        rho_k = 1.0 / np.dot(yk, sk)
        I = np.eye(n)
        Hk = (I - rho_k * np.outer(sk, yk)) @ Hk @ (
            I - rho_k * np.outer(yk, sk)
            ) + rho_k * np.outer(sk, sk)

        hist_bfgs["x"].append(xk_next)
        hist_bfgs["f"].append(func(xk_next))
        hist_bfgs["alpha"].append(alpha)
        xk = xk_next
    return xk, hist_bfgs
```

۱: کد نمونه

```python
# DFP implementation
def dfp(func, grad_func, x0, max_iter=100, tol=1e-5):
    n = len(x0)
    Hk = np.eye(n)
    xk = x0
    hist_dfp = {"x": [], "f": [], "alpha": []}
    for _ in range(max_iter):
        grad = grad_func(xk)
        if np.linalg.norm(grad) < tol:
            break
        pk = -np.dot(Hk, grad)
        alpha = backtracking_line_search(func, grad_func, xk, pk)
        sk = alpha * pk
        xk_next = xk + sk
        yk = grad_func(xk_next) - grad
        rho_k = 1.0 / np.dot(yk, sk)
        Hk = Hk + rho_k*np.outer(sk, sk) - np.outer(np.dot(Hk,yk),
            np.dot(Hk, yk)) / np.dot(yk, np.dot(Hk, yk))

