PracticeSession12

August 22, 2025

Numerical Linear Algebra for Computational Science and Information Engineering

Jacobi-Davidson Method

Nicolas Venkovic (nicolas.venkovic@tum.edu)

[1]:
```
using LinearAlgebra, Plots, Printf, Latexify, LaTeXStrings, BenchmarkTools,␣
 ↪SparseArrays, LinearOperators, Random
using MatrixMarket: mmread
using IncompleteLU: ilu
using IterativeSolvers: gmres
```

Exercise #1: Rayleigh-Ritz Davidson

[2]:
```
function RR_Davidson_sym(A,
                         q::Vector{Float64},
                         k::Int,
                         tol::Float64)
  n, _ = size(A)
  Q = zeros(Float64, (n, k))
  W = zeros(Float64, (n, k))
  B = zeros(Float64, (k, k))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  t .= q
  j = 0
  DA = spdiagm(diag(A, 0))
  res = 1.; rho = 1.
  while (res > tol && j < k)
    for i in 1:j
      h = Q[:, i]'t
      t .-= h .* Q[:, i]
    end
    Q[:, j+1] = t ./ norm(t)
    j += 1
    W[:, j] = A * Q[:, j]
```

```
    B[1:j, j] = Q[:, 1:j]'W[:, j]
    B[j, 1:j-1] = Q[:, j]'W[:, 1:j-1]
    vals, vecs = eigen(B[1:j, 1:j], sortby=norm)
    rho = vals[j]
    y .= Q[:, 1:j] * vecs[:, j]
    r .= W[:, 1:j] * vecs[:, j] - rho .* y
    res = norm(r) / norm(rho)
    t .= (DA - rho * I) \ r
    println(j, ' ', res, " rho = ", rho)
  end
  return res, rho, y
end;
```

[3]:
```
n = 400_000;
Random.seed!(1);
A = spdiagm(-3 => rand(n-3),
            -2 => rand(n-2),
            -1 => rand(n-1),
             0 => rand(n),
             1 => rand(n-1),
             2 => rand(n-2),
             3 => rand(n-3))
A = A + A';
k = 100;
tol = 1e-8;
```

