6 Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors play important roles in a variety of applications. For example, a system of linear ordinary differential equations $\mathbf{y}'(t) = \mathbf{A}\mathbf{y}(t)$ can be transformed to the decoupled system $\mathbf{z}'(t) = \mathbf{\Lambda}\mathbf{z}(t)$, where $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ is an eigenvalue decomposition, and $\mathbf{z}(t) = \mathbf{V}^{-1}\mathbf{y}(t)$. An ideal spring-mass system with no damping can be modeled by an eigenvalue problem $\mathbf{K}\mathbf{v} = \lambda\mathbf{M}\mathbf{v}$, where $\sqrt{\lambda}$ is a resonant frequency. Eigenvalue problems also arise from linear stability analysis of steady-state solutions of dynamical systems, where the solution is linearly stable if all eigenvalues have negative real parts. A steady state of Markov chains is the eigenvector associated with its dominant eigenvalue 1. Time-independent Schrödinger equation is an eigenvalue problem where the eigenvalue describes the energy of stationary states. The Google PageRank values are the entries of the dominant right eigenvector of a modified adjacency matrix of the Webgraph. Eigenvalues are also used to find the roots of a polynomial. If

$$p(x) = c_0 + c_1 x + c_2 x^2 + ... + c_{n-1} x^{n-1} + x^n,$$

then the roots of p are the same as the eigenvalues of the **companion matrix**

$$\begin{pmatrix} 0 & 0 & ... & ... & 0 & -c_0 \\ 1 & 0 & 0 & ... & 0 & -c_1 \\ 0 & 1 & 0 & ... & 0 & -c_2 \\ : & : & : & : & : & : \\ 0 & ... & ... & 0 & 1 & -c_{n-1} \end{pmatrix}.$$

Since root formulas for quadratic, cubic, and quartic equations are susceptible to catastrophic round-off error, finding roots as eigenvalues of a companion matrix is usually a more reliable approach. Moreover, as there is no formula for higher order polynomial roots, finding the roots as eigenvalues circumvents this problem.

6.1 Theoretical background

We now give just a brief review of the mathematical problem, since it is assumed that students in this course have already completed a linear algebra course.

An eigenvalue-eigenvector pair (or eigenpair) of a matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$ is a scalar λ and vector $\mathbf{v} \in \mathbb{C}^n \setminus \{0\}$ satisfying

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v},$$

or equivalently, $(\lambda\mathbf{I}-\mathbf{A})\mathbf{v}=\mathbf{0}$. In other words, λ is a scalar for which $\lambda\mathbf{I}-\mathbf{A}$ is a singular matrix, and $\mathbf{v}\in\text{null}(\lambda\mathbf{I}-\mathbf{A})$. Eigenvectors can be scaled by any nonzero factor, but are usually normalized to have unit 2-norm.

Since $\lambda\mathbf{I}-\mathbf{A}$ is singular with an eigenvalue λ, $\det(\lambda\mathbf{I}-\mathbf{A})=0$. That is, eigenvalues are the roots of the *characteristic polynomial* $p_n(\lambda)\equiv\det(\lambda\mathbf{I}-\mathbf{A})$, whose coefficient for λ^n is 1. If $\mathbf{A}$ is real, $\det(\lambda\mathbf{I}-\mathbf{A})$ has all real coefficients, and so the eigenvalues are either real or are complex conjugate pairs. The set of all eigenvalues of $\mathbf{A}$ is called the *spectrum*, denoted as $\Lambda(\mathbf{A})$. It can be shown that $\det(\mathbf{A})=\prod_{i=1}^{n}\lambda_i$ and $\text{trace}(\mathbf{A})=\sum_{i=1}^{n}\lambda_i$.

Eigenvalue algorithms for general matrices of order ≥ 5 are necessarily iterative, because there is no root formula for general polynomials of order ≥ 5.

Diagonalizability. The algebraic multiplicity $alg_A(\lambda)$ of an eigenvalue λ is the smallest integer m such that $p_n^{(m)}(\lambda)\neq 0$, and the geometric multiplicity $geo_A(\lambda)=\dim\text{null}(\lambda\mathbf{I}-\mathbf{A})$. An eigenvalue λ is simple if $alg_A(\lambda)=geo_A(\lambda)=1$, semi-simple if $alg_A(\lambda)=geo_A(\lambda)>1$, and defective if $alg_A(\lambda)>geo_A(\lambda)$ (note that $alg_A(\lambda)\geq geo_A(\lambda)$ always).

