CSCI 5304 HW 5

Jingxiang Li

December 3, 2014

Problem 1

Suppose the eigenvalues of a real matrix A of largest absolute value are a complex conjugate pair, and that all the other eigenvalues are strictly smaller in absolute value. How does the power method behave when started with a real vector? Show the power method yields an approximate basis for the invariant subspace spanned by the two eigenvectors of those two largest eigenvalues.

Solution

Let $\lambda_1 \dots \lambda_n$ be n eigenvalues of a real matrix A, and $u_1 \dots u_n$ be the corresponding eigenvectors. Suppose λ_1, λ_2 are the two largest eigenvalues, which is also a complex conjugate pair with largest absolute value among all other eigenvalues.

Then let's consider the power method. let v be any initial vector for the power method, then we can represent v as a linear combination of A's eigenvectors, that is

$$v = a_1 u_1 + a_2 u_2 + \cdots + a_n u_n$$

When power method started with vector v, we have

$$\begin{aligned} v := Av &= a_1 A u_1 + a_2 A u_2 + \cdots + a_n A u_n \\ &= a_1 \lambda_1 u_1 + a_2 \lambda_2 u_2 + \cdots + a_n \lambda_n u_n \end{aligned}$$

Then after k times iteration, we have

$$\begin{aligned} v := Av &= a_1 A^k u_1 + a_2 A^k u_2 + \cdots + a_n A^k u_n \\ &= a_1 \lambda_1^k u_1 + a_2 \lambda_2^k u_2 + \cdots + a_n \lambda_n^k u_n \end{aligned}$$

Since λ_1 and λ_2 is a complex conjugate pair with largest absolute value among all other eigenvalues, all other λ_j^k, $\forall j! = 1, 2$ can be ignored. Thus

$$v := a_1 \lambda_1^k u_1 + a_2 \lambda_2^k u_2$$

which is the invariant subspace spanned by the two eigenvectors of those two largest eigenvalues.

Problem 2

What are the eigenvalues and eigenvectors of a Householder reflection $P = I - 2uu'$ and a 2×2 Givens rotation $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$?

Solution

For $P = I - 2uu'$, consider $v = u$ we have $Pu = u - 2u = -u$, suggesting that u is an eigenvector with -1 as the corresponding eigenvalue. Then consider any vector v perpendicular to vector u, we have $Pv = v$, which suggests that v is an eigenvector with 1 as the corresponding eigenvalue.

Thus consider $(v_1, v_2, \ldots, v_{n-1})$ be the basis of the subspace perpendicular to u
the eigenvector matrix is

$$\begin{bmatrix} u & v_1 & v_2 & \cdots & v_{n-1} \end{bmatrix}$$

with the corresponding eigenvalues $(-1, 1, \ldots, 1)$.

For G a Givens rotation $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$

First consider $u = (1, i)'$, then $Gu = (c + si)u$; then consider $v = (1, -i)'$, then $Gv = (c - si)v$. The the eigenvector matrix of G is

$$\begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix}$$

with eigenvalues $(c + si, c - si)$

Problem 3

For each of the following upper triangular matrices, list all the eigenvalues together with their algebraic and geometric multiplicities. Solve for all the corresponding eigenvectors, scaled so that their largest component in absolute value is a 1. (Hand calculation).

Solution

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, D = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

for matrix A, eigenvalues are $(1, 2)'$, whose algebraic multiplicities are respectively 2 and 1, and geometric multiplicities are 1 and 1.

$$\text{Eigenvectors} \begin{bmatrix} 1 & 1 \\ 0 & 0.5 \\ 0 & 0.5 \end{bmatrix}$$

for matrix B, eigenvalues are $(1, 2)'$, whose algebraic multiplicities are respectively 2 and 1, and geometric multiplicities are 2 and 1.

$$\text{Eigenvectors} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

for matrix C, eigenvalues is $(1)'$, whose algebraic multiplicity is 3, and geometric multiplicity is 2.

$$\text{Eigenvectors} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

for matrix D, eigenvalues are $(0, 1, 2)'$, whose algebraic multiplicities are respectively 1, 1 and 1, and geometric multiplicities are 1, 1 and 1.

$$\text{Eigenvectors} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Problem 4

Generate a random $n \times n$ symmetric matrix A with eigenvalues $1, \ldots, n-2, 2n-1, 2n$ using the following matlab commands:

- `[Q,R] = qr(rand(n,n));`
- `A = Q * diag([1:n-2,2*n-1,2*n]) * Q';`

Use this matrix as test case for the following algorithms. For each algorithm, show the code [fragment] and give an outline of the method (if not self-explanatory from the code fragment). find how many steps it takes to find a eigenvalue/vector pair within an accuracy of 10^{-7} . You should check the accuracy by computing the residual, not by comparing to the true answer. Use $n = 5, 10$ to test your algorithm, and then use $n = 40$ to report the number of iterations needed.