[4]:
```
res, val, y = RR_Davidson_sym(A, rand(n), k, tol);
```

```
1 0.5623342655194349 rho = 5.5025042176074805
2 0.1284415373003457 rho = 7.129215157698077
3 0.08353032086902117 rho = 7.309344587438295
4 0.08764699824055568 rho = 7.472717002187027
5 0.0776641431800263 rho = 7.612319098726835
6 0.07631265570321319 rho = 7.742804800501181
7 0.07346630946266092 rho = 7.861734448417484
8 0.07232020581705179 rho = 7.9814689105623655
9 0.06900933079867855 rho = 8.09093199631315
10 0.06538851955457156 rho = 8.194397160438847
11 0.060352732450336094 rho = 8.28353311565387
12 0.05546433321143787 rho = 8.361549823227614
13 0.05046331291828853 rho = 8.426372156629713
14 0.046222130125136686 rho = 8.481369118132104
15 0.042957645528949116 rho = 8.528137572681425
16 0.04010419317074076 rho = 8.569579597617865
17 0.03738049224492895 rho = 8.605760798157572
18 0.034010550971681965 rho = 8.637110569011542
19 0.030122445612559945 rho = 8.662455000389603
20 0.02591615457530581 rho = 8.681989223870302
```

```
21 0.021943744604167422 rho = 8.696215144695126
22 0.018424533563062468 rho = 8.70635346776391
23 0.015575716248041459 rho = 8.713530576824866
24 0.013344297955686375 rho = 8.718745496783992
25 0.011525582590096003 rho = 8.722616519433519
26 0.010072761432781926 rho = 8.725543834107835
27 0.008855935620503394 rho = 8.727801944682856
28 0.007824091347839846 rho = 8.72954855503826
29 0.006991003452097231 rho = 8.730924511608691
30 0.006347269629965446 rho = 8.7320346528038
31 0.005905694887304 1915 rho = 8.732968407273498
32 0.005596034776359827 rho = 8.733791210681506
33 0.005365559206834378 rho = 8.734540217098672
34 0.005145835720290661 rho = 8.73523173971389
35 0.004841921363809384 rho = 8.735862013891774
36 0.004410161198420 8745 rho = 8.736404464034702
37 0.003884831911585 4517 rho = 8.73684330846649
38 0.003311939349343335 rho = 8.73717322653072
39 0.002750923070496 4595 rho = 8.737407224351603
40 0.002246198433282336 rho = 8.73756602492816
41 0.001796195236678 4652 rho = 8.73766979717198
42 0.001430879785734 1068 rho = 8.737735749880853
43 0.001132842858326447 rho = 8.737777433509471
44 0.000892152425533 3152 rho = 8.737803381711377
45 0.000700952133698 0109 rho = 8.737819446316957
46 0.000552039310964 7698 rho = 8.737829377539029
47 0.000436049993846 79765 rho = 8.737835548104105
48 0.000347310079877 49376 rho = 8.737839425727103
49 0.000275356849804 22074 rho = 8.737841872352508
50 0.000219658479039 9337 rho = 8.737843408023112
51 0.000175873972781 82742 rho = 8.737844388291396
52 0.000140533948365 20216 rho = 8.737845014125636
53 0.000112232342491 46414 rho = 8.737845413693224
54 8.947789652984869e-5 rho = 8.737845668997407
55 7.088164154144676e-5 rho = 8.737845830666796
56 5.6003366041352054e-5 rho = 8.737845932036025
57 4.4007539362660756e-5 rho = 8.73784599532869
58 3.459039816406418e-5 rho = 8.737846034463349
59 2.712283765768511e-5 rho = 8.737846058731375
60 2.1096206905614416e-5 rho = 8.737846073582894
61 1.636995801854681e-5 rho = 8.737846082549819
62 1.265101339694211e-5 rho = 8.73784608794026
63 9.771323419369616e-6 rho = 8.737846091144696
64 7.528599236500348e-6 rho = 8.737846093055486
65 5.771501890823205e-6 rho = 8.737846094183604
66 4.44043999161414e-6 rho = 8.737846094848608
67 3.424734657718221e-6 rho = 8.737846095243743
68 2.6475694694680784e-6 rho = 8.737846095478888
```

```
69 2.0554061437494194e-6 rho = 8.737846095620014
70 1.6018872532011814e-6 rho = 8.737846095705502
71 1.2467156659721734e-6 rho = 8.737846095757341
72 9.686617262936191e-7 rho = 8.737846095788758
73 7.484427538565848e-7 rho = 8.737846095807596
74 5.710186387802647e-7 rho = 8.73784609581871
75 4.3331635086220644e-7 rho = 8.737846095825144
76 3.263791811921448e-7 rho = 8.737846095828797
77 2.4720606600071884e-7 rho = 8.737846095830891
78 1.8724143287951344e-7 rho = 8.7378460958321
79 1.425738706181647e-7 rho = 8.73784609583282
80 1.1046735944767512e-7 rho = 8.737846095833175
81 8.619639360948359e-8 rho = 8.737846095833417
82 6.782501326977861e-8 rho = 8.737846095833575
83 5.340052348765676e-8 rho = 8.737846095833682
84 4.186719563452158e-8 rho = 8.737846095833714
85 3.272780963871188e-8 rho = 8.737846095833765
86 2.5409036060718906e-8 rho = 8.73784609583381
87 1.9675532436217256e-8 rho = 8.737846095833824
88 1.521252276169369e-8 rho = 8.737846095833778
89 1.1875443449477244e-8 rho = 8.73784609583379
90 9.28211220814716e-9 rho = 8.737846095833824
```

Exercise #2: Rayleigh-Ritz generalized Davidson

[5]:
```
function RR_GD_sym(A,
                   q::Vector{Float64},
                   k::Int,
                   tol::Float64;
                   T=I)
  n, _ = size(A)
  Q = zeros(Float64, (n, k))
  W = zeros(Float64, (n, k))
  B = zeros(Float64, (k, k))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  t .= q
  j = 0
  res = 1.; rho = 1.
  while (res > tol && j < k)
    for i in 1:j
      h = Q[:, i]'t
      t .-= h .* Q[:, i]
    end
    Q[:, j+1] = t ./ norm(t)
    j += 1
```

```
    W[:, j] = A * Q[:, j]
    B[1:j, j] = Q[:, 1:j]'W[:, j]
    B[j, 1:j-1] = Q[:, j]'W[:, 1:j-1]
    vals, vecs = eigen(B[1:j, 1:j], sortby=norm)
    rho = vals[j]
    y .= Q[:, 1:j] * vecs[:, j]
    r .= W[:, 1:j] * vecs[:, j] - rho .* y
    res = norm(r) / norm(rho)
    t .= T \ r
    println(j, ' ', res, " rho = ", rho)
  end
  return res, rho, y
end;
```