        hist_dfp["x"].append(xk_next)
        hist_dfp["f"].append(func(xk_next))
        hist_dfp["alpha"].append(alpha)
        xk = xk_next
    return xk, hist_dfp
```

۲: کد نمونه

۱.۲.۵ تعریف تابع روزن براک و گرادیان آن

- $rosenbrock(x)$: این تابع، تابع روزن براک را تعریف می‌کند..
- $rosenbrockgrad(x)$: گرادیان تابع روزن براک را محاسبه می‌کند که در ادامه برای تعیین جهت حرکت استفاده می‌شود.

۲.۲.۵ خط جستجوی بک‌ترکینگ

$backtrackinglinesearch$: این تابع به دنبال پیدا کردن یک اندازه گام (α) مناسب برای حرکت در جهت کاهشی تابع است. این کار با استفاده از روش خط جستجوی بک‌ترکینگ انجام می‌شود که در آن اندازه گام به طور پیوسته کاهش می‌یابد تا زمانی که شرایط آرمیجو برقرار شود.

۳.۲.۵ روش BFGS

$bfgs$: این تابع روش BFGS را پیاده‌سازی می‌کند. در این روش، معکوس هسیان (H_k) در هر تکرار با استفاده از اطلاعات جدید به‌روزرسانی می‌شود.

- در هر تکرار، گرادیان تابع محاسبه می‌شود.
- سپس، گام به سمت مینیمم با استفاده از معکوس هسیان و گرادیان محاسبه می‌شود.
- پس از آن، بردارهای s_k و y_k محاسبه شده و معکوس هسیان به‌روزرسانی می‌شود.

۴.۲.۵ روش DFP

dfp : این تابع مشابه روش BFGS است، اما برای به‌روزرسانی معکوس هسیان از فرمول متفاوتی استفاده می‌کند. در این روش نیز گام به سمت مینیمم با استفاده از به‌روزرسانی H_k و بردارهای s_k و y_k محاسبه می‌شود.

۵.۲.۵ نمایش نتایج

$plotresults$: این تابع برای ترسیم نمودارهایی از نتایج استفاده می‌شود.

- **نمودار اول:** تغییرات مقدار تابع هدف در هر تکرار برای دو روش نشان داده می‌شود.
- **نمودار دوم:** کاهش فاصله از نقطه بهینه $[1, 1]$ در هر تکرار برای دو روش مقایسه می‌شود.
- **نمودار سوم:** تغییرات اندازه گام (α) را در هر تکرار برای هر روش نشان می‌دهد.

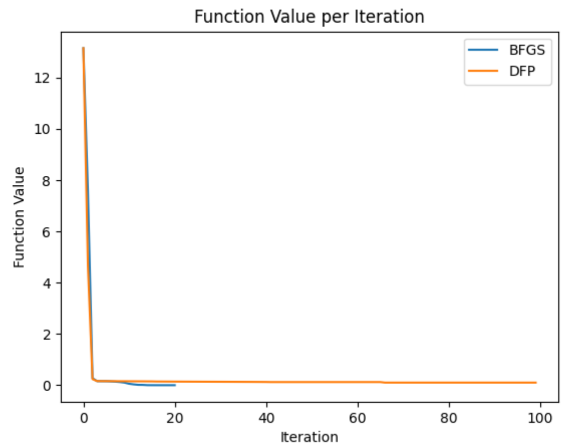
۶.۲.۵ اجرای الگوریتم‌ها

با استفاده از نقطه شروع $[-1.2, 1]$، هر دو روش BFGS و DFP اجرا شده و نتایج به‌دست آمده ثبت می‌شود. پس از اجرا، نتایج در قالب نمودارها نمایش داده می‌شود.

۳.۵ نتایج و تحلیل

۱.۳.۵ نمودار مقدار تابع هدف

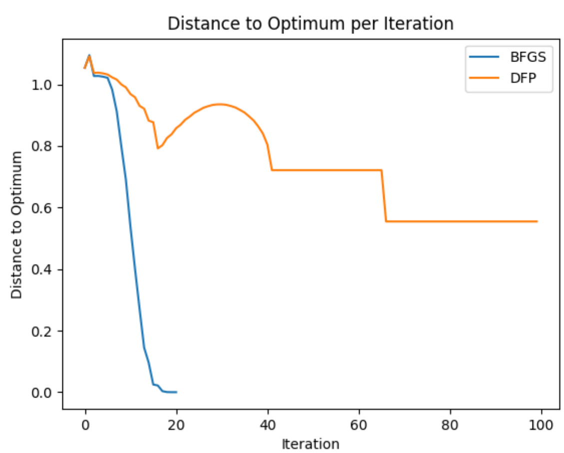
این نمودار نشان‌دهنده کاهش مقدار تابع هدف در هر تکرار است. روش BFGS سریع‌تر از DFP به مینیمم تابع می‌رسد (۱.۵).



شکل ۱.۵: نمودار مقدار تابع هدف

۲.۳.۵ نمودار فاصله از نقطه بهینه

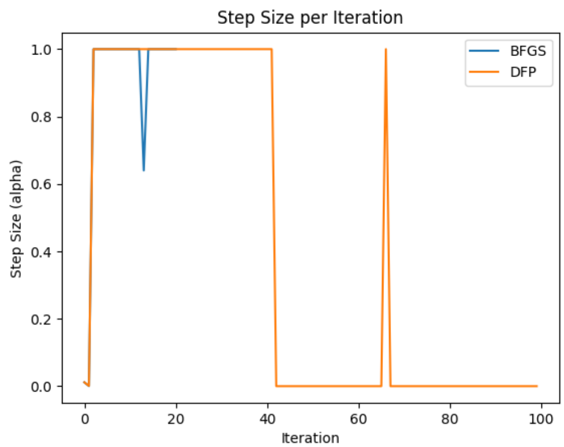
این نمودار کاهش فاصله از نقطه بهینه $[1, 1]$ را نشان می‌دهد. BFGS سریع‌تر از DFP فاصله را کاهش می‌دهد (۲.۵).



شکل ۲.۵: نمودار فاصله از نقطه بهینه

۳.۳.۵ نمودار اندازه گام (α)

این نمودار نشان‌دهنده تغییرات اندازه گام در هر تکرار است. BFGS اندازه گام‌های بزرگتری می‌گیرد و به نظر می‌رسد که انتخاب‌های گام در آن پایدارتر از DFP هستند (۳.۵).



شکل ۳.۵: نمودار اندازه گام (α)

۴.۵ نتیجه‌گیری

- BFGS نسبت به DFP سریع‌تر و پایدارتر به مینیمم تابع می‌رسد.
- روش BFGS عملکرد بهتری در به‌روزرسانی معکوس هسیان و کنترل همگرایی دارد.
- هر دو روش برای بهینه‌سازی موفق هستند، اما BFGS در عمل به‌طور عمومی ترجیح داده می‌شود به دلیل سرعت و پایداری بهتر.

در پایان، نتیجه در جدول ۱.۵ خلاصه شده است.

جدول ۱.۵: نتایج مقایسه روش‌های BFGS و DFP

ویژگی α	**نتیجه** β
سرعت همگرایی	BFGS نسبت به DFP سریع‌تر به مینیمم تابع می‌رسد.
عملکرد در به‌روزرسانی معکوس هسیان	روش BFGS عملکرد بهتری در به‌روزرسانی معکوس هسیان دارد.
کنترل همگرایی	روش BFGS کنترل بهتری بر همگرایی دارد.
ترجیح‌پذیری در عمل	BFGS به دلیل سرعت و پایداری بهتر در عمل ترجیح داده می‌شود.

سوال ۶

روش گرادیان مزدوج برای حل دستگاه معادلات خطی

۱.۶ مقدمه

این کد با استفاده از روش گرادیان مزدوج، دستگاه معادلات خطی $A\mathbf{x} = \mathbf{b}$ را حل می‌کند. هدف این روش کاهش باقی‌مانده $\mathbf{r}$ در هر تکرار است تا زمانی که مقدار $\mathbf{r}$ کمتر از مقدار آستانه مشخص‌شده شود. ماتریس A متقارن و مثبت معین است و از فرمول زیر برای تولید آن استفاده شده است:

$$A_{i,j} = \frac{1}{i+j+1}, \quad i,j = 1,\ldots,n.$$

۲.۶ شرح بخش‌های اصلی کد

در نمونه کد ۷ و نمونه کد ۴ توابع استفاده‌شده برای ایجاد ماتریس A و الگوریتم گرادیان مزدوج آمده است. جزئیات این توابع در ادامه تشریح شده است.