A matrix $\mathbf{A}$ is *diagonalizable* if it has no defective eigenvalues, i.e., $\mathbf{A}\mathbf{v}_i=\lambda_i\mathbf{v}_i$ $(1\leq i\leq n)$ or $\mathbf{A}=\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ where $\mathbf{V}=[\mathbf{v}_1,\ldots,\mathbf{v}_n]$ and $\mathbf{\Lambda}=\text{diag}(\lambda_1,\ldots,\lambda_n)$; otherwise it is *defective* (non-diagonalizable) and has a Jordan canonical form. We only consider diagonalizable matrices in this chapter.

We call $\mathbf{A}$ a *normal* matrix if $\mathbf{A}^H\mathbf{A}=\mathbf{A}\mathbf{A}^H$. Examples include real symmetric ($\mathbf{A}^T=\mathbf{A}$), complex Hermitian ($\mathbf{A}^H=\mathbf{A}$), real skew-symmetric ($\mathbf{A}^T=-\mathbf{A}$) and complex skew-Hermitian ($\mathbf{A}^H=-\mathbf{A}$) matrices. All eigenvalues of the first two classes of matrices are real, and those of the last two classes are pure imaginary or zero. Normal matrices have a complete set of orthonormal eigenvectors, such that $\mathbf{A}=\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}=\mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$ or $\mathbf{A}=\mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$.

Two matrices $\mathbf{A}$ and $\mathbf{B}$ are called *similar* if there exists a nonsingular matrix $\mathbf{V}$ such that $\mathbf{A}=\mathbf{V}\mathbf{B}\mathbf{V}^{-1}$. Similar matrices have the same eigenvalues.

6.2 Single-vector iterations

For certain applications, we may require only one eigenpair of a matrix. Single-vector iterations are particularly suitable for this purpose. If used appropriately, these methods can be a simple and elegant solution; however, one needs to understand the conditions under which they converge robustly and rapidly.

Power Method. The power method is probably the most well-known eigenvalue algorithm, aiming to compute the dominant eigenpair (largest eigenvalue in modulus) of a matrix. If the dominant eigenvalue is unique, this method is essentially guaranteed to converge. Starting with an initial eigenvector approxi-

mation $\mathbf{x}_0$ (can be set as a random vector) normalized in 2-norm, this algorithm in each iteration multiplies $\mathbf{x}_k$ by $\mathbf{A}$ and then normalizes the new approximation $\mathbf{x}_{k+1}$. Let $\mu_k = \mathbf{x}_k^H \mathbf{A}\mathbf{x}_k$ be the Rayleigh quotient associated with $\mathbf{x}_k$. Once the condition $\frac{\|\mathbf{A}\mathbf{x}_k - \mu_k \mathbf{x}_k\|_2}{\|\mathbf{A}\mathbf{x}_k\|_2} \leq tol$ is satisfied for some small tolerance $tol > 0$, the method terminates; the final $\mathbf{x}_k$ is the approximate dominant eigenvector, and the Rayleigh quotient μ_k is the approximate dominant eigenvalue.

Power Method for computing the dominant eigenpair of $\mathbf{A}$

Choose $tol > 0$, $\mathbf{x}_0 \in \mathbb{C}^n$ **with** $\|\mathbf{x}_0\|_2 = 1$.

For $k = 0, 1, \ldots,$ **until convergence**

$\mu_k = \mathbf{x}_k^H \mathbf{A}\mathbf{x}_k$; $\quad \mathbf{r}_k = \mathbf{A}\mathbf{x}_k - \mu_k \mathbf{x}_k$;

If $\frac{\|\mathbf{r}_k\|_2}{\|\mathbf{A}\mathbf{x}_k\|_2} \leq tol$, **exit;**

$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$;

$\mathbf{x}_{k+1} = \mathbf{x}_{k+1}/\|\mathbf{x}_{k+1}\|_2$;

End For

The MATLAB code for this algorithm is given below.

```
function [eval,evec,itcount] = PowerMethod(A, x0, tol)

% normalize initial condition
x = x0/norm(x0);

% keep track of Rayleigh quotient as approx of lambda
rq = x' * (A*x);

itcount=0;
while norm( A*x - rq*x)/norm(A*x) >tol
    itcount=itcount+1;

    y = A*x;
    x = y / norm(y);

    rq = x' * (A*x) / (x'*x);
end
eval=rq;
evec=x;
```