[6]:
```
function power_iters(A, x, it=4)
  val = 1.
  for i in 1:it
    x ./= norm(x)
    Ax = A * x
    val = x'Ax
    x .= Ax
  end
  return val
end;

Random.seed!(1);
val = power_iters(A, rand(n), 10);
P = ilu(A - val * I);

res, val, y = RR_GD_sym(A, rand(n), k, tol, T=P);
```

```
1 0.5652921354850687 rho = 5.502991844091799
2 0.0004564739385947983 rho = 7.614679111888191
3 0.0010429837013429507 rho = 7.614716169746169
4 0.0008114505698087089 rho = 7.6148277006865
5 0.0004154993029105994 rho = 7.614848274492811
6 0.0006019638798012076 rho = 7.614849160463414
7 0.0014115101560862092 rho = 7.614995047217575
8 0.0011658296432102498 rho = 7.6149998574321724
9 0.0013103667788831959 rho = 7.615076169844155
10 0.0012103880817764133 rho = 7.615077148245075
11 0.00237746016289937 rho = 7.615182809981137
12 0.0017570752449538188 rho = 7.615332087059788
13 0.0027012269515586146 rho = 7.615456597489421
14 0.0023159834847760433 rho = 7.615685092232995
15 0.002771640797325327 rho = 7.6157103198756575
16 0.0024328411154661807 rho = 7.615930030415946
17 0.002704471297357363 rho = 7.615938653385452
```

```
18 0.002500340030988222 rho = 7.6161090224985175
19 0.0026073262755803082 rho = 7.616110664942767
20 0.0028698688529993784 rho = 7.6164140100091355
21 0.0037376783967447954 rho = 7.616499221497966
22 0.0037119277266869323 rho = 7.6166853903922025
23 0.003660719510753583 rho = 7.616685602795339
24 0.004547063334677753 rho = 7.617445666585844
25 0.005414580833985792 rho = 7.617508346152069
26 0.005289426841187092 rho = 7.618370822436297
27 0.00680797752497033 rho = 7.618727721572579
28 0.005792654277023232 rho = 7.619186825031942
29 0.007349272554799169 rho = 7.619814380733926
30 0.006396925656201402 rho = 7.620087593451255
31 0.007744623797653047 rho = 7.620893111754628
32 0.00674110832059622 rho = 7.621242037848414
33 0.00808640962963781 rho = 7.621848268583001
34 0.007156502585989917 rho = 7.6221522220079025
35 0.008340772534919176 rho = 7.622794494518729
36 0.007682575820764469 rho = 7.6229635186322655
37 0.008624246200285408 rho = 7.623846485115216
38 0.008065714915511996 rho = 7.624013675256042
39 0.008971173100905998 rho = 7.625031327538066
40 0.008376222613361585 rho = 7.625208168171494
41 0.00937652246248486 rho = 7.625959369782292
42 0.008747788979068333 rho = 7.626162430761336
43 0.009786716318295367 rho = 7.626797043925395
44 0.009080105558650695 rho = 7.6270848782104474
45 0.010294582817753114 rho = 7.627812534897339
46 0.009496133019402058 rho = 7.628325029026282
47 0.01048657027656981 rho = 7.62889024437003
48 0.01005971545144636 rho = 7.629610229346046
49 0.010745809056793109 rho = 7.629811041033001
50 0.01033862031998934 rho = 7.630522479137997
51 0.011044253395233943 rho = 7.630778887989021
52 0.010610118514646858 rho = 7.631387706368109
53 0.011308545214972123 rho = 7.631585731125301
54 0.01099284501194834 rho = 7.632534600216525
55 0.011558624928993642 rho = 7.6326887821813045
56 0.01089163831894301 rho = 7.633465508947011
57 0.011737936219625334 rho = 7.6340445683072815
58 0.011125676045642591 rho = 7.634254387291748
59 0.012256339564437679 rho = 7.635407650403889
60 0.011886712992889983 rho = 7.635477034424006
61 0.012683821423748366 rho = 7.636541943627841
62 0.01239334639055247 rho = 7.636596161166726
63 0.013207265347529161 rho = 7.637334969412448
64 0.012602678313665365 rho = 7.637812700579612
65 0.01368163388198239 rho = 7.638808958366562
```

```
66 0.01305588225339245 rho = 7.639213490773478
67 0.014081782020771594 rho = 7.640291767091625
68 0.013529595531905567 rho = 7.640498316074468
69 0.014438947663797117 rho = 7.64173371218145
70 0.014046203470698506 rho = 7.641848097485911
71 0.014630955501705864 rho = 7.643418854420895
72 0.01463452340228794 rho = 7.643418861129096
73 0.014654571929325247 rho = 7.645009090629656
74 0.01491496976069584 rho = 7.645042723178154
75 0.014804872863026362 rho = 7.646396400320728
76 0.015311652917842934 rho = 7.646553612912283
77 0.014847089058998376 rho = 7.64774594567219
78 0.0157324365793459 rho = 7.648310995128022
79 0.01521096994805431 rho = 7.648965179281971
80 0.01613657990344554 rho = 7.649864319058488
81 0.015576161828853961 rho = 7.6501738036450755
82 0.016453568067653315 rho = 7.651776988941859
83 0.016342654847809533 rho = 7.651785026639651
84 0.0164908039265479 rho = 7.653252061598938
85 0.016610468194785497 rho = 7.6532622173776765
86 0.016568062831101578 rho = 7.654889930663522
87 0.017270903536713195 rho = 7.655230840380457
88 0.01673539120788653 rho = 7.656205660446949
89 0.01740244174780603 rho = 7.6565934765456385
90 0.016971227589828042 rho = 7.657775612534865
91 0.017751539762612158 rho = 7.658373691898981
92 0.017128639684468552 rho = 7.659004601404208
93 0.017823029631710164 rho = 7.659992210736967
94 0.01713874856803746 rho = 7.660653355667013
95 0.01793777675923049 rho = 7.661742622650129
96 0.017468969408683255 rho = 7.661986820659884
97 0.017788460981609366 rho = 7.6637610407231955
98 0.01825103166925543 rho = 7.663920646695679
99 0.017831487894864734 rho = 7.665289034025154
100 0.018305317367766607 rho = 7.665564128921554
```