Problem a

Power method starting with a random starting vector. At each iteration compute and save the Rayleigh quotient, then plot the difference between the values of these Rayleigh quotients and the final value (use `semilogy`). What is the rate of convergence? Did the power method converge to the largest eigenvalue? Can the rate of convergence give an indication of the second largest eigenvalue? Hint: since this matrix is symmetric, you might need to take the square root of the rate of convergence. Do not go over 1,000 iterations.

Solution

Here we first create a function `powerMethod`, specific details can be found in the following code chunk.

```
%% Power Method to compute the largest eigenvector for matrix A
%% Input matrix A; maximum iteration times t_max; eps0
%% Output eigenvector eigVec; eigenvalue eigVal; number of iteration
   iterNum; array of residuals for each iteration resArray
%% Output Rayleigh quotient for each iteration Rq

function [eigVec eigVal iterNum resArray Rq] = powerMethod(A, iterMax,
   eps0)
  [n m] = size(A);
  eigVec = (rand(n, 1) - 0.5) * 2;
  [tmp1 tmp2] = max(abs(eigVec));
  eigVec = eigVec ./ eigVec(tmp2);
  eigVal = eigVec' * A * eigVec / (eigVec' * eigVec);
  iterNum = 1;

  resArray = zeros(iterMax, 1);
  Rq = zeros(iterMax, 1);
```

```
  resArray(iterNum) = norm(A * eigVec - eigVal * eigVec, 1);
  Rq(iterNum) = eigVal;

  while (resArray(iterNum) > eps0 && iterNum < iterMax)
    eigVec = A * eigVec;
    [tmp1 tmp2] = max(abs(eigVec));
    eigVec = eigVec ./ eigVec(tmp2);
    eigVal = eigVec' * A * eigVec / (eigVec' * eigVec);
    iterNum = iterNum + 1;
    resArray(iterNum) = norm(A * eigVec - eigVal * eigVec, 1);
    Rq(iterNum) = eigVal;
  end
end
```

Then we apply powerMethod to solve the eigenvalue problem.

```
n = 40
[Q,R] = qr(rand(n,n));
A = Q * diag([1:n-2,2*n-1,2*n]) * Q';
iterMax = 1000;
eps0 = 1e-07;
[eigVec eigVal iterNum resArray Rq] = powerMethod(A, iterMax, eps0);

eigVal
% > eigVal =  80.000
iterNum
% > 1000
resArray(iterNum) / resArray(iterNum - 1)
% > 0.98750

semilogy(1:iterNum, abs(Rq(1:iterNum) - eigVal));
hold on;
xlabel('iterNum');ylabel('log(rayleighQuotient - finalValue)');title('
   log(rayleighQuotient - finalValue) VS iterNum');
fontsize=16;
set([gca; findall(gca, 'Type','text')], 'FontSize', fontsize);
set([gca; findall(gca, 'Type','line')], 'linewidth', 3);
saveas(1, 'p1.pdf');
hold off;
```

Note that the algorithm does not converge within 1000 steps.

First, Let's see the graph 1 which is about difference between the values of these Rayleigh quotients in each iteration and the final value.

Next, we see that the convergence rate estimated by the the ratio of residuals is 0.98750, which is exactly equivalent to the ratio of the second largest eigenvalue over the largest eigenvalue.

Then, the power method converge to the largest eigenvalue.

Lastly, since we know the convergence rate is equal to the ratio of the second largest eigenvalue over the largest eigenvalue, we can estimate the second largest eigenvalue by calculating the product of the convergence rate and the largest eigenvalue, which is $80 \times 0.98750 = 79$.