Example 47. *Find the dominant eigenvalue of the matrix*

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 1 \\ 2 & 6 & 2 \end{pmatrix}.$$

We run the power method code above with initial guess $[1;2;3]$*, via the following commands:*

```
>> A = [1, 2, 3; 1, 3, 1; 2, 6, 2];
>> [l,x,itcount]=PowerMethod(A,[1;2;3],1e-6)

l =
   6.4641e+00

x =
   5.4779e-01
   3.7415e-01
   7.4829e-01

itcount =
     6

>> norm(A*x - l*x)

ans =
   2.6768e-07

>> eig(A)

ans =
   6.4641e+00
  -4.6410e-01
  -7.4015e-16
```

Observe that the algorithm converges in 6 *iterations to the eigevalue* $\lambda = 6.4641$ *and associated eigenvector. As a sanity check, we make sure* $\mathbf{Ax} \approx \lambda\mathbf{x}$*. We use the* `eig` *function (discussed more later in this chapter) as an additional sanity check:* `eig` *returns all the eigenvalues of a matrix, and we see the three eigenvalues of* $\mathbf{A}$ *are approximately* $6.4641, -0.4610$*, and* 0*. Thus the power method did in fact converge to the largest eigenvalue.*

We now discuss why the power method converges to the dominant eigenvalue of a matrix. We need three assumptions to do this analysis: first, we assume a

matrix $\mathbf{A}$ is diagonalizable (which is almost always true in practice). Second, we assume the matrix $\mathbf{A}$ has a unique dominant eigenvalue, i.e. we can order the eigenvalues of $\mathbf{A}$ as $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$. Let $\mathbf{v}_i$ be the eigenvector associated with λ_i such that $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$. The final assumption is that, as the initial guess $\mathbf{x}_0$ is written as a linear combination of all eigenvectors of A, the linear combination has a nonzero component of $\mathbf{v}_1$. This is also almost always true in practice.

Since $\mathbf{A}$ is diagonalizable, its eigenvectors form a basis for $\mathbb{C}^n$. Then we can decompose $\mathbf{x}_0$ in this basis:

$$\mathbf{x}_0 = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + ... + \alpha_n \mathbf{v}_n = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i,$$

where the α_i are (possibly complex) scalars. The power method at each iteration multiplies the previous iterate by $\mathbf{A}$, normalizes the resulting vector, and checks for convergence. Multiplying $\mathbf{x}_0$ by $\mathbf{A}$ yields

$$\mathbf{A}\mathbf{x}_0 = \mathbf{A} \sum_{i=1}^{n} \alpha_i \mathbf{v}_i = \sum_{i=1}^{n} \alpha_i \mathbf{A}\mathbf{v}_i,$$

where $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$. Thus we have

$$\mathbf{A}\mathbf{x}_0 = \sum_{i=1}^{n} \alpha_i \lambda_i \mathbf{v}_i,$$

and now applying the normalization, we obtain $\mathbf{x}_1$:

$$\mathbf{x}_1 = \frac{\mathbf{A}\mathbf{x}_0}{\|\mathbf{A}\mathbf{x}_0\|_2} = \frac{1}{\|\mathbf{A}\mathbf{x}_0\|_2} \sum_{i=1}^{n} \alpha_i \lambda_i \mathbf{v}_i.$$

Repeating this process yields for $\mathbf{x}_2$:

$$\mathbf{x}_2 = \frac{1}{\|\mathbf{A}\mathbf{x}_0\|_2} \frac{1}{\|\mathbf{A}\mathbf{x}_1\|_2} \sum_{i=1}^{n} \alpha_i \lambda_i^2 \mathbf{v}_i,$$

and for $\mathbf{x}_m$:

$$\mathbf{x}_m = \left(\Pi_{k=0}^{m-1} \frac{1}{\|\mathbf{A}\mathbf{x}_k\|_2} \right) \sum_{i=1}^{n} \alpha_i \lambda_i^m \mathbf{v}_i.$$

