Applied Mathematics 205

Unit V: Eigenvalue Problems

Lecturer: Dr. David Knezevic

Unit V: Eigenvalue Problems

Chapter V.3: Algorithms for Eigenvalue Problems

Power Method

Power Method

The power method is perhaps the simplest eigenvalue algorithm

It finds the eigenvalue of $A \in \mathbb{C}^{n \times n}$ with largest modulus

```
1: choose $x_0 \in \mathbb{C}^n$ arbitrarily
2: for $k = 1, 2, \ldots$ do
3:     $x_k = Ax_{k-1}$
4: end for
```

Question: How does this algorithm work?

Power Method

Assuming A is nondefective, then the eigenvectors $v_1, v_2, \ldots, v_n$ provide a basis for $\mathbb{C}^n$

Therefore there exists coefficients α_i such that $x_0 = \sum_{j=1}^{n} \alpha_j v_j$

Then, we have

$$\begin{aligned} x_k &= A x_{k-1} = A^2 x_{k-2} = \cdots = A^k x_0 \\ &= A^k \left(\sum_{j=1}^{n} \alpha_j v_j \right) = \sum_{j=1}^{n} \alpha_j A^k v_j \\ &= \sum_{j=1}^{n} \alpha_j \lambda_j^k v_j \\ &= \lambda_n^k \left(\alpha_n v_n + \sum_{j=1}^{n-1} \alpha_j \left[\frac{\lambda_j}{\lambda_n} \right]^k v_j \right) \end{aligned}$$

Power Method

Then if $|\lambda_n| > |\lambda_j|$, $1 \leq j < n$, we see that $x_k \to \lambda_n^k \alpha_n v_n$ as $k \to \infty$

This algorithm converges linearly: the “error terms” are scaled by a factor at most $|\lambda_{n-1}|/|\lambda_n|$ at each iteration

Also, we see that the method converges faster if λ_n is “well separated” from the rest of the spectrum

Power Method

However, in practice the exponential factor λ_n^k could cause overflow or underflow after relatively few iterations

Therefore the standard form of the power method is actually the normalized power method

```
1: choose $x_0 \in \mathbb{C}^n$ arbitrarily
2: for $k = 1, 2, \ldots$ do
3:     $y_k = Ax_{k-1}$
4:     $x_k = y_k / \|y_k\|$
5: end for
```

Power Method

Convergence analysis of the normalized power method is essentially the same as the un-normalized case

Only difference is we now get an extra scaling factor, $c_k \in \mathbb{R}$, due to the normalization at each step

$$x_k \quad = \quad c_k \lambda_n^k \left(\alpha_n v_n + \sum_{j=1}^{n-1} \alpha_j \left[\frac{\lambda_j}{\lambda_n} \right]^k v_j \right)$$

Power Method

This algorithm directly produces the eigenvector v_n

One way to recover λ_n is to note that

$$y_k = Ax_{k-1} \approx \lambda_n x_{k-1}$$

Hence we can compare an entry of y_k and x_{k-1} to approximate λ_n

We also note two potential issues:

1. We require x_0 to have a nonzero component of v_n
2. There may be more than one eigenvalue with maximum modulus

Power Method

Issue 1:

- In practice, very unlikely that x_0 will be orthogonal to v_n
- Even if $x_0^* v_n = 0$, rounding error will introduce a component of v_n during the power iterations

Issue 2:

- We cannot ignore the possibility that there is more than one "max. eigenvalue"
- In this case x_k would converge to a member of the corresponding eigenspace

Power Method

An important idea in eigenvalue computations is to consider the "shifted" matrix $A - \sigma \mathrm{I}$, for $\sigma \in \mathbb{R}$

We see that

$$(A - \sigma \mathrm{I}) v_i = (\lambda_i - \sigma) v_i$$

and hence the spectrum of $A - \sigma \mathrm{I}$ is shifted by $-\sigma$, and the eigenvectors are the same

For example, if all the eigenvalues are real, a shift can be used with the power method to converge to λ_1 instead of λ_n

Power Method

Matlab example: Consider power method and shifted power method for

$$A = \begin{bmatrix} 4 & 1 \\ 1 & -2 \end{bmatrix},$$

which has eigenvalues $\lambda_1 = -2.1623$, $\lambda_2 = 4.1623$

Inverse Iteration

Inverse Iteration

The eigenvalues of A^{-1} are the reciprocals of the eigenvalues of A, since

$$Av = \lambda v \iff A^{-1}v = \frac{1}{\lambda}v$$

Question: What happens if we apply the power method to A^{-1}?