```
def create_A_matrix(n):
    A = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            A[i, j] = 1 / (i + j + 1)
    return A
```

کد نمونه ۳: A

```
def conjugate_gradient(A, b, x0, tol=1e-6):
    x = x0
    r = b - np.dot(A, x)
    p = r.copy()
    residuals = [np.linalg.norm(r)]
    num_iterations = 0

    while np.linalg.norm(r) > tol:
        Ap = np.dot(A, p)
        alpha = np.dot(r, r) / np.dot(p, Ap)
        x = x + alpha * p
        r_new = r - alpha * Ap
        residuals.append(np.linalg.norm(r_new))

        if np.linalg.norm(r_new) < tol:
            break

        beta = np.dot(r_new, r_new) / np.dot(r, r)
        p = r_new + beta * p
        r = r_new
        num_iterations += 1

    return x, num_iterations, residuals
```

کد نمونه ۴: $A\mathbf{x} = \mathbf{b}$

۱.۲.۶ تابع ایجاد ماتریس A

$create\_A\_matrix(n)$: این تابع ماتریس A متقارن مثبت معین را برای بعد n ایجاد می‌کند.

۲.۲.۶ الگوریتم گرادیان مزدوج

$conjugate\_gradient(A, b, x_0, tol)$: این تابع با استفاده از الگوریتم گرادیان مزدوج مسئله را حل می‌کند. جزئیات گام‌های این الگوریتم به شرح زیر است:

- محاسبه باقی‌مانده اولیه: $\mathbf{r} = \mathbf{b} - A\mathbf{x}$
- تنظیم جهت اولیه: $\mathbf{p} = \mathbf{r}$
- تکرار گام‌های زیر تا همگرایی:

۱. محاسبه مقدار گام: $\alpha = \frac{\mathbf{r}^T\mathbf{r}}{\mathbf{p}^T A\mathbf{p}}$

۲. به‌روزرسانی $\mathbf{x}$: $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$

۳. به‌روزرسانی باقی‌مانده: $\mathbf{r}_{\text{new}} = \mathbf{r} - \alpha A\mathbf{p}$

۴. بررسی همگرایی: اگر $||\mathbf{r}_{\text{new}}|| < \text{tol}$، توقف.

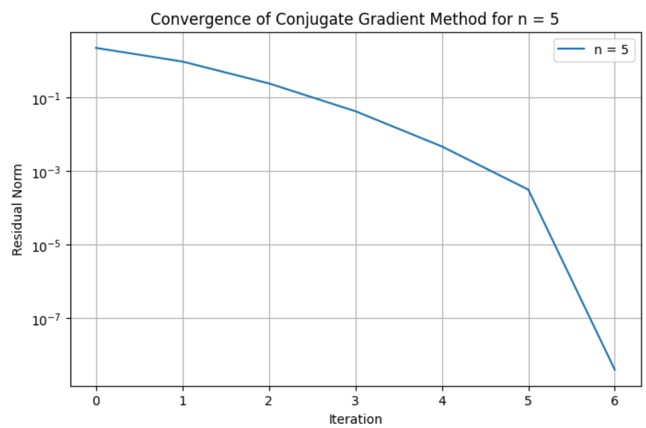
۵. به‌روزرسانی جهت: $\beta = \frac{\mathbf{r}_{\text{new}}^T\mathbf{r}_{\text{new}}}{\mathbf{r}^T\mathbf{r}}$

۶. $\mathbf{p} = \mathbf{r}_{\text{new}} + \beta\mathbf{p}$.

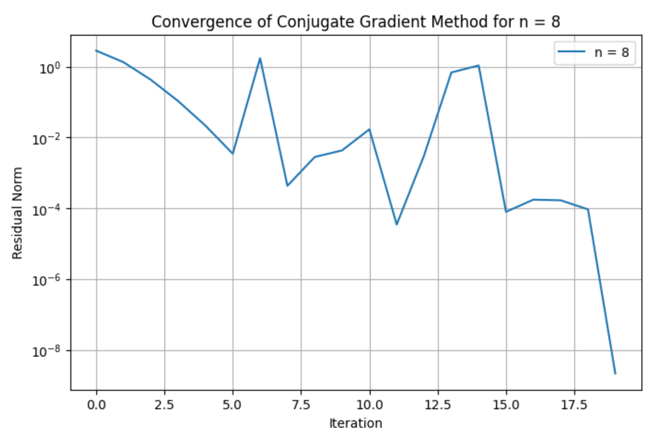
۳.۶ نمایش نتایج

۱.۳.۶ نمودار همگرایی

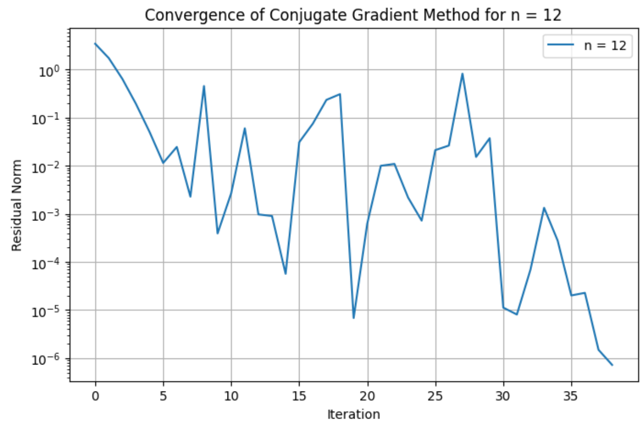