Here is the key point: Since $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$, then for m large enough,

$$\sum_{i=1}^{n} \alpha_i \lambda_i^m \mathbf{v}_i = \lambda_1^m \left(\alpha_1 \mathbf{v}_1 + \sum_{i=2}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^m \mathbf{v}_i \right) \approx \alpha_1 \lambda_1^m \mathbf{v}_1,$$

that is, all the terms containing $\mathbf{v}_2, \ldots, \mathbf{v}_n$ eventually become negligible, and only the one with $\mathbf{v}_1$ remains. This gives us that for m large enough,

$$\mathbf{x}_m \approx \left(\Pi_{k=0}^{m-1} \frac{1}{\|\mathbf{A}\mathbf{x}_k\|_2} \right) \alpha_1 \lambda_1^m \mathbf{v}_1 = C\mathbf{v}_1,$$

where $C = \left(\Pi_{k=0}^{m-1} \frac{1}{\|\mathbf{A}\mathbf{x}_k\|_2} \right) \alpha_1 \lambda_1^m$ is just a scalar. Since eigenvectors are only unique in their direction, this means that $\mathbf{x}_m$ converges to the eigenvector $\mathbf{v}_1$ (and is normalized to have length 1 in the Euclidean norm).

With $A\mathbf{x}_m \approx \lambda_1 \mathbf{x}_m$ established, and $\|\mathbf{x}_m\|_2 = 1$, then the Rayleigh quotient satisfies

$$\mu_m = \frac{\mathbf{x}_m^H (\mathbf{A}\mathbf{x}_m)}{\|\mathbf{x}_m\|_2^2} = \mathbf{x}_m^H (\mathbf{A}\mathbf{x}_m) \approx \lambda_1 \|\mathbf{x}_m\|_2^2 = \lambda_1.$$

Hence, once the power method converges to a direction (eigenvector $\mathbf{v}_1$), the Rayleigh quotient recovers the dominant eigenvalue.

The rate of convergence depends on how quickly $\sum_{i=1}^n \alpha_i \lambda_i^m \mathbf{v}_i \approx \alpha_1 \lambda_1^m \mathbf{v}_1$. Since the eigenvalues are ordered by magnitude, and the α_i's and $\mathbf{v}_i$'s are fixed, convergence happens when $|\lambda_1|^m \gg |\lambda_2|^m$, which is equivalent to $\left(\frac{|\lambda_2|}{|\lambda_1|} \right)^m$ being very small. Hence, it is the ratio of the second largest to the largest eigenvalues that determines the convergence rate: if $\frac{|\lambda_2|}{|\lambda_1|}$ is small (close to zero) then convergence is rapid, but if it is close to one then convergence can be slow.

Example 48. *Run the power method on the matrix*

$$\mathbf{A} = \begin{pmatrix} 1.0181e+00 & 4.4535e-02 & 3.1901e-02 \\ -1.6856e-03 & 1.0017e+00 & -6.5115e-04 \\ -6.5794e-03 & -3.6852e-02 & 9.8021e-01 \end{pmatrix}.$$

How many iterations does it need to converge to a tolerance of 10^{-6}*? We enter the matrix into MATLAB and run the power method, with initial guess* $[1;1;1]$*:*

```
>> A = [1.0181e+00   4.4535e-02   3.1901e-02
       -1.6856e-03   1.0017e+00  -6.5115e-04
       -6.5794e-03  -3.6852e-02   9.8021e-01];

>> [l,x,itcount]=PowerMethod(A,[1;1;1],1e-6)

l =
   1.0100e+00

x =
   9.7905e-01
```

```
   -2.0103e-01
    3.2415e-02


itcount =
   996
```

This took many more iterations than our previous example to converge (996 *versus* 6). *This is because the ratio for this matrix is* $\frac{|\lambda_2|}{|\lambda_1|} = \frac{1}{1.01} = 0.991$ (*close to* 1), *but for the previous example it was* $\frac{|\lambda_2|}{|\lambda_1|} = \frac{0.4641}{6.4641} = 0.0072$ (*close to* 0).