Inverse Iteration

Answer: We converge to the largest (in modulus) eigenvalue of A^{-1}, which is $1/\lambda_1$ (recall that λ_1 is the smallest eigenvalue of A)

This is called inverse iteration

```
1: choose $x_0 \in \mathbb{C}^n$ arbitrarily
2: for $k = 1, 2, \ldots$ do
3:     solve $Ay_k = x_{k-1}$ for $y_k$
4:     $x_k = y_k / \|y_k\|$
5: end for
```

Inverse Iteration

Hence inverse iteration gives λ_1 without requiring a shift

This is helpful since it may be difficult to determine what shift is required to get λ_1 in the power method

Question: What happens if we apply inverse iteration to the shifted matrix $A - \sigma \mathrm{I}$?

Inverse Iteration

The smallest eigenvalue of $A - \sigma \mathrm{I}$ is $(\lambda_{i^*} - \sigma)$, where

$$i^* = \arg \min_{i=1,2,\ldots,n} |\lambda_i - \sigma|,$$

and hence...

Answer: We converge to $\tilde{\lambda} = 1/(\lambda_{i^*} - \sigma)$, then recover λ_{i^*} via

$$\lambda_{i^*} = \frac{1}{\tilde{\lambda}} + \sigma$$

Inverse iteration with shift allows us to find the eigenvalue closest to σ

Inverse Iteration

Matlab example: Eigenvalues of the Laplacian via inverse iteration

Rayleigh Quotient Iteration

Rayleigh Quotient

For the remainder of this chapter (Rayleigh Quotient Iteration, QR Algorithm) we will assume that $A \in \mathbb{R}^{n\times n}$ is real and symmetric[1]

The Rayleigh quotient is defined as

$$r(x) \equiv \frac{x^T A x}{x^T x}$$

If $(\lambda, v) \in \mathbb{R} \times \mathbb{R}^n$ is an eigenpair, then

$$r(v) = \frac{v^T A v}{v^T v} = \frac{\lambda v^T v}{v^T v} = \lambda$$

[1]Much of the material generalizes to complex non-hermitian matrices, but symmetric case is simpler

Rayleigh Quotient

Theorem: Suppose $A \in \mathbb{R}^{n \times n}$ is a symmetric matrix, then for any $x \in \mathbb{R}^n$ we have

$$\lambda_1 \leq r(x) \leq \lambda_n$$

Proof: We write x as a linear combination of (orthogonal) eigenvectors $x = \sum_{j=1}^n \alpha_j v_j$, and the lower bound follows from

$$r(x) = \frac{x^T A x}{x^T x} = \frac{\sum_{j=1}^n \lambda_j \alpha_j^2}{\sum_{j=1}^n \alpha_j^2} \geq \lambda_1 \frac{\sum_{j=1}^n \alpha_j^2}{\sum_{j=1}^n \alpha_j^2} = \lambda_1$$

The proof of the upper bound $r(x) \leq \lambda_n$ is analogous $\square$

Rayleigh Quotient

Corollary: A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive definite if and only if all of its eigenvalues are positive

Proof: ($\Rightarrow$) Suppose A is symmetric positive definite (SPD), then for any nonzero $x \in \mathbb{R}^n$, we have $x^T A x > 0$ and hence

$$\lambda_1 = r(v_1) = \frac{v_1^T A v_1}{v_1^T v_1} > 0$$

($\Leftarrow$) Suppose A has positive eigenvalues, then for any nonzero $x \in \mathbb{R}^n$

$$x^T A x = r(x)(x^T x) \geq \lambda_1 \|x\|_2^2 > 0$$

$\square$

Rayleigh Quotient

But also, if x is an approximate eigenvector, then $r(x)$ gives us a good approximation to the eigenvalue

This is because estimation of an eigenvalue from an approximate eigenvector is an $n \times 1$ linear least squares problem: $x\lambda \approx Ax$

$x \in \mathbb{R}^n$ is our "tall thin matrix" and $Ax \in \mathbb{R}^n$ is our right-hand side

Hence the normal equation for $x\lambda \approx Ax$ yields the Rayleigh quotient, i.e.

$$x^T x\lambda = x^T Ax$$

Rayleigh Quotient

Question: How accurate is the Rayleigh quotient approximation to an eigenvalue?