Exercise #3: harmonic Ritz Davidson

[7]:
```
function HR_Davidson_sym(A,
                         q::Vector{Float64},
                         k::Int,
                         sigma::Float64,
                         tol::Float64)
  n, _ = size(A)
  Q = zeros(Float64, (n, k))
  W = zeros(Float64, (n, k))
  H = zeros(Float64, (k, k))
```

```
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  t .= q
  j = 0
  DA = spdiagm(diag(A, 0))
  res = 1.; rho = 1.
  while (res > tol && j < k)
    j += 1
    w = (A - sigma * I) * t
    for i in 1:j
      witw = W[:, i]'w
      w .-= witw * W[:, i]
      t .-= witw * Q[:, i]
    end
    w_norm = norm(w)
    W[:, j] = w / w_norm
    Q[:, j] = t / w_norm
    H[j, 1:j] = W[:, j]'Q[:, 1:j]
    H[1:j, j] = W[:, 1:j]'Q[:, j]
    vals, vecs = eigen(H[1:j, 1:j])
    vals = 1. ./ vals
    ind = argmin(norm.(vals))
    y .= Q[:, 1:j] * vecs[:, ind]
    yty = y'y
    y ./= sqrt(yty)
    drho = 1. / (vals[ind] * yty)
    rho = sigma + drho
    r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - drho * y
    res = norm(r) / norm(rho)
    t .= (DA - rho * I) \ r
    println(j, ' ', res, " rho = ", rho)
  end
  return res, rho, y
end;
```