Inverse Power Method. Recall from your linear algebra course that a nonsingular matrix $\mathbf{A}$ has the same eigenvectors as its inverse, and its eigenvalues are reciprocals of its inverse. To see this, take $\mathbf{Ax} = \lambda\mathbf{x}$, left-multiply both sides by $\mathbf{A}^{-1}$, and then by $\frac{1}{\lambda}$, which reveals

$$\lambda^{-1}\mathbf{x} = \mathbf{A}^{-1}\mathbf{x}.$$

Note $\lambda \neq 0$ since $\mathbf{A}$ is assumed nonsingular. Thus, the smallest eigenvalue of $\mathbf{A}$ is the reciprocal of the largest eigenvalue of $\mathbf{A}^{-1}$. We can therefore run the power method on $\mathbf{A}^{-1}$ to determine the smallest eigenvalue of $\mathbf{A}$. Of course, one never should explicitly take an inverse, so in the implementation the method's multiplication via $\mathbf{x}_{k+1} = \mathbf{A}^{-1}\mathbf{x}_k$ can be changed to a linear solve: find $\mathbf{x}_{k+1}$ satisfying $\mathbf{A}\mathbf{x}_{k+1} = \mathbf{x}_k$. Code for the inverse power method is as follows:

```
function [eval,evec,itcount] = InvPowerMethod(A, x0, tol)

% normalize initial condition
x = x0/norm(x0);

% keep track of Rayleigh quotient as approx of lambda
rq = x' * (A*x);

% will do many solves with A, so prefactor
[L,U,P]=lu(A);

itcount=0;
while norm( A*x - rq*x)/norm(A*x) >tol
    itcount=itcount+1;

    % solve Ay=x -> PAy=Px -> LUy = Px
    y = U \ (L \ P*x);
    % normalize
    x = y / norm(y);
```

```
    rq = x' * (A*x);
end
eval=rq;
evec=x;
```

We run the inverse power method on the same matrix as the previous example (code shown below), and correctly converge to the smallest eigenvalue $\lambda_3 = 0.99$ and its corresponding eigenvector:

```
>> [l,x,itcount]=InvPowerMethod(A,[1;1;1],1e-6)

l =
   9.9000e-01

x =
  -7.0438e-01
  -6.2037e-02
   7.0710e-01

itcount =
   764
```

Shift-invert Power Method. We have shown above how to find the largest and smallest eigenvalues of a matrix. We can also use a similar technique to find the eigenvalue of $\mathbf{A}$ that is closest to a particular scalar $\sigma \in \mathbb{C}$. The key idea is that the eigenvalues of $(\mathbf{A} - \sigma\mathbf{I})$ are given by $\lambda_i - \sigma$, where λ_i is an eigenvalue of $\mathbf{A}$. To see this, start with $\mathbf{Ax} = \lambda\mathbf{x}$, and subtract $\sigma\mathbf{x}$ from both sides to reveal

$$\mathbf{Ax} - \sigma\mathbf{x} = \lambda\mathbf{x} - \sigma\mathbf{x} = (\lambda - \sigma)\mathbf{x},$$

and consequently, $(\mathbf{A} - \sigma\mathbf{I})\mathbf{x} = (\lambda - \sigma)\mathbf{x}$.

Thus for any eigenpair $(\lambda, \mathbf{x})$ of $\mathbf{A}$, $(\lambda - \sigma, \mathbf{x})$ is an eigenpair of $(\mathbf{A} - \sigma\mathbf{I})$. Therefore if $(\mathbf{A} - \sigma\mathbf{I})$ is nonsingular, then we can run the inverse power method on it to find the smallest eigenvalue, which has the form $\lambda_j - \sigma$. Adding σ to it recovers λ_j, which must be the closest eigenvalue to σ. The code to do this is identical to the inverse power method, except that the matrix $(\mathbf{A} - \sigma\mathbf{I})$ is used:

```
function [eval,evec,itcount] = ShiftInvPowerMethod(A, sigma, x0, tol)

% normalize initial condition
x = x0/norm(x0);

% keep track of Rayleigh quotient as approx of lambda
rq = x' * (A*x);
```

```
% will do many solves with A, so prefactor
n = size(A,1);
[L,U,P]=lu(A - sigma*eye(n));

itcount=0;
while norm( A*x - rq*x)/norm(A*x) >tol
    itcount=itcount+1;

    y = U \ (L \ P*x);
    % normalize
    x = y / norm(y);

    rq = x' * (A*x);
end
eval=rq;
evec=x;
```

Using the same matrix as the previous examples, we can find the second largest eigenvalue by choosing σ closest to 1 (the 3 eigenvalues of the matrix are 1.01, 1, and 0.99). We choose $\sigma = 0.98$ in the commands below, and converge to the eigenvalue 0.99 in 12 iterations. Note that this is much faster than using the inverse power method to find this eigenvalue.

```
>> [l,x,itcount]=ShiftInvPowerMethod(A,1.02, [1;1;1],1e-6)

l =
   9.9000e-01

x =
  -7.0437e-01
  -6.2080e-02
   7.0712e-01

itcount =
    12
```