Let's consider r as a function of x, so $r : \mathbb{R}^n \to \mathbb{R}$

$$\begin{aligned}\frac{\partial r(x)}{\partial x_j} &= \frac{\frac{\partial}{\partial x_j}(x^T A x)}{x^T x} - \frac{(x^T A x)\frac{\partial}{\partial x_j}(x^T x)}{(x^T x)^2} \\ &= \frac{2(Ax)_j}{x^T x} - \frac{(x^T A x)2x_j}{(x^T x)^2} \\ &= \frac{2}{x^T x}(Ax - r(x)x)_j\end{aligned}$$

(Note that the second equation relies on the symmetry of A)

Rayleigh Quotient

Therefore

$$\nabla r(x) = \frac{2}{x^T x}(Ax - r(x)x)$$

For an eigenpair (λ, v) we have $r(v) = \lambda$ and hence

$$\nabla r(v) = \frac{2}{v^T v}(Av - \lambda v) = 0$$

This shows that eigenvectors of A are stationary points of r

Rayleigh Quotient

Suppose (λ, v) is an eigenpair of A, and let us consider a Taylor expansion of $r(x)$ about v:

$$\begin{aligned} r(x) &= r(v) + \nabla r(v)^T(x-v) \\ &\quad + \frac{1}{2}(x-v)^T H_r(v)(x-v) + \text{H.O.T.} \\ &= r(v) + \frac{1}{2}(x-v)^T H_r(v)(x-v) + \text{H.O.T.} \end{aligned}$$

Hence as $x \to v$ the error in a Rayleigh quotient approximation is

$$|r(x) - \lambda| = O(\|x - v\|_2^2)$$

That is, the Rayleigh quotient approx. to an eigenvalue squares the error in a corresponding eigenvector approx.

Rayleigh Quotient Iteration

The Rayleigh quotient gives us an eigenvalue estimate from an eigenvector estimate

Inverse iteration gives us an eigenvector estimate from an eigenvalue estimate

It is natural to combine the two, this yields the Rayleigh quotient iteration

```
1: choose $x_0 \in \mathbb{R}^n$ arbitrarily
2: for $k = 1, 2, \dots$ do
3:     $\sigma_k = x_{k-1}^T A x_{k-1} / x_{k-1}^T x_{k-1}$
4:     solve $(A - \sigma_k \mathrm{I}) y_k = x_{k-1}$ for $y_k$
5:     $x_k = y_k / \|y_k\|$
6: end for
```

Rayleigh Quotient Iteration

Suppose, at step k, we have $\|x_{k-1} - v\| \leq \epsilon$

Then, from the Rayleigh quotient in line 3 of the algorithm, we have $|\sigma_k - \lambda| = O(\epsilon^2)$

In lines 4 and 5 of the algorithm, we then perform an inverse iteration with shift σ_k to get x_k

Recall the eigenvector error in one inverse iteration step is scaled by ratio of "second largest to largest eigenvalues" of $(A - \sigma_k \mathrm{I})^{-1}$

Rayleigh Quotient Iteration

Let λ be the closest eigenvalue of A to σ_k, then the magnitude of largest eigenvalue of $(A - \sigma_k \mathrm{I})^{-1}$ is $1/|\sigma_k - \lambda|$

The second largest eigenvalue magnitude is $1/|\sigma_k - \hat{\lambda}|$, where $\hat{\lambda}$ is the eigenvalue of A "second closest" to σ_k

Hence at each inverse iteration step, the error is reduced by a factor

$$\frac{|\sigma_k - \lambda|}{|\sigma_k - \hat{\lambda}|} = \frac{|\sigma_k - \lambda|}{|(\sigma_k - \lambda) + (\lambda - \hat{\lambda})|} \longrightarrow \mathrm{const.}|\sigma_k - \lambda| \text{ as } \sigma_k \to \lambda$$

Therefore, we obtain cubic convergence as $k \to \infty$:

$$\|x_k - v\| \to (\mathrm{const.}|\sigma_k - \lambda|)\|x_{k-1} - v\| = O(\epsilon^3)$$

