

دانشگاه صنعتی امیر کبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

تمرین دوم درس بهینه سازی (بخش عملی)

دکتر امیرمزلقانی

غلامرضا دار ۴۰۰۱۳۱۰۱۸

بهار ۱۴۰۱

فهرست مطالب

بخش اول: پرسشهای تشریحی	Error! Bookmark not defined.
سوال (۱)	Error! Bookmark not defined.
سوال (۲)	Error! Bookmark not defined.
بخش دوم: پیاده سازی	Error! Bookmark not defined.
سوال (۱)	Error! Bookmark not defined.
سوال (۲)	Error! Bookmark not defined.
سوال (۳)	Error! Bookmark not defined.
سوال (۴)	Error! Bookmark not defined.

سوال ۹

ابتدا مانند خواسته سوال متغیرهای مسئله بهینه سازی را تعریف میکنیم.

```
1 m = 300
2 n = 100
3 A = np.random.rand(m,n)
4 b = A@np.ones((n,1))/2
5 c = -np.random.rand(n, 1)
6
```

سپس مسئله شماره ۲ را برای بدست آوردن x^{rlx} حل میکنیم.

```
Optimal value: -34.58971131665731
X_rlx solution:
[[0.3602951]
 [1.         ]
 [1.         ]
 [0.         ]
 [0.         ]
 [1.         ]
```

مقدار L برابر با همین عدد منفی ۳۴ و خوردی است.

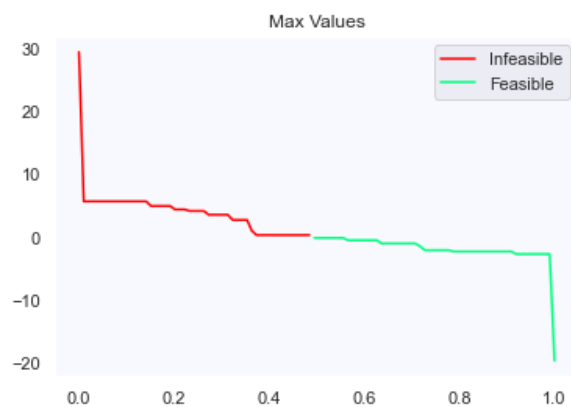
در ادامه با در نظر گرفتن مقادیر مختلف بین ۰ تا ۱ (۱۰۰ عدد مختلف) برای t ، موارد خواسته شده سوال را با \hat{x} که به صورت باینری تبدیل شده است، بدست می‌آوریم.

```

1  # Generate 100 t values between 0 and 1
2  N = 100
3  t_values = np.linspace(0, 1, N)
4  max_values = np.zeros(N)
5  u_values = np.zeros(N)
6
7  feasible_Us = []
8
9  for i in range(N):
10     t = t_values[i]
11
12     # Calculate x_hat based on x_rlx (x_hat is binary)
13     x_hat = (x_rlx>t).astype(float)
14
15     # Calculate upperbound
16     u = c.T@x_hat
17
18     # Calculate maximum element of AX-b (X=x_hat)
19     max_val = np.max(A@x_hat-b)
20
21     if max_val<=0:
22         feasible_Us.append(u)
23
24     # save the maximum value and the corresponding u
25     max_values[i] = max_val
26     u_values[i] = u.flatten()[0]
27

```

همچنین نمودار خواسته شده در سوال را در تصویر زیر مشاهده می‌کنید.



با انجام آزمایش‌های قبل به این نتیجه می‌رسیم که مقدار U برابر با -33.8836 است که طبق خواسته سوال تفاضل L و U را محاسبه میکنیم.

```
[58] 1 U-L
      ✓ 0.8s
</> 0.7060227946903481
```

سوال ۱۰)

در این سوال ابتدا به پیاده سازی توابعی برای تولید ماتریس های به فرم Hilbert و Tridiag پرداختیم.

```
1 tridiag()
[67]
</> array([[ 4., -1.,  0.,  0.,  0.],
          [-1.,  4., -1.,  0.,  0.],
          [ 0., -1.,  4., -1.,  0.],
          [ 0.,  0., -1.,  4., -1.],
          [ 0.,  0.,  0., -1.,  4.]])

1 hilbert()
[68]
</> array([[1.         , 0.5        , 0.33333333, 0.25       , 0.2        ],
          [0.5        , 0.33333333, 0.25       , 0.2        , 0.16666667],
          [0.33333333, 0.25       , 0.2        , 0.16666667, 0.14285714],
          [0.25       , 0.2        , 0.16666667, 0.14285714, 0.125       ],
          [0.2        , 0.16666667, 0.14285714, 0.125       , 0.11111111]])
```