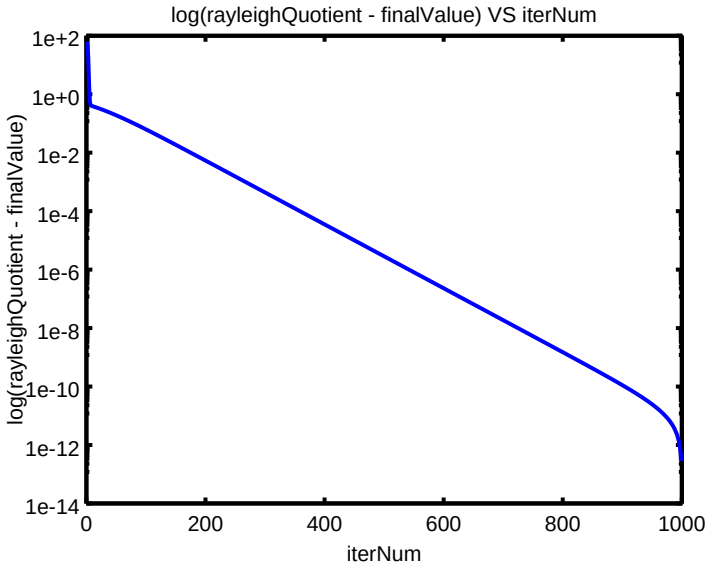

Figure 1: log(rayleighQuotient - finalValue) VS iterNum

Problem b

Rayleigh Quotient Iteration (RQI: inverse iteration with the shift at each step determined by the Rayleigh quotient with the iterate). Start with a random normalized vector. Which eigenvalue does it converge to? To force it to converge to the largest eigenvalue, apply RQI with a shift fixed at A_∞ for a few initial iterations before switching to the pure Rayleigh quotient. How many initial iterations with fixed shifts do you need to make it converge to the largest eigenvalue? This number should be less than 10.

Solution

First we create a function `QRI` to implement the naive Qayleigh Quotient Iteration algorithm, and then we apply it to the test matrix.

```
%% QRI Rayleigh Quotient Iteration to get the largest eigenvalue
%% Input matrix A; maximum iteration times t_max; eps0
%% Output eigenvector eigVec; eigenvalue eigVal; number of iteration
   iterNum; array of residuals for each iteration resArray

function [eigVec eigVal iterNum resArray] = QRI(A, iterMax, eps0)
  [n m] = size(A);
  eigVec = (rand(n, 1) - 0.5) * 2;
  [tmp1 tmp2] = max(abs(eigVec));
  eigVec = eigVec ./ eigVec(tmp2);
  eigVal = eigVec' * A * eigVec / (eigVec' * eigVec);
```

```
  B = (A - eigVal .* eye(size(A)))^(-1);
  iterNum = 1;
  resArray = zeros(iterMax, 1);
  resArray(iterNum) = norm(A * eigVec - eigVal * eigVec, 1);

  while (resArray(iterNum) > eps0 && iterNum < iterMax)
      eigVec = B * eigVec;
      [tmp1 tmp2] = max(abs(eigVec));
      eigVec = eigVec ./ eigVec(tmp2);
      eigVal = eigVec' * A * eigVec / (eigVec' * eigVec);
      B = (A - eigVal .* eye(size(A)))^(-1);
      iterNum = iterNum + 1;
      resArray(iterNum) = norm(A * eigVec - eigVal * eigVec, 1);
  end
end

n = 40
[Q,R] = qr(rand(n,n));
A = Q * diag([1:n-2,2*n-1,2*n]) * Q';
iterMax = 1000;
eps0 = 1e-07;
[eigVec eigVal iterNum resArray] = QRI(A, iterMax, eps0);

eigVal
% > 22.000
iterNum
% > 5
```

Here we see the algorithm converges within 5 steps. Note that the eigenvalue Rayleigh Quotient Iteration converges to is 22, not the largest one. In fact theoretically it's hard to say which eigenvalue the algorithm will converge to.