Rayleigh Quotient Iteration

A drawback of Rayleigh iteration: we can't just LU factorize $A - \sigma_k \mathrm{I}$ once since the shift changes each step

Also, it's harder to pick out specific parts of the spectrum with Rayleigh quotient iteration since σ_k can change unpredictably

Matlab demo: Rayleigh iteration to compute an eigenpair of

$$A = \begin{bmatrix} 5 & 1 & 1 \\ 1 & 6 & 1 \\ 1 & 1 & 7 \end{bmatrix}$$

Matlab demo: Rayleigh iteration to compute an eigenpair of Laplacian

QR Algorithm

The QR Algorithm

The QR algorithm for computing eigenvalues is one of the best known algorithms in Numerical Analysis[2]

It was developed independently in the late 1950s by John G.F. Francis (England) and Vera N. Kublanovskaya (USSR)

The QR algorithm efficiently provides approximations for all eigenvalues/eigenvectors of a matrix

We will consider what happens when we apply the power method to a set of vectors — this will then motivate the QR algorithm

[2]Recall that here we focus on the case in which $A \in \mathbb{R}^{n \times n}$ is symmetric

The QR Algorithm

Let $x_1^{(0)}, \dots, x_p^{(0)}$ denote p linearly independent starting vectors, and suppose we store these vectors in the columns of X_0

We can apply the power method to these vectors to obtain the following algorithm:

```
1: choose an $n \times p$ matrix $X_0$ arbitrarily
2: for $k = 1, 2, \dots$ do
3:     $X_k = AX_{k-1}$
4: end for
```

The QR Algorithm

From our analysis of the power method, we see that for each $i = 1, 2, \ldots, p$:

$$\begin{aligned} x_i^{(k)} &= \left(\lambda_n^k \alpha_{i,n} v_n + \lambda_{n-1}^k \alpha_{i,n-1} v_{n-1} + \cdots + \lambda_1^k \alpha_{i,1} v_1\right) \\ &= \lambda_{n-p}^k \left(\sum_{j=n-p+1}^{n} \left(\frac{\lambda_j}{\lambda_{n-p}}\right)^k \alpha_{i,j} v_j + \sum_{j=1}^{n-p} \left(\frac{\lambda_j}{\lambda_{n-p}}\right)^k \alpha_{i,j} v_j \right) \end{aligned}$$

Then, if $|\lambda_{n-p+1}| > |\lambda_{n-p}|$, the sum in green will decay compared to the sum in blue as $k \to \infty$

Hence the columns of X_k will converge to a basis for $\text{span}\{v_{n-p+1}, \ldots, v_n\}$

The QR Algorithm

However, this method doesn't provide a good basis: each column of X_k will be very close to v_n

Therefore the columns of X_k become very close to being linearly dependent

We can resolve this issue by enforcing linear independence at each step

The QR Algorithm

We orthonormalize the vectors after each iteration via a (reduced) QR factorization, to obtain the simultaneous iteration:

```
1: choose $n \times p$ matrix $Q_0$ with orthonormal columns
2: for $k = 1, 2, \ldots$ do
3:     $X_k = A\hat{Q}_{k-1}$
4:     $\hat{Q}_k \hat{R}_k = X_k$
5: end for
```

The column spaces of $\hat{Q}_k$ and X_k in line 4 are the same

Hence columns of $\hat{Q}_k$ converge to orthonormal basis for span$\{v_{n-p+1}, \ldots, v_n\}$

The QR Algorithm

In fact, we don't just get a basis for $\mathrm{span}\{v_{n-p+1}, \ldots, v_n\}$, we get the eigenvectors themselves!

Theorem: The columns of $\hat{Q}_k$ converge to the p dominant eigenvectors of A

We will not discuss the full proof, but we note that this result is not surprising since:

- the eigenvectors of a symmetric matrix are orthogonal
- columns of $\hat{Q}_k$ converge to an orthogonal basis for $\mathrm{span}\{v_{n-p+1}, \ldots, v_n\}$

Simultaneous iteration approximates eigenvectors, we obtain eigenvalues from the Rayleigh quotient $\hat{Q}^T A \hat{Q} \approx \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$

The QR Algorithm

With $p = n$, the simultaneous iteration will approximate all eigenpairs of A

We now show a more convenient reorganization of the simultaneous iteration algorithm

We shall require some extra notation: the Q and R matrices arising in the simultaneous iteration will be underlined $\underline{Q}_k$, $\underline{R}_k$

(As we will see shortly, this is to distinguish between the matrices arising in the two different formulations...)

