

2025 年度 科学技術計算

学生番号 36714143
情報工学科知能情報分野
和田小百合

2025 年 12 月 11 日

目次

| | | |
|-----|---|---|
| 1 | 課題 05-1 | 2 |
| 1.1 | 課題の内容 | 2 |
| 1.2 | ハウスホルダー変換について | 2 |
| 1.3 | ギブンス変換について | 3 |
| 1.4 | 固有値固有ベクトルの導出について | 4 |
| 1.5 | 実行結果 | 5 |
| 2 | 課題 05-3 | 7 |
| 2.1 | 課題の内容 | 7 |
| 2.2 | QR 分解を利用する関数 full/reduced QR 分解を利用した thin SVD の計算方法 | 7 |
| 2.3 | 固有値分解を利用した単純な thinSVD の計算方法 | 7 |
| 2.4 | 実行結果 | 8 |

1 課題 05-1

1.1 課題の内容

小規模な密行列のすべての固有値を同時に反復的に求める方法として QR 法があります。QR 法は行列 A を QR 分解し、QR 変換を繰り返し適用することで固有値を求める手法です。一般の n 次対称行列に対する QR 分解の計算量は $O(n^3)$ であるため、QR 法の反復において一般の QR 分解を行ってしまうと、非常に計算量が大きくなってしまいます。また、QR 法の収束は 1 次であるため、反復回数も多くなることがあります。これを改善するための工夫として、三重対角行列に対する QR 法の反復では三重対角は保たれる性質から、まずハウスホルダー変換で元の対称行列を三重対角行列に変換します。この計算量は $O(n^3)$ ですが、前処理として一度だけで済みます。QR 法の反復における QR 分解は、ギブンス変換を用いて、三重対角行列を上三角行列へ変換します。この計算量は $O(n^2)$ で済みます。

それを実装した関数が授業資料の `eigh_by_qr_method()` です。この関数の QR 法の収束条件として「主対角成分 (diagonals) が変化しなくなるまで」と設定していたが、この条件では収束が不十分であり、非対角成分が完全には 0 に収束しません。そこで、QR 法の収束の精度を上げるために「優対角成分 (superdiagonals) も変化しなくなるまで」という条件を追加しているが、これにより QR 法の反復回数が増加してしまいます。なので、今回の課題では収束条件を主対角成分 (diagonals) のみとし、その時点で得られる近似固有値（主対角成分）と近似固有ベクトルを初期値としてレイリー商反復法を適用し、より精度の高い固有値と固有ベクトルを得る方法を実装し、計算量をはやくしました。

1.2 ハウスホルダー変換について

ハウスホルダー法は、鏡映行列とベクトルの積を使って、結果のベクトルが 1 つの要素を除いて 0 になるような鏡映行列（ハウスホルダー行列）を利用し、 $n - 1$ 回の変換で n 次対称行列を三重対角行列に変換する方法です。以下がハウスホルダー変換を行う関数です。

```

1 #ハウスホルダー法
2 def householder_step( A: np.ndarray, k: int ) -> Tuple[np.ndarray, np.ndarray]:
3     assert A.ndim == 2 #次元であるかチェック
4     assert A.shape[0] == A.shape[1] #正方行列かどうかチェック
5
6     n = A.shape[0]
7     assert 0 <= k < n - 2
8
9     x = A[k + 1:, k] # の列目のakk行目以降の部分を取得縦の部分+1()
10    y = np.zeros_like(x) #全部ゼロの配列
11    y[0] = - np.sign(x[0]) * norm(x) # ベクトルの先頭要素を設定y
12
13    u = (x - y) / norm(x - y) # 単位ベクトルを計算u
14    H = np.eye(len(x)) - 2 * np.outer(u, u) # ハウスホルダー行列を計算H
15
16    Q = np.eye(n) #単位行列
17    Q[k + 1:, k + 1:] = H # にを組み込むQH
18
19    QAQ = Q @ A @ Q # 変換後の行列を計算
20    return QAQ, Q
21
22 def householder_transform(A: np.ndarray,) -> Tuple[np.ndarray, np.ndarray]:
23     assert A.ndim == 2

```

```

24     assert A.shape[0] == A.shape[1]
25     n = A.shape[0]
26
27     Atri = A.copy()
28     Qhh = np.eye(n) # サイズの単位行列n
29
30     for k in range(n - 2):
31         Atri, Q = householder_step(Atri, k)
32         Qhh = Qhh @ Q # 直交行列を更新
33     return Atri, Qhh