Next, we create a function `QRI_f` to implement the modified Rayleigh Quotient Iteration algorithm, which fix the shift at $||A||_\infty$ for a few initial iterations before switching to the pure Rayleigh quotient.

```
%% QRI_f Modified Rayleigh Quotient Iteration to get the largest
   eigenvalue
%% Input matrix A; maximum iteration times t_max; eps0, iterFix
%% Output eigenvector eigVec; eigenvalue eigVal; number of iteration
   iterNum; array of residuals for each iteration resArray

function [eigVec eigVal iterNum resArray] = QRI_f(A, iterMax, eps0,
   iterFix)
  fixShift = norm(A, inf);
  [n m] = size(A);
  eigVec = (rand(n, 1) - 0.5) * 2;
  [tmp1 tmp2] = max(abs(eigVec));
  eigVec = eigVec ./ eigVec(tmp2);
```

```
  eigVal = eigVec' * A * eigVec / (eigVec' * eigVec);
  B = (A - fixShift .* eye(size(A)))^(-1);
  iterNum = 1;
  resArray = zeros(iterMax, 1);
  resArray(iterNum) = norm(A * eigVec - eigVal * eigVec, 1);
  while (resArray(iterNum) > eps0 && iterNum < iterMax)
    eigVec = B * eigVec;
    [tmp1 tmp2] = max(abs(eigVec));
    eigVec = eigVec ./ eigVec(tmp2);
    eigVal = eigVec' * A * eigVec / (eigVec' * eigVec);
    if (iterNum < iterFix)
      B = (A - fixShift .* eye(size(A)))^(-1);
    else
      B = (A - eigVal .* eye(size(A)))^(-1);
    end
    iterNum = iterNum + 1;
    resArray(iterNum) = norm(A * eigVec - eigVal * eigVec, 1);
  end
end

n = 40;
iterFix = 10;
[Q,R] = qr(rand(n,n));
A = Q * diag([1:n-2,2*n-1,2*n]) * Q';
iterMax = 1000;
eps0 = 1e-07;
[eigVec eigVal iterNum resArray] = QRI_f(A, iterMax, eps0, iterFix);

eigVal
% > 79
iterNum
% > 14
```

Note that the algorithm converges within 14 steps.

Actually it's hard to say how many initial iterations with fixed shifts is needed to make it converge to the largest eigenvalue. We have tried different number of initial iterations and find that no matter which value you choose, you can't avoid the possibility that the algorithm may converge to some other eigenvalues. However, as the number of initial iterations grows up, the algorithm is more likely to converge to large eigenvalues, like 79 and 80 in this example.

Problem c

Simple explicit QR algorithm (no shifts). Try symmetrizing after each iteration. Allow up to 1000 iterations.

Solution

Here we create the function `QRAlgo` to implement the explicit QR algorithm.

```
%% QR Algorithm to solve the eigenvalue eigenvector problem
function [eigVec eigVal iterNum resArray] = QRAlgo(A, iterMax, eps0)
  U = eye(size(A));
  iterNum = 1;
  resArray = zeros(iterMax, 1);
  A0 = A;
  [Q R] = qr(A);
  A = R * Q;
  U = U * Q;

  resArray(iterNum) = norm(diag(A0 * U - diag(diag(A)) * U), 1);
  while (resArray(iterNum) > eps0 && iterNum < iterMax)
    [Q R] = qr(A);
    A = R * Q;
    U = U * Q;
    iterNum = iterNum + 1;
    resArray(iterNum) = norm(diag(A0 * U - diag(diag(A)) * U), 1);
  end

  eigVal = A;
  eigVec = U;
end

n = 40;
[Q,R] = qr(rand(n,n));
A = Q * diag([1:n-2,2*n-1,2*n]) * Q';
iterMax = 1000;
eps0 = 1e-07;
[eigVec eigVal iterNum resArray] = QRAlgo(A, iterMax, eps0);

iterNum
% > 1000
```

Note that the QR algorithm does not converge within 1000 steps.

Problem 5

Implement a simple Lanczos procedure (as given in the lecture notes, or Alg. 10.1.1 in the text or Alg. 36.1 in TB) to compute the factorization $AX = XT$ where A is a given symmetric matrix, X is a set of generated orthonormal columns, and T is a generated tridiagonal matrix. Apply the procedure to the test matrices in the previous question and use the result to obtain a Ritz approximation (ğ10.1.4) to the leading eigenvalue/vector pair for A. You should use the standard simple Lanczos procedure, except that you should check for loss of orthogonality by the following heuristic: check that each generated Lanczos vector is orthogonal to the starting vector (i.e., the inner product is less than the given tolerance). Carry out the Lanczos expansion until n Lanczos vectors have been generated, or until an off-diagonal element of T is less than 10^{-7} (in absolute value), or until orthogonality is lost (to the same tolerance). Use a random starting vector.