The QR Algorithm

Define[3] the k^{th} Rayleigh quotient matrix: $A_k \equiv \underline{Q}_k^T A \underline{Q}_k$, and the QR factors Q_k, R_k as: $Q_k R_k = A_{k-1}$

Our goal is to show that $A_k = R_k Q_k$, $k = 1, 2, \ldots$

Initialize $\underline{Q}_0 = \mathrm{I} \in \mathbb{R}^{n \times n}$, then in the first simultaneous iteration we obtain $X_1 = A$ and $\underline{Q}_1 \underline{R}_1 = A$

It follows that $A_1 = \underline{Q}_1^T A \underline{Q}_1 = \underline{Q}_1^T (\underline{Q}_1 \underline{R}_1) \underline{Q}_1 = \underline{R}_1 \underline{Q}_1$

Also $Q_1 R_1 = A_0 = \underline{Q}_0^T A \underline{Q}_0 = A$, so that $Q_1 = \underline{Q}_1$, $R_1 = \underline{R}_1$, and $A_1 = R_1 Q_1$

[3]We now we use the full, rather than the reduced, QR factorization hence we omit ˆ notation

The QR Algorithm

In the second simultaneous iteration, we have $X_2 = A\underline{Q}_1$, and we compute the QR factorization $\underline{Q}_2\underline{R}_2 = X_2$

Also, using our QR factorization of A_1 gives

$$X_2 = A\underline{Q}_1 = (\underline{Q}_1\underline{Q}_1^T)A\underline{Q}_1 = \underline{Q}_1 A_1 = \underline{Q}_1(Q_2 R_2),$$

which implies that $\underline{Q}_2 = \underline{Q}_1 Q_2 = Q_1 Q_2$ and $\underline{R}_2 = R_2$

Hence

$$A_2 = \underline{Q}_2^T A\underline{Q}_2 = Q_2^T\underline{Q}_1^T A\underline{Q}_1 Q_2 = Q_2^T A_1 Q_2 = Q_2^T Q_2 R_2 Q_2 = R_2 Q_2$$

The QR Algorithm

The same pattern continues for $k = 3, 4, \ldots$: we QR factorize A_k to get Q_k and R_k, then we compute $A_{k+1} = R_k Q_k$

The columns of the matrix $\underline{Q}_k = Q_1 Q_2 \cdots Q_k$ approximates the eigenvectors of A

The diagonal entries of the Rayleigh quotient matrix $A_k = \underline{Q}_k^T A \underline{Q}_k$ approximate the eigenvalues of A

(Also, due to eigenvector orthogonality for symmetric A, A_k converges to a diagonal matrix as $k \to \infty$)

The QR Algorithm

This discussion motivates the famous QR algorithm:

```
1: $A_0 = A$
2: **for** $k = 1, 2, \ldots$ **do**
3:     $Q_k R_k = A_{k-1}$
4:     $A_k = R_k Q_k$
5: **end for**
```

The QR Algorithm

Matlab demo: Compute eigenvalues and eigenvectors of[4]

$$A = \begin{bmatrix} 2.9766 & 0.3945 & 0.4198 & 1.1159 \\ 0.3945 & 2.7328 & -0.3097 & 0.1129 \\ 0.4198 & -0.3097 & 2.5675 & 0.6079 \\ 1.1159 & 0.1129 & 0.6079 & 1.7231 \end{bmatrix},$$

(This matrix has eigenvalues 1, 2, 3 and 4)

[4]Heath example 4.15

The QR Algorithm

We have presented the simplest version of the QR algorithm: the "unshifted" QR algorithm

In order to obtain an "industrial strength" algorithm, there are a number of other issues that need to be considered:

- convergence can be accelerated significantly by introducing shifts, as we did in inverse iteration and Rayleigh iteration
- it is more efficient to reduce A to tridiagonal form (via Householder reflectors) before applying QR algorithm
- reliable convergence criteria for the eigenvalues/eigenvectors are required

High-quality implementations, e.g. `LAPACK` or Matlab's `eig`, handle all of these subtleties for us