[8]:
```
sigma = 2.;
Random.seed!(1);
res, val, y = HR_Davidson_sym(A, rand(n), k, sigma, tol);
```

```
1 0.5216274696252504 rho = 5.754506744294758
2 5.207422833772338 rho = -0.35041663590349614
3 1.751702319956281 rho = -0.8447805228684984
4 73.91980975631617 rho = 0.022978058389639067
5 73.9217541048969 rho = 0.022977407121740345
6 73.92083350112044 rho = 0.02297769291496543
7 73.86482751744754 rho = 0.02299506277166885
8 73.92716095700938 rho = 0.02297553378178896
```

```
9 73.95467337606122 rho = 0.022966955001164857
10 73.94741909872266 rho = 0.022969122076255788
11 73.93499833704101 rho = 0.022972962126049268
12 73.93607207408535 rho = 0.022972627675061696
13 73.75223017938532 rho = 0.023029714849970073
14 73.74826709040514 rho = 0.023030948055970724
15 73.6925329386231 rho = 0.02304834511551812
16 73.69649917559141 rho = 0.023047091774109907
17 73.66817696560311 rho = 0.023055947644672337
18 73.67280344337838 rho = 0.023054480394906207
19 73.59699665841377 rho = 0.023078229951535967
20 73.60943012864344 rho = 0.0230742708785463
21 73.613852940564 rho = 0.02307283342356281
22 73.53116238237656 rho = 0.023098687514461647
23 73.44215913761698 rho = 0.023126652364934053
24 65.6371544161288 rho = 0.022990824065580506
25 60.20694925009618 rho = 0.02506160255760226
26 76.51316276193573 rho = 0.019678404794351234
27 143.76810893525266 rho = 0.010433968025008511
28 85.25751664434411 rho = -0.017361426293751592
29 7.803013304558141 rho = -0.17209356549170396
30 3.818100762568524 rho = -0.31141259652840203
31 3.399595472474665 rho = -0.34187931367604785
32 3.8613953510999712 rho = -0.30355275948277427
33 3.5314545624294733 rho = -0.32324768736602394
34 3.242608826999808 rho = -0.34540715902549657
35 3.293444907501821 rho = -0.3409864983306852
36 3.2628069126128563 rho = -0.3386971700835826
37 2.987318788758643 rho = -0.3611884896500035
38 4.421817259908993 rho = -0.2590013924422552
39 4.920665511150169 rho = -0.23551977041649552
40 6.813544634423831 rho = 0.19365683206711615
41 6.626970082124243 rho = 0.19932891527360153
42 6.560356538804974 rho = 0.20143283999575812
43 6.485202736281298 rho = 0.20363732971644044
44 6.585020889036982 rho = 0.20026990015722013
45 7.206693747804992 rho = 0.18190410258345757
46 7.234259041146542 rho = 0.18117511001682796
47 7.174083537325386 rho = 0.18276522885240576
48 7.207746416424981 rho = 0.18186842445520957
49 7.218950623550748 rho = 0.18157230414317316
50 7.214341687247792 rho = 0.18169383402975492
51 7.215248513141146 rho = 0.1816698925903335
52 7.213744550227628 rho = 0.18170959084075555
53 7.219696545948612 rho = 0.1815524462899758
54 7.183039110133378 rho = 0.18252066668288136
55 8.855899285982632 rho = 0.1465219563985709
56 8.910037325100198 rho = 0.14557055028630828
```

```
57 9.130277029186168 rho = 0.14189125120502588
58 9.346534600304434 rho = 0.1384476208267047
59 9.546535308641085 rho = 0.1353933722130123
60 9.634140820667053 rho = 0.13411205462901243
61 9.662833445648493 rho = 0.13369704679192895
62 9.705829985065252 rho = 0.13308044039475497
63 9.598978431599614 rho = 0.1346215807312865
64 9.555975722268512 rho = 0.13525181654083274
65 9.569954508634046 rho = 0.13504612390437054
66 9.629582053421649 rho = 0.13417537331712337
67 9.821178985102637 rho = 0.13144986939982028
68 9.033893666679694 rho = 0.14337327031327418
69 9.25937724192234 rho = 0.1397225504037749
70 9.282976266660825 rho = 0.13934934993097325
71 9.447954928767388 rho = 0.13679852773933132
72 8.239504378166497 rho = 0.15771683496714983
73 8.433629688432678 rho = 0.15392082466594537
74 6.970400632487189 rho = 0.1876472291070952
75 5.097201944798465 rho = -0.20994009818369053
76 4.475502547629957 rho = -0.23098244275917246
77 8.397044050222046 rho = -0.12563028071426663
78 7.313400107843987 rho = -0.1408064383921115
79 10.310354419893324 rho = -0.1020675771488877
80 13.891563764037322 rho = -0.07687178945650697
81 22.338392395445513 rho = -0.04765437900267999
82 86.84711025590262 rho = -0.012485123871843307
83 38.06034323089395 rho = -0.027240331300606435
84 36.16705611352239 rho = -0.028627826817540836
85 97.25866320140607 rho = 0.010682655722015433
86 63.58385344876303 rho = 0.016395383837694677
87 62.98404599909872 rho = 0.016552311009105924
88 62.78692342922597 rho = 0.016604560700634874
89 55.758977064172356 rho = 0.018711682145757802
90 55.32556078558913 rho = 0.018859934239153553
91 54.96427277796414 rho = 0.01898538461970345
92 55.106014423474676 rho = 0.01893587988149692
93 55.10338277966862 rho = 0.018936795594938438
94 54.88988034810414 rho = 0.019011336147632063
95 54.77722920242416 rho = 0.019050898288105245
96 54.77401779557173 rho = 0.019052029394104553
97 54.730312953256814 rho = 0.019067434597923816
98 54.76272294402635 rho = 0.019055981848435932
99 54.78048843027843 rho = 0.019049710390428798
100 54.7925155281926 rho = 0.01904547409838786
```

