

Homework 4

Qianlang Chen

Problem 1

```
[1]: import numpy

X = numpy.genfromtxt("data/X4.csv", delimiter=",")
y = numpy.genfromtxt("data/y4.csv")
n, d = X.shape
print(n, d)
```

30 4

Part (a)

Let's first define the cost function, which is just the SSE, along with its gradient:

```
[2]: # Alpha is 1*d.
def sse(alpha):
    return numpy.sum([(X[i] * alpha.T - y[i])**2 for i in range(n)])

def batch_grad_sse(alpha):
    return numpy.sum([2 * X[i] * (X[i] * alpha.T - y[i])
                      for i in range(n)], axis=0)
```

Next, let's introduce the batch gradient descent algorithm:

```
[3]: from numpy.linalg import norm
```

```
def batch_grad_descent(num_iter, alpha0, gamma, f, grad_f):
    print(f"i      {'alpha':<39}{'f(alpha)':<11}"
          f"norm(grad_f(alpha))" ) # header
    alpha = alpha0
    with numpy.printoptions(formatter={'float': '{:<8.6g}'.format}):
        print(f"{0:<4}{'{'alpha}':<39}{'f(alpha)':<11.4f}"
              f"{'norm(grad_f(alpha)):.6g}'")
        for i in range(num_iter):
            alpha = alpha - gamma * grad_f(alpha)
            print(f"{(i+1):<4}{'{'alpha}':<39}{'f(alpha)':<11.4f}"
                  f"{'norm(grad_f(alpha)):.6g}'")
    return alpha
```

Now, let's perform the batch gradient descent:

```
[4]: num_iter = 24
alpha0 = numpy.array([0, 0, 0, 0], float)
gamma = .0243

alpha = batch_grad_descent(num_iter, alpha0, gamma, sse, batch_grad_sse)
```

i	alpha				f(alpha)	norm(grad_f(alpha))
0	[0	0	0	0]	9746.8047	700.899
1	[12.8809	7.69675	5.38971	5.98979]	4097.1791	332.421
2	[6.98144	10.7968	8.74955	9.07992]	2756.7395	163.514
3	[9.68338	12.0455	10.844	10.6741]	2404.5216	83.9889
4	[8.44589	12.5484	12.1496	11.4966]	2302.1036	45.1687
5	[9.01266	12.751	12.9635	11.9208]	2269.5943	25.3471
6	[8.75308	12.8325	13.4709	12.1397]	2258.5575	14.7232
7	[8.87197	12.8654	13.7872	12.2527]	2254.6295	8.76917
8	[8.81752	12.8786	13.9843	12.3109]	2253.1870	5.31167
9	[8.84246	12.884	14.1072	12.341]	2252.6465	3.25228
10	[8.83103	12.8861	14.1839	12.3565]	2252.4413	2.0048
11	[8.83626	12.887	14.2316	12.3645]	2252.3628	1.24098
12	[8.83387	12.8873	14.2614	12.3686]	2252.3326	0.770154
13	[8.83497	12.8875	14.28	12.3707]	2252.3209	0.478729

```

14 [8.83446 12.8875 14.2915 12.3718 ] 2252.3164 0.297879
15 [8.83469 12.8876 14.2987 12.3724 ] 2252.3146 0.185468
16 [8.83459 12.8876 14.3032 12.3727 ] 2252.3140 0.115525
17 [8.83464 12.8876 14.306 12.3728 ] 2252.3137 0.0719784
18 [8.83461 12.8876 14.3078 12.3729 ] 2252.3136 0.0448541
19 [8.83462 12.8876 14.3089 12.373 ] 2252.3136 0.0279545
20 [8.83462 12.8876 14.3096 12.373 ] 2252.3135 0.0174235
21 [8.83462 12.8876 14.31 12.373 ] 2252.3135 0.0108603
22 [8.83462 12.8876 14.3102 12.373 ] 2252.3135 0.00676958
23 [8.83462 12.8876 14.3104 12.373 ] 2252.3135 0.0042198
24 [8.83462 12.8876 14.3105 12.373 ] 2252.3135 0.00263045

```

The batch gradient descent took 24 iterations to bring the model very close to the minimum, having a norm-of-gradient of only .00263 at the end. Impressive.