Rayleigh Quotient Iteration. The shift-invert power method with a fixed shift σ converges linearly to the unique eigenvalue closest to σ. Following the analysis of the power method, we can show that the factor of convergence is $\left|\frac{\lambda_1-\sigma}{\lambda_2-\sigma}\right|$. That is, if $|\sigma - \lambda_1| \ll |\sigma - \lambda_2|$, we can expect rapid convergence. This suggests that we may use a variable shift σ_k for the shift-invert power method, letting it converge to the eigenvalue of interest. In particular, if we let $\sigma_k = \mu_k = \mathbf{x}_k^H \mathbf{A} \mathbf{x}_k$ with $\|\mathbf{x}_k\|_2 = 1$, the resulting algorithm is called the Rayleigh quotient iteration.

Assume that $\mathbf{x}_k$ (with $\|\mathbf{x}_k\|_2 = 1$) is very close to a particular eigenvector $\mathbf{v}_\ell$ of A in direction, such that the Rayleigh quotient $\mu_k = \mathbf{x}_k^H \mathbf{A} \mathbf{x}_k$ is closer to the

corresponding eigenvalue λ_ℓ than it is to all other eigenvalues, then one can show that the Rayleigh quotient iteration converges asymptotically to $(\lambda_\ell, \mathbf{v}_\ell)$ cubically for real symmetric or complex Hermitian A, and in general quadratically for a nonsymmetric or non-Hermitian A.

Two issues need attention for the Rayleigh quotient iteration. First, note that the Rayleigh quotient iteration converges quadratically or cubically only if $\mathbf{x}_k$ is already sufficiently close to the desired eigenvector $\mathbf{v}_\ell$ in direction. This is similar to the Newton's method for solving nonlinear equations. To make sure such a condition is satisfied, we may first start with the shift-invert power method with σ very close to λ_ℓ, let the method proceed several iterations such that $\mathbf{x}_k \approx \mathbf{v}_\ell$, then switch to the Rayleigh quotient iteration. Meanwhile, though we can achieve quadratic/cubic asymptotic convergence, we have to solve a linear system with a new coefficient matrix $\mathbf{A} - \mu_k \mathbf{I}$ in each iteration. The linear solves would be much more expensive than those in the shift-invert power method, which can use the LU factors of $\mathbf{A} - \sigma\mathbf{I}$ computed only once, since σ is fixed.

6.3 Multiple-vector iterations

The single-vector iteration methods in the original form can find only one eigenpair. In many situations, however, we need more eigenpairs with similar spectral characteristics, i.e., several dominant eigenpairs or a dozen eigenpairs around a specified shift $\sigma \in \mathbb{C}$. Iterations using multiple-vectors can be used in this setting, and are a natural extensions of the single-vector methods.

The subspace iteration (also called simultaneous iteration) is a widely used extension of this type. We provide the outline of a basic version of this method. One can see easily that it is a straightforward extension of the power method, and the only essential difference is that the single-vector iterate $\mathbf{x}_k$ (with $\|\mathbf{x}_k\|_2 = 1$) of the power method is replaced with a block of p vectors $\mathbf{X}_k$ with orthonormal columns. In addition, suppose that we want individual dominant eigenpairs (instead of an orthonormal basis of this invariant subspace), a post-processing step is needed to retrieve these eigenpairs of $\mathbf{A}$ from $\mathbf{X}_k$ and $\mathbf{M}_k = \mathbf{X}_k^H \mathbf{A} \mathbf{X}_k$.

The analysis of the subspace iteration is almost parallel to that of the power method. Assume that the eigenvalues of $\mathbf{A}$ are ordered such that $|\lambda_1| \geq \ldots \geq |\lambda_p| > |\lambda_{p+1}| \geq \ldots \geq |\lambda_n|$. If the algorithm proceeds without orthonormalizing $\mathbf{X}_{k+1}$ in each iteration, then each column of $\mathbf{X}_{k+1}$ will converge to the dominant eigenvector $\mathbf{v}_1$ in direction if $|\lambda_1| > |\lambda_2|$. However, the orthonormalization enforces the columns of $\mathbf{X}_{k+1}$ to have unit 2-norm and be orthogonal to each other.