Exercise #4: Harmonic Ritz generalized Davidson

[9]:
```
function HR_GD_sym(A,
                   q::Vector{Float64},
                   k::Int,
                   sigma::Float64,
                   tol::Float64;
                   T=I)
  n, _ = size(A)
  Q = zeros(Float64, (n, k))
  W = zeros(Float64, (n, k))
  H = zeros(Float64, (k, k))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  t .= q
  j = 0
  res = 1.; rho = 1.
  while (res > tol && j < k)
    j += 1
    w = (A - sigma * I) * t
    for i in 1:j
      witw = W[:, i]'w
      w .-= witw * W[:, i]
      t .-= witw * Q[:, i]
    end
    w_norm = norm(w)
    W[:, j] = w / w_norm
    Q[:, j] = t / w_norm
    H[j, 1:j] = W[:, j]'Q[:, 1:j]
    H[1:j, j] = W[:, 1:j]'Q[:, j]
    vals, vecs = eigen(H[1:j, 1:j])
    vals = 1. ./ vals
    ind = argmin(norm.(vals))
    y .= Q[:, 1:j] * vecs[:, ind]
    yty = y'y
    y ./= sqrt(yty)
    drho = 1. / (vals[ind] * yty)
    rho = sigma + drho
    r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - drho * y
    res = norm(r) / norm(rho)
    t .= T \ r
    println(j, ' ', res, " rho = ", rho)
  end
  return res, rho, y
end;
```

[10]:
```
sigma = 2.;
P = ilu(A - sigma * I);

Random.seed!(1);
res, val, y = HR_GD_sym(A, rand(n), k, sigma, tol, T=P);
```

```
1 0.5216274696252504 rho = 5.754506744294758
2 0.0038650683397191033 rho = 1.9999508688776502
3 5.3244384752941755e-6 rho = 1.9999811874462514
4 2.3799647481973046e-6 rho = 1.9999820396357997
5 2.004744252775856e-6 rho = 1.9999839750160375
6 7.189217607108482e-7 rho = 1.9999847004451703
7 1.2172513393817337e-7 rho = 1.999984712315548
8 7.090976830598623e-8 rho = 1.9999847153196997
9 2.8119364224033014e-8 rho = 1.9999847167865226
10 3.83615710946413e-9 rho = 1.999984716850456
```

Exercise #5: Rayleigh-Ritz Jacobi-Davidson

[11]:
```
function mul_tA(A, lbda, y, x, tAx)
  c = y'x
  tAx[:] = x - c * y
  tAx[:] = (A - lbda * I) * tAx
  c = y'tAx
  tAx[:] -= c * y
end;

mutable struct OrthoPrecond
  T
  y::Vector{Float64}
end;

function LinearAlgebra.ldiv!(M::OrthoPrecond, R)
  u = M.T \ M.y
  V = M.T \ R
  alpha = - M.y'V / M.y'u
  R[:] = u * alpha + V
end;

function LinearAlgebra.ldiv!(Z, M::OrthoPrecond, R)
  ldiv!(M, R)
  Z .= R
end;
```

[12]:
```
function RR_JD_sym(A,
                   q::Vector{Float64},
                   k::Int,
```

```
                    tol::Float64;
                    T=I,
                    gmres_iters=10)
  n, _ = size(A)
  Q = zeros(Float64, (n, k))
  W = zeros(Float64, (n, k))
  B = zeros(Float64, (k, k))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  t .= q
  j = 0
  if T != I
    M = OrthoPrecond(T, y)
  end
  res = 1.; rho = 1.
  while (res > tol && j < k)
    for i in 1:j
      h = Q[:, i]'t
      t .-= h .* Q[:, i]
    end
    Q[:, j+1] = t ./ norm(t)
    j += 1
    W[:, j] = A * Q[:, j]
    B[1:j, j] = Q[:, 1:j]'W[:, j]
    B[j, 1:j-1] = Q[:, j]'W[:, 1:j-1]
    vals, vecs = eigen(B[1:j, 1:j], sortby=norm)
    rho = vals[j]
    y .= Q[:, 1:j] * vecs[:, j]
    r .= W[:, 1:j] * vecs[:, j] - rho .* y
    res = norm(r) / norm(rho)
    tA = LinearOperator(Float64, n, n, false, false,
                        (tAx, x) -> mul_tA(A, rho, y, x, tAx),
                        nothing, nothing)
    if T == I
      t = gmres(tA, -r, maxiter=gmres_iters)
    else
      M.y =  y
      t = gmres(tA, -r, Pl=M, maxiter=gmres_iters)
    end
    println(j, ' ', res, " rho = ", rho)
  end
  return res, rho, y
end;
```