سپس توابع مربوط به تابع داده شده و الگوریتم Backtracking برای یافتن اندازه قدم بهینه را پیاده کردیم.

```
1 # 1/2 * X^T * A * X - b^T * X
2 def f(x, A, b):
3     return 0.5 * x.T @ A @ x - b.T @ x
[162] ✓ 0.2s

1 def f_prim(x, A, b):
2     return A @ x - b
3
4 def f_prim_norm(x, A, b):
5     return np.linalg.norm(f_prim(x, A, b))
[163] ✓ 0.1s

1 def backtracking_algorithm(x, A, b, pk, rho, alpha_0, c):
2     alpha = alpha_0
3     # Check sufficient decrease condition
4     while f(x+alpha*pk, A, b) > f(x, A, b) + c * alpha * f_prim(x, A, b).T @ pk:
5         alpha *= rho
6     return alpha
7
[164] ✓ 0.1s
```

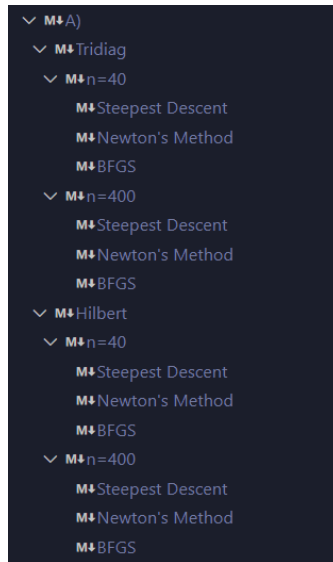
در ادامه، به پیاده‌سازی الگوریتم‌های بهینه‌سازی Steepest Descent و Newton و BFGS پرداختیم.

```
1 def steepest_descent(x0, A, b, rho, alpha_0, c, max_iter=100):
2     x = x0
3     for _ in range(max_iter):
4         grad = f_prim(x, A, b)
5         pk = -grad
6         alpha = backtracking_algorithm(x, A, b, pk, rho, alpha_0, c)
7         x += alpha * pk
8     return x
```

```
1 def newtons_method(x0, A, b, rho, alpha_0, c, max_iter=100):
2     x = x0
3     for _ in range(max_iter):
4         grad = f_prim(x, A, b)
5         hess = A
6         pk = -(np.linalg.inv(hess)@grad)
7
8         alpha = backtracking_algorithm(x, A, b, pk, rho, alpha_0, c)
9         # Update the current iterate
10        x += alpha * pk
11    return x
```

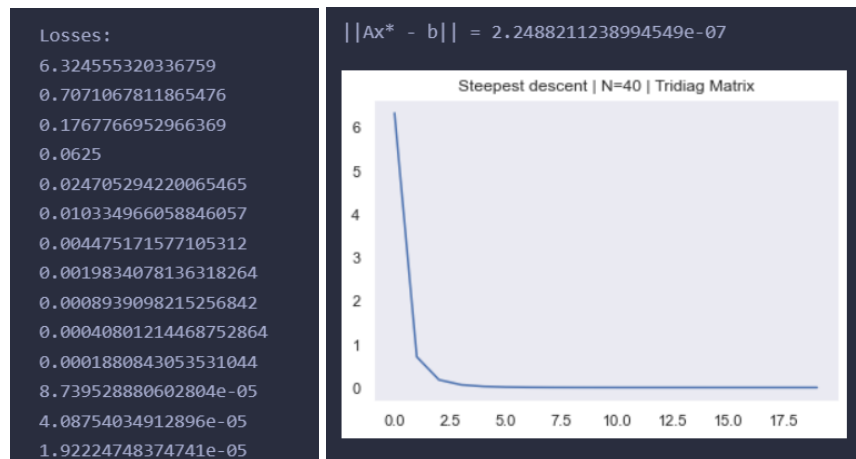
```
1 def BFGS(x0, A, b, rho, alpha_0, c, max_iter=100):
2     x = x0
3     # Initialize Hessian approx to I_n
4     x_prev = x
5     hess_approx = np.eye(A.shape[0])
6     for _ in range(max_iter):
7         grad = f_prim(x, A, b)
8         # Update the Hessian approximation
9         hess_approx = approximate_hessian(x0, x_prev, hess_approx, A, b)
10        # Calculate the BFGS pk = -(hess_approx @ grad)
11        pk = -(np.linalg.inv(hess_approx)@grad)
12        # Calculate the step size
13        alpha = backtracking_algorithm(x, A, b, pk, rho, alpha_0, c)
14        # Update the current iterate
15        x += alpha * pk
16        x_prev = x
17    return x
```

سپس آزمایش های گفته شده در سوال را انجام دادیم.