Part (b)

Since the incremental gradient descent only calculates the gradient for one particular “dimension” at a time, let’s modify the SSE’s gradient slightly:

```

[5]: def inc_grad_sse(alpha, i):
      return 2 * X[i] * (X[i] * alpha.T - y[i])

```

Next, the actual incremental gradient descent algorithm:

```

[6]: def inc_grad_descent(num_iter, alpha0, gamma, f, grad_f):
      print(f"i    {'alpha':<39}{'f(alpha)':<11}"
            f"norm(grad_f(alpha))" ) # header
      alpha = alpha0
      with numpy.printoptions(formatter={'float': '{:<8.6g}'.format}):
          print(f"{0:<4}{'f{'alpha'}':<39}{'f(alpha):<11.4f}"
                f"{norm(grad_f(alpha, 0)):.6g}")
          for i in range(num_iter):
              alpha = alpha - gamma * grad_f(alpha, i % n)
              print(f"{'(i+1)':<4}{'f{'alpha'}':<39}{'f(alpha):<11.4f}"
                    f"{norm(grad_f(alpha, i % n)):.6g}")
      return alpha

```

Now, let's perform the incremental gradient descent:

```
[7]: num_iter = 60
alpha0 = numpy.array([0, 0, 0, 0], float)
gamma = .25

alpha = inc_grad_descent(num_iter, alpha0, gamma, sse, inc_grad_sse)
```

i	alpha	f(alpha)	norm(grad_f(alpha))
0	[0 0 0 0]	9746.8047	23.7694
1	[5.05936 3.11242 0.145915 0.0772709]	6914.8987	14.3077
2	[7.63489 6.26646 2.38686 3.12437]	4788.0377	16.1583
3	[9.06871 8.41602 3.34634 6.63763]	3758.9346	10.4325
4	[9.74182 9.64648 5.8509 8.61336]	3101.4976	10.8281
5	[8.28545 9.95591 6.67654 9.29007]	2913.2871	4.58128
6	[7.84555 10.6282 6.83634 10.0192]	2832.5049	3.89651
7	[6.58193 10.789 7.27801 8.05638]	3027.5523	5.34848
8	[8.01594 11.6229 7.78818 9.43872]	2707.5235	6.42348
9	[8.68244 12.5074 8.24793 10.3335]	2581.0624	5.16412
10	[7.61708 12.7749 8.22023 10.7543]	2610.4948	2.82066
11	[9.15829 11.9889 9.95804 11.9565]	2413.9254	7.43851
12	[8.20498 12.1174 10.5559 11.961]	2382.4396	2.8977
13	[7.81684 12.595 11.1688 12.2999]	2360.9922	3.29386
14	[7.3343 13.0153 11.2882 12.1957]	2391.1515	1.98448
15	[9.36531 12.2226 12.6565 13.3709]	2297.3213	7.27754
16	[8.28192 12.5987 12.7703 10.9168]	2302.0141	6.1229
17	[8.01957 12.9971 11.1859 10.9804]	2367.3772	4.19647
18	[8.32605 12.5552 11.4083 10.0547]	2380.2467	2.52523
19	[9.41696 11.5573 12.5993 10.8506]	2330.0168	5.4614
20	[9.45276 12.158 13.2598 11.5771]	2285.1866	3.66551
21	[8.71918 12.7003 13.8461 11.72]	2259.0641	3.48796
22	[7.97708 13.2012 11.604 11.143]	2347.4273	6.18958
23	[9.76071 12.3856 12.6544 12.0797]	2303.2535	6.74789
24	[9.50435 11.8162 13.0726 11.7796]	2295.2595	2.3976
25	[8.4708 11.8192 11.9992 12.1428]	2312.2438	3.90992
26	[8.18913 12.3461 12.5812 10.4094]	2330.0006	4.83144
27	[7.39571 12.5715 12.8461 10.869]	2354.8089	2.65379