```

1.3 ギブンス変換について

ギブンス変換はギブンス回転 (Givens rotation) を利用して、指定した 1 つの要素を 0 にするような変換を行います。1 回の変換で特定の要素を消去し、それを繰り返して他の要素も消していきます。これを繰り返すことによって、このギブンス変換では、三重対角対称行列を上三角行列へと変換します。目標とするのは、劣対角成分 $a_{i+1,i}$ をすべて 0 にすることです。なので、これらの要素をすべて 0 にし、かつ下三角成分を 0 に保つことで、ギブンス回転行列を用いて上三角行列 R を得ます、つまり QR 分解を実現できます。

以下にギブンス変換を行う関数を示す。

```

1  # ギブンス変換
2  def givens_r_matrix(xp_xq: np.ndarray) -> np.ndarray:
3      assert len(xp_xq) == 2
4
5      xp, xq = xp_xq[0], xp_xq[1]
6
7      denominator = norm(xp_xq)
8      c = xp / denominator
9      s = -xq / denominator
10
11     R = np.array([
12         [c, s],
13         [-s, c]
14     ])
15     return R
16
17 def qr_decom_by_givens(Atri_org: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
18     assert len(Atri_org.shape) == 2
19     assert Atri_org.shape[0] == Atri_org.shape[1]
20     n = Atri_org.shape[0]
21
22     Atri = Atri_org.copy()
23     Q = np.eye(n)
24
25     for k in range(n - 1):
26         p, q = k, k + 1
27         Rot2x2 = givens_r_matrix(Atri[[p, q]][:, p])
28         Atri[[p, q]] = Rot2x2.T @ Atri[[p, q]]
29         Q[:, [p, q]] = Q[:, [p, q]] @ Rot2x2
30
31     R = Atri
32     return Q, R

```

1.4 固有値固有ベクトルの導出について

ハウスホルダー変換を行い行列 A を三重行列にして、QR 分解にギブンス変換を行うことによって近似的な固有値と固有ベクトルを導出することができた。この近似値を正確な固有値、固有ベクトルにするためにレイリー商反復を行い実質的な値に近づけていく。

以下がレイリー商反復を行う関数です。

```

1  #レイリー商反復について
2  #関数
3  def rayleigh_quotient_iteration(A: np.ndarray, mu: float, u, tol = 1e-12, n = np.
4      ndarray, maxiter: int = 5000,) -> Tuple[float, np.ndarray]:
5      #入力 行列が対称かどうか
6      assert A.ndim == 2
7      assert A.shape[0] == A.shape[1]
8      n = A.shape[0]
9      assert len(u) == n
10
11     u_pre = u.copy()
12
13     #レイリー商反復
14     for i in range(maxiter):
15         try:
16             u = solve(A - mu * np.eye(n), u)      #シフト値を用いて連立方程式を解く mu
17         except:
18             break
19
20         u /= norm(u)      #正規化
21         mu = u.T @ A @ u
22
23         if np.allclose(u, u_pre) or np.allclose(u, -u_pre):
24             #の値がめっちゃちかくなつた→近似 u
25             break
26         u_pre = u.copy()
27
28     return mu, u

```

レイリー商反復では $A - \mu I$ の連立方程式を解く部分があり、 μ が A の固有値に近づくにつれて得意行列になりやすくなるため、その例外処理として $u = \text{solve}(A - \mu * \text{np.eye}(n), u)$ の部分で try catch を行っています。

また、これらの操作をすべて一つで行う関数を作成しました。以下がその関数です。また、QR 法関数も以下にしめす。

```

1  #法関数 QR
2  def eigh_by_qr_method(A0: np.ndarray, maxiter: int = 1000,) -> Tuple[np.ndarray
3      , np.ndarray]:
4      assert A0.ndim == 2
5      assert A0.shape[0] == A0.shape[1]
6      n = A0.shape[0]
7      A = A0.copy()
8
9      Qall = np.eye(n)
10     diagonals_pre = diag(A)
11     #ハウスホルダー変換初めの一回で計算量を削減()