نمودارهای زیر نشان‌دهنده کاهش نرم باقی‌مانده $||\mathbf{r}||$ در هر تکرار برای ابعاد مختلف هستند.



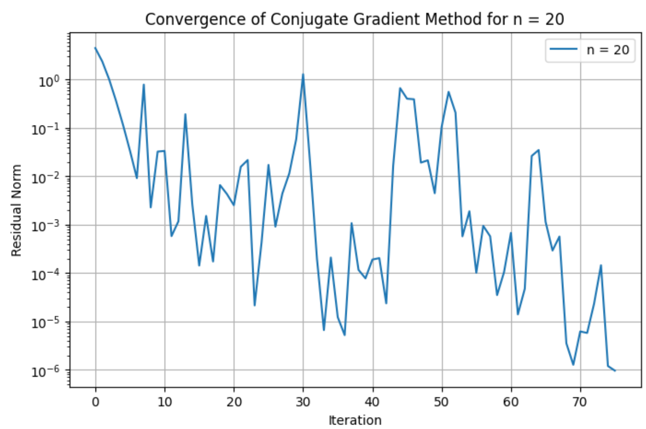
شکل ۱.۶: همگرایی باقی‌مانده برای $n = 5$



شکل ۲.۶: همگرایی باقی‌مانده برای $n = 8$



شکل ۳.۶: همگرایی باقی‌مانده برای $n = 12$



شکل ۴.۶: همگرایی باقی‌مانده برای $n = 20$

۲.۳.۶ تعداد تکرارها برای همگرایی

جدول زیر تعداد تکرارها برای رسیدن به همگرایی در هر بعد را نشان می‌دهد:

جدول ۱.۶: تعداد تکرارها برای ابعاد مختلف

بعد n	تعداد تکرارها
5	5
8	18
12	37
20	74

۴.۶ نتیجه‌گیری

- تعداد تکرارها برای رسیدن به همگرایی با افزایش ابعاد n افزایش می‌یابد.
- الگوریتم گرادیان مزدوج توانایی حل دستگاه معادلات خطی را با نرخ همگرایی مناسب دارد.
- نمودارهای باقی‌مانده نشان‌دهنده کاهش نمایی مقدار $||\mathbf{r}||$ در هر گام هستند.

سوال ۷

مقایسه روش‌های بهینه‌سازی برای تابع Logistic

۱.۷ مقدمه

این گزارش به مقایسه روش‌های بهینه‌سازی مختلف برای تابع Logistic می‌پردازد. تابع هدف مورد نظر برای داده‌های دودویی تعریف شده و هدف، کمینه‌سازی تابع زیان باینری کراس انتروپی است. روش‌های بررسی شده شامل Fletcher-Reeves، Polak-Ribiere و L-BFGS هستند که با استفاده از جستجوی خط بک‌ترکینگ گام بهینه را پیدا می‌کنند.

۲.۷ شرح بخش‌های اصلی کد

در فایل‌های مربوطه توابع و پیاده‌سازی هر روش آمده است. توضیح بخش‌های اصلی کد در ادامه آورده شده است.

۱.۲.۷ پیش‌پردازش داده

- $fetchopenml$: داده‌های MNIST بارگذاری شده و فقط نمونه‌های مربوط به ارقام 0 و 1 برای طبقه‌بندی دودویی انتخاب می‌شوند.
- $standardscaler$: داده‌ها استانداردسازی شده و سپس به مجموعه‌های آموزش و آزمون تقسیم می‌شوند.

۲.۲.۷ توابع Logistic

- $sigmoid$: تابع سیگموید برای تبدیل مقادیر ورودی به احتمال.
- $lossfunction$: تابع زیان Binary Cross-Entropy برای کمینه‌سازی.
- $computegradients$: محاسبه گرادیان‌ها برای وزن‌ها و بایاس.

۳.۲.۷ روش Fletcher-Reeves

این روش برای بهینه‌سازی غیرخطی از گرادیان مزدوج استفاده می‌کند. به‌روزرسانی بردار جهت‌دار به صورت زیر انجام می‌شود:

$$\beta_k = \frac{\nabla f(w_{k+1})^T \nabla f(w_{k+1})}{\nabla f(w_k)^T \nabla f(w_k)} \quad (۱.۷)$$