[13]:
```
Random.seed!(1);
val = power_iters(A, rand(n));
```

```
P = ilu(A - val * I);

res, val, y = RR_JD_sym(A, rand(n), k, tol, T=P);
```

```
1 0.5652921354850687 rho = 5.502991844091799
2 0.0033175957149939113 rho = 7.391537105460751
3 0.002057798555631144 rho = 7.391706673521782
4 0.0007844649546339336 rho = 7.39173814131558
5 0.0005358775162960447 rho = 7.391762723064658
6 0.001186132296158454 rho = 7.391903942187004
7 0.0010817779356960312 rho = 7.391919900326394
8 0.0019331629957956655 rho = 7.391956871252985
9 0.0019027150798482722 rho = 7.392128111548407
10 0.0015949167705126104 rho = 7.392279931046083
11 0.0017076087543885549 rho = 7.392544616475645
12 0.001564985361676476 rho = 7.392548751479266
13 0.004832821701236848 rho = 7.393159777786551
14 0.004171303496598973 rho = 7.393409455410072
15 0.0041367498428043345 rho = 7.393409534999574
16 0.005980062412619207 rho = 7.394244101027532
17 0.005749784801209879 rho = 7.394415111371908
18 0.006058498665077307 rho = 7.395017806928446
19 0.005726430826767345 rho = 7.395141061017928
20 0.005424564989312227 rho = 7.39561751255829
21 0.005642360713682617 rho = 7.395634747201311
22 0.007173401868196142 rho = 7.39628350476621
23 0.00687114988893327 rho = 7.396991842072098
24 0.00696702489203164 rho = 7.397214952177624
25 0.008456849435186882 rho = 7.398154959474748
26 0.008719306913956854 rho = 7.398478575977707
27 0.008571211779350899 rho = 7.398524305815906
28 0.010263648043788608 rho = 7.399472954507454
29 0.009900048202727423 rho = 7.399602847212132
30 0.009159128728926815 rho = 7.399714236980546
31 0.009457679582838842 rho = 7.400235634988883
32 0.01026831636704352 rho = 7.400924071347117
33 0.010230607222223478 rho = 7.401003102627257
34 0.012075700211570872 rho = 7.401946510160052
35 0.011503903266781855 rho = 7.402166512623031
36 0.010929635578134226 rho = 7.402430047686826
37 0.011333474037928884 rho = 7.403063314195104
38 0.011335644518837016 rho = 7.403226819037827
39 0.012456965479002024 rho = 7.40418088257034
40 0.013621363598054534 rho = 7.404835689059412
41 0.013215903268617938 rho = 7.405089151189049
42 0.01405541063690538 rho = 7.4062834503846355
43 0.013394386672385251 rho = 7.406423107749321
44 0.013028150679327923 rho = 7.407056764019251
```

```
45 0.01360876243449039 rho = 7.407637303504715
46 0.013623485318783327 rho = 7.407723358536503
47 0.015027176532447123 rho = 7.408921983906307
48 0.014139702910210484 rho = 7.409496353824187
49 0.013777999764934809 rho = 7.409637673667482
50 0.014393754208969548 rho = 7.410485707873169
51 0.014723053396013031 rho = 7.410894935866558
52 0.014792794572906446 rho = 7.411093720721708
53 0.016287635614256484 rho = 7.412874915949798
54 0.016854428190127755 rho = 7.413831098938606
55 0.01685358342830743 rho = 7.414255450046656
56 0.017215405050056275 rho = 7.415414227457404
57 0.0174111818534019 rho = 7.416646787810145
58 0.016876468599087324 rho = 7.417245118188839
59 0.01649914849424265 rho = 7.418126620089094
60 0.0162144296603174 rho = 7.418204457758628
61 0.016691229062988727 rho = 7.419604678232487
62 0.018489408001291337 rho = 7.421216641435877
63 0.01894844790201179 rho = 7.421802869231708
64 0.01856421850295231 rho = 7.4224108010029095
65 0.01897023813975275 rho = 7.422952713601991
66 0.019487975343356993 rho = 7.423596706645892
67 0.0198559138575221 rho = 7.425037750686098
68 0.019757108243540527 rho = 7.426251329050177
69 0.01958825194088421 rho = 7.4262936837854925
70 0.02037764868839678 rho = 7.428287849967919
71 0.021172981173474013 rho = 7.429380901257804
72 0.020843195792673384 rho = 7.429788362474755
73 0.02240638785683264 rho = 7.43140139680824
74 0.02183042391839153 rho = 7.431566147817806
75 0.022488336616019802 rho = 7.43291549411661
76 0.021680363500748822 rho = 7.433270909817994
77 0.022713899877789358 rho = 7.4349041853634334
78 0.022208349837063712 rho = 7.435095515610255
79 0.02239965313796431 rho = 7.436713952236634
80 0.021831351786977062 rho = 7.4375361252968935
81 0.021867002165216903 rho = 7.438626486097317
82 0.02282695508889656 rho = 7.440596513366156
83 0.022511124249387712 rho = 7.440689555262254
84 0.023467346881393028 rho = 7.441910763303527
85 0.023680740188086552 rho = 7.442527038462964
86 0.02329078172164982 rho = 7.442614421982655
87 0.024559749726536306 rho = 7.444864066280616
88 0.023285832798238922 rho = 7.445637340121496
89 0.023794859674127384 rho = 7.447641727273936
90 0.023453603090797626 rho = 7.447795439794129
91 0.024233766670172276 rho = 7.449730282694336
92 0.02394155170711315 rho = 7.45060623780971
```

```
93 0.023570839530956515 rho = 7.450727185287562
94 0.02456319237852761 rho = 7.452593916495262
95 0.025387671881390482 rho = 7.453464232778004
96 0.026016478058393074 rho = 7.454853739350041
97 0.025168000891062185 rho = 7.455286632922486
98 0.025008434907405726 rho = 7.457266332004305
99 0.024902897333059823 rho = 7.457860119465265
100 0.024606715298321032 rho = 7.459226169375764
```