```

```

11     Atri, Qhh = householder_transform(A)
12     Qall = Qhh
13     Atri_org = Atri.copy()
14
15     for i in range(maxiter):
16         #ギブンス変換
17         Q, R = qr_decom_by_givens(Atri_org)
18         Atri_org = R @ Q
19         Qall = Qall @ Q
20
21         diagonals = diag(Atri_org)
22         if (np.allclose(diagonals, diagonals_pre)):
23             break
24
25     diagonals_pre = diagonals.copy()
26
27     return diagonals, Qall
28
29
30 #一括で固有値固有ベクトルを求める関数
31 def eigh_via_qr_rayleigh( A: np.ndarray ) -> Tuple[np.ndarray, np.ndarray]:
32     assert A.ndim == 2
33     assert A.shape[0] == A.shape[1]
34     n = A.shape[0]
35
36     #法QR
37     eigenvalues_approx, Qqr = eigh_by_qr_method(A)
38     eigenvalues = np.zeros(n)
39     eigenvectors = np.zeros((n, n))
40
41     #レイリー商反復
42     for k in range(n):
43         mu0 = eigenvalues_approx[k]
44         u0 = Qqr[k, :]
45
46         mu_refined, u_refined = rayleigh_quotient_iteration(A, mu0, u0)
47
48         eigenvalues[k] = mu_refined
49         eigenvectors[:, k] = u_refined
50
51     return eigenvalues, eigenvectors

```

このように QR 法を行う前にハウスホルダー変換を行うことによって

三重対角化:1 回だけ $\Rightarrow O(n^3)$

QR iteration: 一回あたり $\Rightarrow O(n^2)$ (for 文で回す)

合計 $\Rightarrow O(n^3) + O(n^2)$

となり、授業資料よりも計算量を削減することができました。

1.5 実行結果

numpy と自作の固有値固有ベクトルの比較はこのように検証した。

```

1 #main

```

```

2 #実行
3 for i in tqdm(range(10)):
4     n = rng.integers(low = 2, high = 10)
5     A = rng.random(size=(n, n))
6
7     #対称行列
8     A = A.T @ A
9     print("行列 A:")
10    print(A)
11
12    d, U = eigh_via_qr_rayleigh(A)
13    eigvals_np, eigvecs_np = np.linalg.eigh(A)
14
15
16    with np.printoptions(formatter={'float': '{: 12.8f}'.format}, suppress=True
17        , linewidth=100):
18        d_np, U_np = np.linalg.eigh(A)
19        U_np = U_np[:, ::-1]
20        d_np = d_np[::-1]
21
22        # print("d_np", d_np)
23        # print("d", d)
24        print("d == d_np", np.allclose(d, d_np))
25
26        print("U_np\n", U_np, sep=',')
27        print("U\n", U, sep=',')
28
29        for i in range(len(U)):
30            print(
31                f"column {i}: U[{i}] == U_np[{i}] ?",
32                np.allclose(U[:, i], U_np[:, i]) or np.allclose(U[:, i], -U_np
33                    [:, i])
34            )
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
836
837
838
838
839
839
840
841
842
843
844
845
845
846
847
847
848
848
849
849
850
851
852
853
854
855
855
856
857
857
858
858
859
859
860
861
862
863
864
864
865
866
866
867
867
868
868
869
869
870
871
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
881
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
```

2 課題 05-3

2.1 課題の内容

固有値分解（対角化）を用いた thin SVD の単純な計算する関数と QR 分解を用いた full/reduced QR 分解を利用した thin SVD の計算をする関数を作成し、numpy 関数の np.linalg.svd と一致するか検証しました。

2.2 QR 分解を利用する関数 full/reduced QR 分解を利用した thin SVD の計算方法

以下が作成した関数です。

```

1  #分解を利用する関数QR full/reduced 分解を利用したQRthin の計算方法SVD
2  def svd_via_qr(A: np.ndarray,) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
3      #行列のサイズ計算
4      m, n = A.shape
5      print("m =",m)
6      print("n =",n)
7
8      if(m >= n):
9          #A.T @ A対称行列()の対角化->分解QR固有値計算->
10         ATA = A.T @ A
11         sigma, V = np.linalg.eigh(ATA)
12         sigma = np.sqrt(np.maximum(sigma, 0))
13
14         V = V[:, :-1]      #と行列の順番が違うため変換numpy
15         sigma = sigma[:-1]
16         a = A @ V
17         U, c = np.linalg.qr(a)
18
19     elif(m < n):
20         AAT = A @ A.T
21         sigma, U = np.linalg.eigh(AAT)
22         sigma = np.sqrt(np.maximum(sigma, 0))
23         U = U[:, :-1]      #関数と行列の順番が違うため変換numpy
24         sigma = sigma[:-1]
25         a = A.T @ U
26         V, c = np.linalg.qr(a)
27
28     VT = V.T    #最後に転置
29
30     return U, sigma, VT

```

基本的には授業資料に説明されていた通りの計算方法でプログラムを作成したが、numpy で実装されている行列と自分の関数で実装した初めの固有値分解の時に出てくる行列の順番が違っていたので順番を入れ替える操作をし、その後、QR 分解を二つめの行列を作成しました。