```
def fletcher_reeves(X, y, tol=1e-6, max_iter=100):
    m, n = X.shape
    w = np.zeros(n)
    b = 0

    dw, db = compute_gradients(w, b, X, y)
    d = -dw

    losses = [loss_function(w, b, X, y)]
    grad_norms = [np.linalg.norm(dw)]

    for i in range(max_iter):
        alpha = backtracking_line_search(w, b, X, y, dw, db)

        w_new = w + alpha * d
        b_new = b - alpha * db

        dw_new, db_new = compute_gradients(w_new, b_new, X, y)

        beta = np.dot(dw_new, dw_new) / (np.dot(dw, dw) + 1e-10)
        d = -dw_new + beta * d

        w, b = w_new, b_new
        dw, db = dw_new, db_new

        losses.append(loss_function(w, b, X, y))
        grad_norms.append(np.linalg.norm(dw))

        if np.linalg.norm(dw) < tol:
            break

    return w, b, losses, grad_norms
```

A کد نمونه ۵:

۴.۲.۷ روش Polak-Ribiere

این روش مشابه Fletcher-Reeves است اما β_k به صورت زیر تعریف می‌شود:

$$\beta_k = \max\left(0, \frac{\nabla f(w_{k+1})^T(\nabla f(w_{k+1}) - \nabla f(w_k))}{\nabla f(w_k)^T \nabla f(w_k)}\right) \tag{۲.۷}$$