Subspace Iteration for computing p dominant eigenpairs of $\mathbf{A}$

Choose $tol > 0$, $\mathbf{X}_0 \in \mathbb{C}^{n\times p}$ **with** $\mathbf{X}_0^H\mathbf{X}_0 = I_p$.

For $k = 0, 1, \ldots,$ **until convergence**

$\mathbf{M}_k = \mathbf{X}_k^H\mathbf{A}\mathbf{X}_k$; $\mathbf{R}_k = \mathbf{A}\mathbf{X}_k - \mathbf{X}_k\mathbf{M}_k$;

If $\frac{\|\mathbf{R}_k\|_F}{\|\mathbf{A}\mathbf{X}_k\|_F} \leq tol$, **exit;**

$\mathbf{X}_{k+1} = \mathbf{A}\mathbf{X}_k$;

Orthonormalize $\mathbf{X}_{k+1}$ **such that** $\mathbf{X}_{k+1}^H\mathbf{X}_{k+1} = I_p$; **(typically done by a reduced QR factorization of** $\mathbf{X}_{k+1}$**)**

End For

Diagonalize $\mathbf{M}_k = [\mathbf{w}_1^{(k)}, \ldots, \mathbf{w}_p^{(k)}]\mathrm{diag}(\mu_1^{(k)}, \ldots, \mu_p^{(k)})[\mathbf{w}_1^{(k)}, \ldots, \mathbf{w}_p^{(k)}]^{-1}$;

output the eigenpair approximations $(\mu_i^{(k)}, \mathbf{X}_k\mathbf{w}_i^{(k)})$ **(**$1 \leq i \leq p$**) of** $\mathbf{A}$.

As a result, $\mathbf{X}_{k+1}$ will converge to an orthonormal basis of the dominant invariant subspace $\mathrm{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_p\}$ associated with $\lambda_1, \ldots, \lambda_p$. The post-processing step at the end will extract each individual eigenpair approximation. The subspace iteration converges linearly to these eigenpairs at the asymptotic rate $\left|\frac{\lambda_{p+1}}{\lambda_p}\right|$.

If we want to compute p eigenvalues closest to a specified shift σ, we can develop the shift-invert variant of the subspace iteration. The only work here is to replace $\mathbf{X}_{k+1} = \mathbf{A}\mathbf{X}_k$ with $\mathbf{X}_{k+1} = (\mathbf{A} - \sigma\mathbf{I})^{-1}\mathbf{X}_k$, achieved by solving the linear system of equations $(\mathbf{A} - \sigma\mathbf{I})\mathbf{X}_{k+1} = \mathbf{X}_k$ with p right-hand sides $\mathbf{X}_k = [\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_p^{(k)}]$. An LU factorization should be performed only once, before the for loop for the solution of such a linear system in each iteration. If there exists a unique set of p eigenvalues of $\mathbf{A}$ closest to σ, the shift-invert subspace iteration will usually converge to the desired eigenpairs.

6.4 Finding all eigenvalues and eigenvectors of a matrix

All the algorithms introduced in previous sections can be used to compute one or a few eigenvalues of a small dense or a large and typically sparse matrix. Notably absent from this chapter is a discussion of finding *all* eigenvalues and eigenvectors of a matrix. In MATLAB, this can be done with the `eig` command, and most students use this command in their introductory linear algebra class.

We have omitted discussion of this topic for two reasons. First, there are additional technical details to efficiently extend the ideas above for subspace iteration to the whole space to make converge rapidly. Second, it is not a practical algorithm for very large matrices: it is slow ($O(n^2)$ flops for real symmetric matrices and $O(n^3)$ flops for general nonsymmetric matrices), and is storage-consuming since the eigenvectors will form a dense matrix the same size as the input matrix. The algorithm is practical for matrices of size up to about twenty thousand on a personal computer. As this is an overview course, we choose to leave the material to a first graduate course in numerical linear algebra.

6.5 Exercises

1. Assume that $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n\times n}$, and at least one of them is nonsingular. Show that $\Lambda(\mathbf{AB}) = \Lambda(\mathbf{BA})$, i.e., the eigenvalues of $\mathbf{AB}$ and $\mathbf{BA}$ are identical.