2.3 固有値分解を利用した単純な thinSVD の計算方法

以下が作成した関数である。

```

1  #固有値分解を利用した単純な計算方法thinSVD

```

```

2 def svd_via_eigh( A: np.ndarray, ) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
3     #行列のサイズ計算
4     m, n = A.shape
5     print("m =", m)
6     print("n =", n)
7
8     if(m >= n):
9         #A.T @ A対称行列()の対角化->分解QR固有値計算->
10        ATA = A.T @ A
11        sigma, V = np.linalg.eigh(ATA)
12        V = V[:, ::-1]    #関数と行列の順番が違うため変換numpy
13        sigma = np.sqrt(np.maximum(sigma, 0))
14        sigma = sigma[::-1]
15        Sigma_inv = np.diag(1.0 / sigma) # 各対角成分を逆数にするだけ
16        U = A @ V @ Sigma_inv
17
18    elif(m < n):
19        AAT = A @ A.T
20        sigma, U = np.linalg.eigh(AAT)
21        sigma = np.sqrt(np.maximum(sigma, 0))
22        sigma = sigma[::-1]
23        U = U[:, ::-1]    #関数と行列の順番が違うため変換numpy
24        Sigma_inv = np.diag(1.0 / sigma) # 各対角成分を逆数にするだけ
25        Sigma_inv = np.diag(1.0 / sigma)
26        V = A.T @ U @ Sigma_inv
27
28    return U, sigma, VT

```

基本的には授業資料に説明されていた通りの計算方法でプログラムを作成したが、numpy で実装されている行列と自分の関数で実装した初めの固有値分解の時に出てくる行列の順番が違っていたので順番を入れ替える操作をしました。

2.4 実行結果

以下のように自作関数と np.linalg.svd を比較しました。

```

1 for i in tqdm(range(10)):
2     m = rng.integers(low = 2, high = 10)
3     n = rng.integers(low = 2, high = 10)
4     A = rng.random(size=(m, n))
5
6     #分解を利用する関数のほうQR
7     U, sigma, VT = svd_via_qr(A)
8
9     with np.printoptions(formatter={'float': '{: 12.8f}'.format}, suppress=True,
10                           linewidth=100):
11         U_np, sigma_np, VT_np = np.linalg.svd(A)
12
13         print("sigma == sigma_np", np.allclose(sigma, sigma_np))
14
15         for i in range(U.shape[1]):
16             print(

```

```

16         f"column {i}: U[{i}] == U_np[{i}] ?",
17         np.allclose(U[:, i], U_np[:, i]) or np.allclose(U[:, i], -U_np
18             [:, i])
19     )
20     for i in range(len(VT)):
21         print(
22             f"column {i}: VT[{i}] == VT_np[{i}] ?",
23             np.allclose(VT[i, :], VT_np[i, :]) or np.allclose(VT[i, :], -
24                 VT_np[i, :])
25         )
26
27 for i in tqdm(range(10)):
28     m = rng.integers(low = 2, high = 10)
29     n = rng.integers(low = 2, high = 10)
30     A = rng.random(size=(m, n))
31
32     U, sigma, VT = svd_via_eigh(A)
33
34     with np.printoptions(formatter={'float': '{: 12.8f}'.format}, suppress=True
35         , linewidth=100):
36         U_np, sigma_np, VT_np = np.linalg.svd(A)
37
38         print("sigma == sigma_np", np.allclose(sigma, sigma_np))
39
40         for i in range(U.shape[1]):
41             print(
42                 f"column {i}: U[{i}] == U_np[{i}] ?",
43                 np.allclose(U[:, i], U_np[:, i]) or np.allclose(U[:, i], -U_np
44                     [:, i])
45             )
46         for i in range(len(VT)):
47             print(
48                 f"column {i}: VT[{i}] == VT_np[{i}] ?",
49                 np.allclose(VT[i, :], VT_np[i, :]) or np.allclose(VT[i, :], -
50                     VT_np[i, :])
51             )

```

行列は符号が違うことがあるのでそれを考慮して allclose で値自身一致しているか確認を行いました。また、自作関数は、thin 関数なので固有値分解したときに出てくる行列は numpy 関数と同じ行列のサイズが期待されるが、その後に出てくるもう一方の行列は実際の SVD と同じサイズで出力されない。なので for 文を自作関数のサイズに合わせてあってるかどうか判定した。

結果としてはそれぞれの関数を 10 回回して False が出ることはなかった。よって自作関数が numpy 関数とほぼ一致していることがわかりました。