```
def polak_ribiere(X, y, tol=1e-6, max_iter=100):
    m, n = X.shape
    w = np.zeros(n)
    b = 0

    dw, db = compute_gradients(w, b, X, y)
    d = -dw

    losses = [loss_function(w, b, X, y)]
    grad_norms = [np.linalg.norm(dw)]

    for i in range(max_iter):
        alpha = backtracking_line_search(w, b, X, y, dw, db)

        w_new = w + alpha * d
        b_new = b - alpha * db

        dw_new, db_new = compute_gradients(w_new, b_new, X, y)

        beta = max(0, np.dot(dw_new, dw_new - dw) / \
            (np.dot(dw, dw) + 1e-10))
        d = -dw_new + beta * d

        w, b = w_new, b_new
        dw, db = dw_new, db_new

        losses.append(loss_function(w, b, X, y))
        grad_norms.append(np.linalg.norm(dw))

        if np.linalg.norm(dw) < tol:
            break

    return w, b, losses, grad_norms
```

کد نمونه ۶: A

۵.۲.۷ روش L-BFGS

روش L-BFGS از حافظه محدود برای تقریب معکوس ماتریس Hessian استفاده می‌کند. این روش با استفاده از دو حلقه بازگشتی جهت حرکت را به صورت زیر محاسبه می‌کند:

$$q = g_k \quad \text{iterate: and} \quad \alpha_i = \frac{s_i^T q}{y_i^T s_i}, \quad q \leftarrow q - \alpha_i y_i \tag{۳.۷}$$

```python
def lbfgs_b():
    for k in range(max_iter):
        dw, db = compute_gradients(w, b, X, y)
        grad = np.concatenate([dw, [db]])
        grad_norms.append(np.linalg.norm(grad))
        q = grad
        alphas = []
        for i in reversed(range(len(s_list))):
            s = s_list[i]
            y_ = y_list[i]
            rho = 1.0 / (np.dot(y_, s) + 1e-10)
            alpha = rho * np.dot(s, q)
            alphas.append(alpha)
            q = q - alpha * y_

        if len(s_list) > 0:
            gamma = np.dot(s_list[-1], y_list[-1]) / \
                (np.dot(y_list[-1], y_list[-1]) + 1e-10)
        else:
            gamma = 1.0

        r = q * gamma
        for i in range(len(s_list)):
            s = s_list[i]
            y_ = y_list[i]
            rho = 1.0 / (np.dot(y_, s) + 1e-10)
            beta = rho * np.dot(y_, r)
            r = r + s * (alphas[len(s_list) - 1 - i] - beta)

        p = -r  # Search direction
        dw_update = p[:-1]
        db_update = p[-1]

        alpha = Backtracking_Line_search
        w_new = w + alpha * dw_update
        b_new = b + alpha * db_update

        s = np.concatenate([w_new - w, [b_new - b]])
        grad_new_dw, grad_new_db = compute_gradients(
                    w_new, b_new, X, y)
        grad_new = np.concatenate([grad_new_dw, [grad_new_db]])
        y_ = grad_new - grad

        s_list.append(s)
        y_list.append(y_)

        w, b = w_new, b_new
        losses.append(loss_function(w, b, X, y))