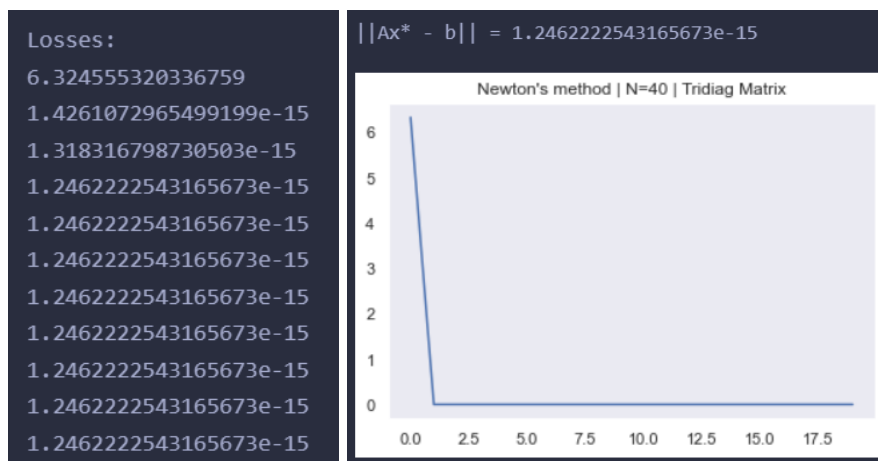


به عنوان مثال چند نمونه از نتایج آزمایش ها را با هم در گزارش میبینیم. برای مشاهده سایر آزمایش ها به نوت بوک مراجعه کنید.

در تصویر زیر نمودار میزان خطای ماتریس tridiag و $N=40$ را مشاهده میکنید که به روش Steepest Descent انجام شده است. میبینیم که پس از تعداد کمی iteration به همگرایی رسیده ایم.



به عنوان مثالی دیگر نمودار $N=40$ tridiag و روش Newton را مشاهده میکنید. همانطور که واضح است، روش نیوتن با تعداد ایتريشن کمتری و با سرعت بالاتری همگرا می شود.



جهت مشاهده سایر نمودارها به نوت بوک مراجعه شود.

الف) در جدول زیر خلاصه نتیجه آزمایش های مختلف موجود است. دقت کنید که هایپرپارامترهایی مانند ρ, α_0, C که در الگوریتم های مختلف مورد استفاده قرار میگیرد به میزان قابل توجهی در میزان خطای نهایی تاثیر دارند.

اسم روش	i	اندازه n	$\ A_i x^* - b\ _2$
Steepest Descent	1	40	2.24 e-7
Newton's Method	1	40	1.24 e-15
BFGS	1	40	2.34 e-5
Steepest Descent	1	400	2.24 e-7
Newton's Method	1	400	0
BFGS	1	400	2.35 e-5
Steepest Descent	2	40	0.26
Newton's Method	2	40	3.24
BFGS	2	40	0.104
Steepest Descent	2	400	0.49
Newton's Method	2	400	19.77
BFGS	2	400	1.34

ب) بخش محاسبه $x - \hat{x}$ ممکن است دچار باگ باشد.

اسم روش	i	اندازه n	$\ A_i x^* - b\ _2$	$\ A_i - \hat{A}_i\ _2$	$\ x^* - \hat{x}^*\ _2$
Steepest Descent	1	40	2.25 e-7	4 e-5	6.10 e-5
Newton's Method	1	40	0	4 e-5	6.10 e-5
BFGS	1	40	2.34 e-5	4 e-5	6.10 e-5
Steepest Descent	1	400	6.49 e-7	4 e-4	1.99 e-3
Newton's Method	1	400	7 e-14	4 e-4	1.99 e-3
BFGS	1	400	2.36 e-5	4 e-4	1.99 e-3
Steepest Descent	2	40	0.26	4 e-5	3.11 e-3
Newton's Method	2	40	5.1	4 e-5	1253275844
BFGS	2	40	0.104	4 e-5	0.01465
Steepest Descent	2	400	0.49	4 e-4	0.24641
Newton's Method	2	400	19.8	4 e-4	11806472648187
BFGS	2	400	2.94	4 e-4	0.13460

ج) همانطور که از نتایج جدول و مخصوصاً نمودارهای همگرایی میبینیم، روش نیوتن نسبت به روش steepest descent سرعن همگرایی بسیار بالاتری دارد. مشکلی که روش نیوتن دارد نیازمند بودن به محاسبه ماتریس Hessian است که باعث می-شود نتوان آن را در همه مسائل استفاده کرد. روش BFGS و سایر روش های شبه نیوتن، با تخمین زدن ماتریس Hessian مشکلات روش نیوتن را رفع میکنند. روش های شبه نیوتن از روش Steepest Descent همگرایی بهتری دارند و از روش نیوتن همگرایی کمتری دارند.

نکته دیگری که مشاهده می شود، تفاوت بین $i=1$ و $i=2$ است. در ماتریس Tridiag در آزمایش های مختلف هم نتیجه همگرایی بهتر بود یعنی خطای کمتر و هم فاصله بین نقاط بین بخش A,B سوال کمتر بود. اما در ماتریس Hilbert هم خطای همگرایی بیشتری داشتیم و هم تفاوت اندکی در ماتریس A باعث شد نقاط بهینه بسیار از یکدیگر دور شوند. علاوه بر این ها، ابعاد ماتریس A نیز هم در سرعت همگرایی و هم در خطای آن تاثیر عکس داشت.