2. Let $\mathbf{A} = \begin{pmatrix} 1 & 10^6 \\ 0 & 2 \end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix} 1 & 10^6 \\ 2\times 10^{-6} & 2 \end{pmatrix}$. Evaluate $\|\mathbf{B}-\mathbf{A}\|_\infty$ and see how large the difference between $\mathbf{A}$ and $\mathbf{B}$ is. How much do the eigenvalues change? If $\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix} 1 & 2\times 10^{-6} \\ 2\times 10^{-6} & 2 \end{pmatrix}$, explore the same questions. (This example shows how sensitive certain eigenvalues of a nonsymmetric matrix could be under perturbations, and how insensitive they are in the symmetric case, respectively)

3. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be an orthogonal matrix such that $\mathbf{A}^T\mathbf{A} = \mathbf{A}\mathbf{A}^T = I_n$. Show that for any vector $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{Ax}\|_2 = \|\mathbf{x}\|_2$. From this fact, what can we say about the modulus of the eigenvalues of $\mathbf{A}$? Would the power method or subspace iteration be able to find one or a few dominant eigenpairs of $\mathbf{A}$?

4. Let $\mathbf{A} = \begin{pmatrix} 5 & 7 & 3 \\ 0 & 1 & 2 \\ 4 & -1 & 6 \end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix} 2 & 4 & -1 \\ 7 & 0 & 1 \\ 3 & 6 & 5 \end{pmatrix}$. Use MATLAB's `eig` to find eigenvalues of both matrices. Is it possible to use the power method to find the dominant eigenvalue of $\mathbf{A}$ and $\mathbf{B}$, respectively? What about using the inverse power method to find the smallest (in modulus) eigenvalue?

5. (a) Let $\mathbf{x}_0 = [1\ 1\ 1]^T$. Use the power method for $\mathbf{A}$ in Problem 4 to compute $\mathbf{x}_1$ without any normalization. Evaluate the Rayleigh quotient $\mu_k = \frac{\mathbf{x}_k^T\mathbf{A}\mathbf{x}_k}{\mathbf{x}_k^T\mathbf{x}_k}$

($k = 0, 1$), and compare with the dominant eigenvalue of $\mathbf{A}$.
(b) Let $\mathbf{x}_0 = [-13 \;\; 15 \;\; -13]^T$. Invoke the inverse power method for $\mathbf{B}$ in Problem 4 to compute $\mathbf{x}_1$, without any normalization. Evaluate the Rayleigh quotient $\mu_k = \frac{\mathbf{x}_k^T \mathbf{A} \mathbf{x}_k}{\mathbf{x}_k^T \mathbf{x}_k}$ ($k = 0, 1$); compare with the smallest eigenvalue of $\mathbf{B}$.

6. (a) Follow the analysis for the power method in section 6.2 to derive an analysis of the shift-invert power method. Show that this method would typically converge to the eigenvalue closest to the specified shift σ under certain assumptions, and the asymptotic rate of convergence is $\left|\frac{\lambda_1 - \sigma}{\lambda_2 - \sigma}\right|$, where λ_1 and λ_2 denote the eigenvalues of $\mathbf{A}$ closest and second closest to σ.
(b) Use the conclusion in part (a), explain why the shift-invert method with shift $\sigma = 0.98$ converges to $\lambda = 0.99$ of the 3×3 matrix in section 6.2 much more quickly than the inverse power method (*hint: the invert power method is nothing but the shift-invert power method with shift* $\sigma = 0$).

7. (a) Implement the subspace iteration in MATLAB. To test your code, run MATLAB command `load west0479;` and compute the 8 eigenvalues of largest modulus to relative tolerance 10^{-8}. Compare your computed eigenvalues with those obtained from MATLAB command `eigs(west0479,8,'lm')`. You may use the command `[X,~] = qr(X,0)` to orthonormalize X_{k+1}.
(b) Implement the shift-invert subspace iteration. Make sure to use `lu` to factorize $\mathbf{A} - \sigma\mathbf{I}$ before the for loop and use the LU factors to solve the linear systems $(\mathbf{A} - \sigma\mathbf{I})\mathbf{X}_{k+1} = \mathbf{X}_k$. To test your code, compute the 6 eigenvalues around $\sigma = 1$ of `west0479` to relative tolerance 10^{-8}. Compare your results with those obtained from command `eigs(west0479,6,1)`.