    return w, b, losses, grad_norms
```

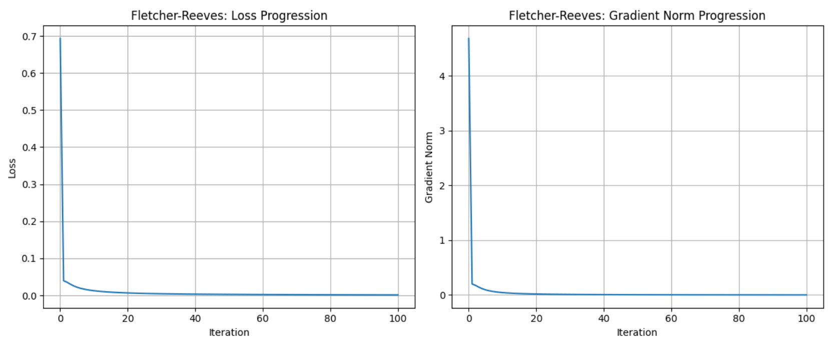
A ۷: کد نمونه

۳.۷ نتایج و تحلیل

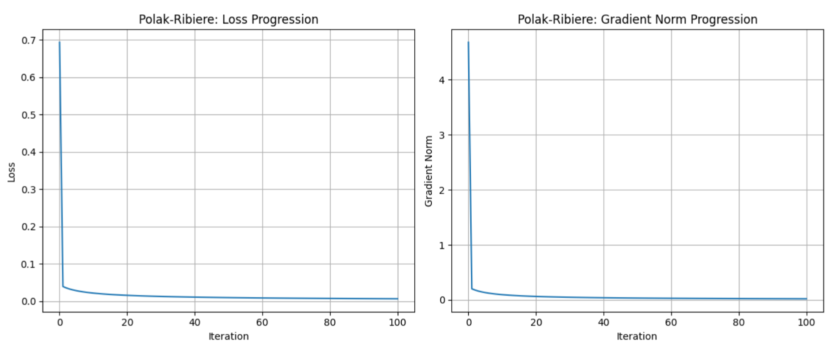
در ادامه مقادیر مختلف طول پله اولیه بررسی می‌شود.

۱.۳.۷ نمودار مقدار تابع هزینه و نرم گرادیان برای مقادیر مختلف $\alpha0$

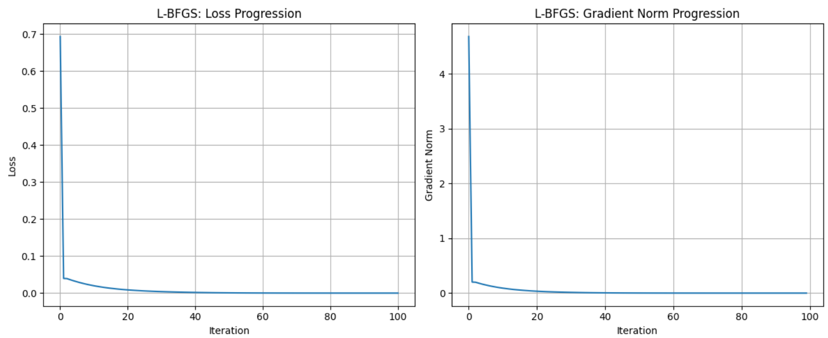
این نمودارها کاهش مقدار تابع هزینه و نرم گرادیان در هر تکرار را برای روش‌های مختلف با طول پله‌های مختلف نشان می‌دهد. برای بررسی، تنها مقادیر $\alpha0 = 0.1$ و $\alpha0 = 5.0$ بررسی می‌شود، اما نمودارهای بیشتر در نمونه کد موجود می‌باشد.



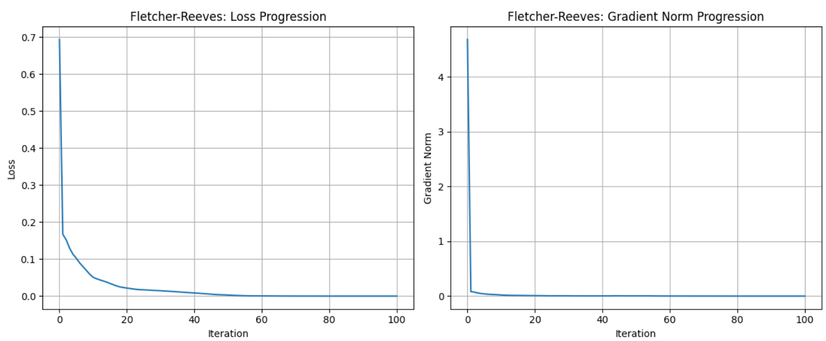
شکل ۱.۷: نمودار مقدار تابع هزینه و نرم گرادیان Fletcher-Reeves: $\alpha0 = 0.1$



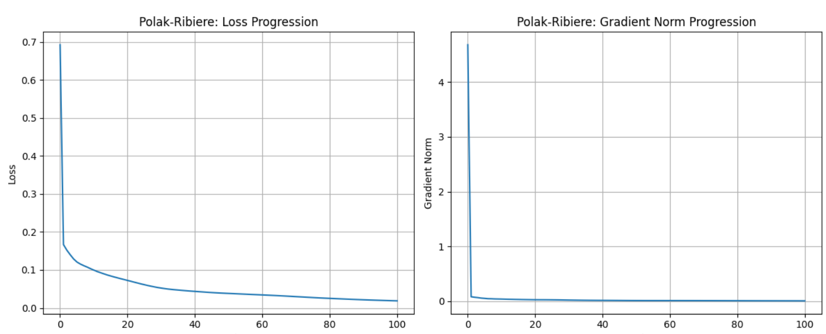
شکل ۲.۷: نمودار مقدار تابع هزینه و نرم گرادیان Polak-Ribiere: $\alpha0 = 0.1$



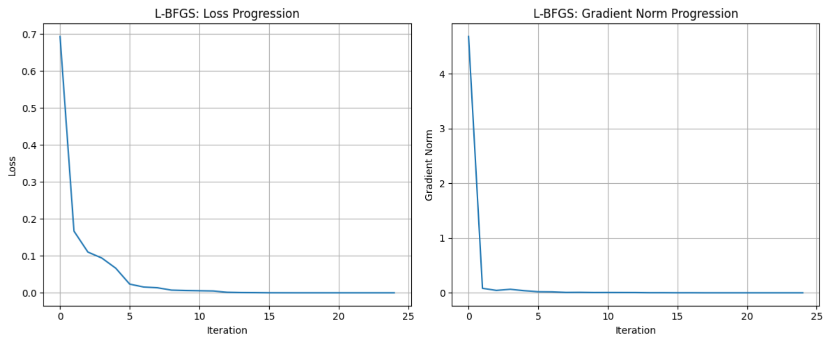
شکل ۳.۷: نمودار مقدار تابع هزینه و نرم گرادیان L-BFGS: $\alpha 0 = 0.1$



شکل ۴.۷: نمودار مقدار تابع هزینه و نرم گرادیان Fletcher-Reeves: $\alpha 0 = 5.0$



شکل ۵.۷: نمودار مقدار تابع هزینه و نرم گرادیان Polak-Ribiere: $\alpha 0 = 5.0$



شکل ۶.۷: نمودار مقدار تابع هزینه و نرم گرادیان L-BFGS: $\alpha 0 = 5.0$

۴.۷ مقایسه عملکرد

نتایج به صورت جدول زیر خلاصه شده است:

جدول ۱.۷: نتایج مقایسه روش‌ها برای مقادیر مختلف α: FR و PR