Compute the Ritz approximation to the eigenvalue at each iteration, and plot the differences between the Ritz values and the final value as done with the Power Method above. You can also just compute the maximum eigenvalue of the leading $k \times k$ principal submatrix of the final T, for $k = 1, 2, \ldots$ To obtain the Ritz values, just apply the built-in eig function to the tridiagonal matrix T .

Solution

We first create a function called `lanczosTri` to implement the Lanczos procedure.

```
%% lanczos Tridiagonalization, Given Matrix, AQ = QT, where Q is
   orthonormal and T is a tridiagonal Matrix
%% Input: n * n matrix A
%% output: orthonormal Q and tridiagonal T, s.t. AQ = QT

function [Q T] = lanczosTri (A, eps0)
    n = size(A, 1);
    beta = zeros(n + 1 ,1);
    alpha = zeros(n + 1, 1);
    Q = zeros(n, n + 1);
    R = Q;
    v = (rand(n, 1) - 0.5) * 2;
    v = v / norm(v, 2);

    k = 1;
    beta(1) = 1;
    R(:, 1) = v;

    while ((k == 1 || abs(beta(k)) > eps0) && k < n + 1)
        Q(:, k + 1) = R(:, k) / beta(k);
        k = k + 1;
        alpha(k) = Q(:, k)' * A * Q(:, k);
        R(:, k) = (A - alpha(k) * eye(size(A))) * Q(:, k) - beta(k - 1)
    * Q(:, k - 1);
```

```
        beta(k) = norm(R(:, k), 2);
        if abs((R(:, k) / beta(k))' * Q(:, 2)) > eps0
            break;
        end
    end

    Q = Q(:, 2 : (k));
    alpha = alpha(2 : (k));
    beta = beta(2 : (k - 1));

    T = zeros(k - 1);
    for i = 1 : k - 1
        T(i, i) = alpha(i);
        if (i < k - 1)
            T(i, i + 1) = beta(i);
            T(i + 1, i) = beta(i);
        end
    end
end
```

Then we try to apply function `lanczosTri` to the test matrix in the previous problem, compute the Ritz approximation to the eigenvalue at each iteration, and plot the differences between the Ritz values and the final value as done with the Power Method above.

```
n = 40;
[Q,R] = qr(rand(n,n));
A = Q * diag([1:n-2,2*n-1,2*n]) * Q';
eps0 = 1e-07;
[eigVec eigVal iterNum resArray] = powerMethod(A, iterMax, eps0);
[Q T] = lanczosTri(A, eps0);
res = zeros(size(T, 1), 1);
for i = 1:size(T, 1)
  eigeig = max(eig(T(1:i, 1:i)));
  res(i) = abs(eigeig - eigVal);
end
semilogy(1:size(T,1), res)
hold on;
xlabel('iterNum');ylabel('Difference');title('Difference VS iterNum');
fontsize=20;
set([gca; findall(gca, 'Type','text')], 'FontSize', fontsize);
set([gca; findall(gca, 'Type','line')], 'linewidth', 3);
saveas(1, 'p2.pdf');
hold off;
```

We can see the differences between the Ritz values and the final value as done with the Power Method above in graph 2.

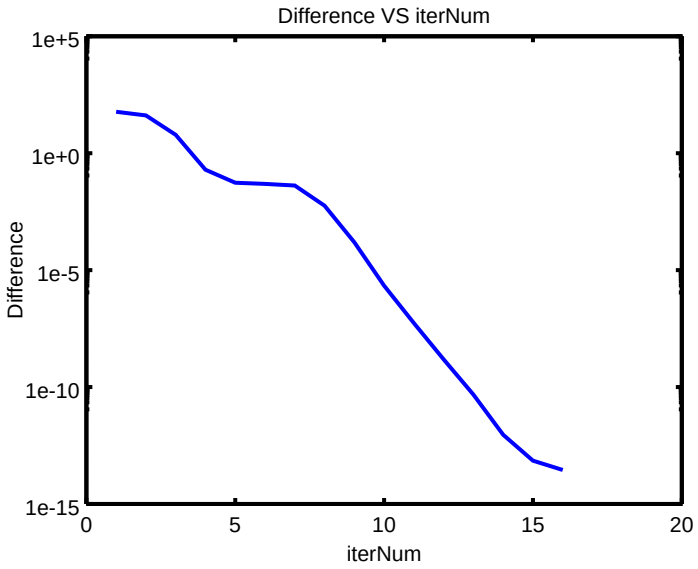

Figure 2: differences VS iterNum