28	[9.38786	12.0244	13.1187	11.3376]	2292.3409	4.66646
29	[10.0122	12.5776	14.1705	11.7849]	2298.6952	4.78289
30	[10.8557	12.7058	15.3217	12.2642]	2383.3037	5.06179
31	[10.4872	13.414	15.4612	12.34]	2347.9204	2.49424
32	[10.3488	13.0575	16.2079	13.1829]	2355.8801	3.98409
33	[10.4257	12.0436	16.9153	12.4225]	2389.6051	3.93611
34	[10.4203	11.7405	17.0218	13.2044]	2407.7575	2.60263
35	[8.62469	12.038	14.0444	13.5285]	2276.3523	8.79948
36	[8.01517	12.6072	14.1968	13.9906]	2299.5870	3.08843
37	[6.66674	12.763	14.4259	10.2534]	2438.3509	8.76585
38	[8.05834	13.2042	14.8855	11.2935]	2285.7925	5.04121
39	[8.70365	13.8911	15.3038	12.1259]	2273.4552	4.5352
40	[7.62768	14.1515	13.0029	12.3612]	2328.9025	6.73318
41	[9.16359	12.7245	13.642	13.2218]	2266.5279	5.47408
42	[8.20763	12.8525	13.8227	13.2263]	2273.2204	2.08045
43	[7.81817	13.3176	14.12	13.5574]	2299.8366	2.61524
44	[7.33497	13.7254	13.7491	13.2373]	2338.2934	2.48125
45	[9.36564	12.5793	14.6504	14.2077]	2296.3644	6.32865
46	[8.28209	12.9188	14.4787	11.39]	2271.3288	6.86696
47	[8.01965	13.3145	12.222	11.4535]	2316.7072	5.72485
48	[8.3261	12.7823	12.1786	10.3091]	2337.8632	2.96158
49	[9.41698	11.6717	13.2663	11.0106]	2307.5960	5.20304
50	[9.45277	12.247	13.8189	11.7005]	2275.1974	3.34401
51	[8.71918	12.7878	14.3873	11.8433]	2255.6756	3.43896
52	[7.97709	13.2862	11.932	11.2318]	2333.1467	6.67083
53	[9.76071	12.4282	12.973	12.1667]	2294.9251	6.74314
54	[9.50435	11.8433	13.3895	11.8369]	2288.6093	2.44349
55	[8.4708	11.8462	12.2198	12.1997]	2303.7887	4.1179
56	[8.18913	12.37	12.7964	10.4406]	2323.0733	4.86174
57	[7.39571	12.5953	13.0418	10.8973]	2349.6474	2.62218
58	[9.38786	12.0365	13.314	11.3658]	2288.1996	4.67176
59	[10.0122	12.5861	14.3544	11.8129]	2298.1727	4.7438
60	[10.8557	12.7108	15.496	12.2921]	2386.1950	5.02967

Even after 60 iterations, the incremental gradient descent still has the model hanging around 5.03 for the norm-of-gradient. I've performed the gradient descent many times with different

gamma-values, but this seems to be about as good as it gets, unfortunately. Since the data size isn't so big in this particular problem, I'd prefer a batch gradient descent more, because it didn't consume any noticable time at all to reach a better result.

Problem 2

TODO

Problem 3

```
[8]: import numpy

A = numpy.genfromtxt("data/A.csv", delimiter=",")
n, d = A.shape
print(n, d)

# Compute the SVD of A.

from numpy import linalg

U, S, Vt = linalg.svd(A)
```

25 7

Part (a)

```
[9]: print(f"The second singular value is {S[1]:.12}")
```

The second singular value is 12.2

Part (b)

```
[10]: rank_A = linalg.matrix_rank(A)
print(f"The rank of A is {rank_A}")
```

The rank of A is 6

Part (c)

```
[15]: # A^T A = V S^2 V^T. Therefore, the eigenvectors are the columns of V,
# and the eigenvalues are the diagonal entries of S^2.
with numpy.printoptions(formatter={'float': '{:<8.4g}'.format}):
    print("Eigenvectors of (A^T A):",
          "\n".join(map(str, Vt[:rank_A,:])),
          "\n",
```



```
"Eigenvalues of (AT A):",
", ".join([f"{w:.12}" for w in (S**2)[:rank_A]]),
sep="\n")
```

Eigenvectors of (A^T A):

```
[-0.1652  0.4827  -0.2869  0.7743   0.1816  -0.1525  0.03968 ],
[0.3929  -0.3505  0.335   0.4659   0.1366  0.2136  -0.5738 ],
[0.11     -0.2947  -0.2068  0.2378  -0.7968  -0.4083  -0.01402],
[-0.2603  -0.3849  0.251   0.03063  0.4248  -0.7325  0.05674 ],
[0.4779   0.5705   0.1218  -0.2809  -0.01138 -0.4739  -0.3572 ],
[0.1591   0.1882   0.7341   0.2133  -0.2008  0.032   0.56    ]
```

Eigenvalues of (A^T A):

292.41, 148.84, 16.81, 4.0, 1.0, 0.25

Part (d)

```
[17]: S_3 = numpy.zeros((n, d))
      for i in range(3): S_3[i, i] = S[i]
      A_3 = U @ S_3 @ Vt

      print(f"Frobenius norm of (A - A_3): {linalg.norm(A - A_3):.12}")
```

Frobenius norm of (A - A₃): 2.29128784748

Part (e)

```
[18]: print(f"L2-norm of (A - A_3): {linalg.norm(A - A_3, 2):.12}")
```

L2-norm of (A - A₃): 2.0

Part (f)

TODO