α	**دقت FR**	**زیان FR**	**زمان FR (ثانیه)**	**دقت PR**	**زیان PR**	**زمان PR (ثانیه)**
0.1	0.9993	0.001334	2.754	0.9989	0.006738	2.532
0.5	0.9986	0.000362	2.565	0.9986	0.003189	2.511
1.0	0.9979	0.000127	2.628	0.9986	0.004642	2.549
5.0	0.9976	0.000010	2.551	0.9986	0.018641	3.163

جدول ۲.۷: نتایج مقایسه روش‌ها برای مقادیر مختلف α: L–BFGS

α	**دقت L-BFGS**	**زیان L-BFGS**	**زمان L-BFGS (ثانیه)**
0.1	0.9993	5.144×10^{-6}	3.396
0.5	0.9993	2.034×10^{-7}	1.087
1.0	0.9989	1.183×10^{-7}	1.449
5.0	0.9979	8.818×10^{-8}	1.100

۵.۷ نتیجه‌گیری

از مقایسه مقادیر جدول مشاهده می‌شود که:

- روش Fletcher-Reeves (FR) و L-BFGS با دقت 0.9993 در مقدار $\alpha = 0.1$ بهترین دقت

را ارائه می‌دهد اما زمان اجرای بیشتری نسبت به سایر روش‌ها دارند. دلیل زمان بیشتر L-BFGS همانطور که در شکل ۳.۷ مشخص شده است، تعداد تکرار بیشتر می‌باشد؛ اما معمولا سرعت بیشتری نسبت به روش‌های دیگر دارد.

- روش L-BFGS با زیان نهایی بسیار کمتر (8.818×10^{-8}) و زمان اجرای کوتاه‌تر (1.100 ثانیه) نسبت به دیگر روش‌ها برای $\alpha = 5.0$ برتری دارد.
- روش Polak-Ribiere (PR) دقتی مشابه به Fletcher-Reeves ارائه می‌دهد اما زیان و زمان اجرای بیشتری در مقایسه با L-BFGS دارد.
- به طور کلی، روش L-BFGS به دلیل زیان کمتر و زمان اجرای کوتاه‌تر، بهترین عملکرد را در میان روش‌ها دارد.
- هر سه روش برای کمینه‌سازی تابع Logistic موفق هستند، اما روش L-BFGS برای کاربردهای عملی مناسب‌تر به نظر می‌رسد.

کتاب‌نامه