Exercise #6: Harmonic Ritz Jacobi-Davidson

[14]:
```
function HR_JD_sym(A,
                   q::Vector{Float64},
                   k::Int,
                   sigma::Float64,
                   tol::Float64;
                   T=I,
                   gmres_iters=10)
  n, _ = size(A)
  Q = zeros(Float64, (n, k))
  W = zeros(Float64, (n, k))
  H = zeros(Float64, (k, k))
  y = zeros(Float64, n)
  r = zeros(Float64, n)
  t = zeros(Float64, n)
  t .= q
  j = 0
  if T != 0
    M = OrthoPrecond(T, y)
  end
  res = 1.; rho = 1.
  while (res > tol && j < k)
    j += 1
    w = (A - sigma * I) * t
    for i in 1:j
      witw = W[:, i]'w
      w .-= witw * W[:, i]
      t .-= witw * Q[:, i]
    end
    w_norm = norm(w)
    W[:, j] = w / w_norm
    Q[:, j] = t / w_norm
    H[j, 1:j] = W[:, j]'Q[:, 1:j]
    H[1:j, j] = W[:, 1:j]'Q[:, j]
    vals, vecs = eigen(H[1:j, 1:j])
    vals = 1. ./ vals
    ind = argmin(norm.(vals))
```

```
    y .= Q[:, 1:j] * vecs[:, ind]
    yty = y'y
    y ./= sqrt(yty)
    drho = 1. / (vals[ind] * yty)
    rho = sigma + drho
    r .= W[:, 1:j] * vecs[:, ind] / sqrt(yty) - drho * y
    res = norm(r) / norm(rho)
    tA = LinearOperator(Float64, n, n, false, false,
                        (tAx, x) -> mul_tA(A, rho, y, x, tAx),
                        nothing, nothing)
    if T == I
      t = gmres(tA, -r, maxiter=gmres_iters)
    else
      M.y =  y
      t = gmres(tA, -r, Pl=M, maxiter=gmres_iters)
    end
    println(j, ' ', res, " rho = ", rho)
  end
  return res, rho, y
end;
```

[15]:
```
sigma = 2.;
P = ilu(A - sigma * I);

Random.seed!(1);
res, val, y = HR_JD_sym(A, rand(n), k, sigma, tol, T=P);
```

```
1 0.5216274696252504 rho = 5.754506744294758
2 0.014267120749643437 rho = 1.9995251745301632
3 7.697774449473055e-5 rho = 1.9998485936854165
4 1.911987835945399e-5 rho = 1.999851026701978
5 1.8032694842696706e-5 rho = 1.9998499907841842
6 1.8221443245487095e-5 rho = 1.9998859692691207
7 3.5233097961482445e-6 rho = 1.9998854180449723
8 8.253545132492748e-6 rho = 1.9998883135381198
9 3.1838760801042937e-6 rho = 1.999893420069769
10 3.126585964313475e-6 rho = 1.999894332670403
11 3.857042083184614e-6 rho = 1.9998971693108225
12 2.1115587974583106e-6 rho = 1.9998981296555
13 5.466784044694908e-7 rho = 1.9998985961257225
14 1.1180512548340814e-7 rho = 1.9998985949638861
15 2.1907769592383254e-8 rho = 1.999898595402957
16 4.384128117250179e-9 rho = 1.9998985953969408
```