Final project proposal due by 6pm on Thu Nov 16. Email Chris or the TFs to set up a meeting. Those who have completed this will see four proposal points on Canvas.

Today: eigenvalue algorithms, QR algorithm
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 $r : \mathbb{R}^n \rightarrow \mathbb{R}$

\[
\frac{\partial r(x)}{\partial x_j} = \frac{\partial}{\partial x_j}(x^TAx) \cdot \frac{x^Tx}{x^Tx} - \frac{(x^TAx) \frac{\partial}{\partial x_j}(x^Tx)}{(x^Tx)^2}
\]

\[
= \frac{2(Ax)_j}{x^Tx} - \frac{(x^TAx)2x_j}{(x^Tx)^2}
\]

\[
= \frac{2}{x^Tx}(Ax - r(x)x)_j
\]

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

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

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

\[
|r(x) - \lambda| = O(\|x - v\|^2)
\]

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**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>choose $x_0 \in \mathbb{R}^n$ arbitrarily</td>
</tr>
<tr>
<td>2</td>
<td><strong>for</strong> $k = 1, 2, \ldots$ <strong>do</strong></td>
</tr>
<tr>
<td>3</td>
<td>$\sigma_k = x_{k-1}^T A x_{k-1} / x_{k-1}^T x_{k-1}$</td>
</tr>
<tr>
<td>4</td>
<td>solve $(A - \sigma_k I)y_k = x_{k-1}$ for $y_k$</td>
</tr>
<tr>
<td>5</td>
<td>$x_k = y_k / |y_k|$</td>
</tr>
<tr>
<td>6</td>
<td><strong>end for</strong></td>
</tr>
</tbody>
</table>
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 $\sigma_k$ to get $x_k$

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

Let $\lambda$ be the closest eigenvalue of $A$ to $\sigma_k$, then the magnitude of largest eigenvalue of $(A - \sigma_k I)^{-1}$ 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 $\sigma_k$.

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

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

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

$$\|x_k - v\| \rightarrow (\text{const.} |\sigma_k - \lambda|)\|x_{k-1} - v\| = O(\epsilon^3)$$
A drawback of Rayleigh iteration: we can’t just LU factorize $A - \sigma_k I$ once since the shift changes each step.

Also, it’s harder to pick out specific parts of the spectrum with Rayleigh quotient iteration since $\sigma_k$ can change unpredictably.

**Python demo**: Rayleigh iteration to compute an eigenpair of

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

The QR algorithm for computing eigenvalues is one of the best known algorithms in Numerical Analysis\(^1\)

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

\(^1\)Recall that here we focus on the case in which \(A \in \mathbb{R}^{n \times n}\) is symmetric
The QR Algorithm

Let \( x_1^{(0)}, \ldots, x_p^{(0)} \) 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: \textbf{for} \( k = 1, 2, \ldots \) \textbf{do}
3: \hspace{1em} \( X_k = AX_{k-1} \)
4: \textbf{end for}
The QR Algorithm

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

\[
\chi_i^{(k)} = \left( \lambda_n^k \alpha_i, n v_n + \lambda_{n-1}^k \alpha_i, n-1 v_{n-1} + \cdots + \lambda_1^k \alpha_i, 1 v_1 \right)
\]

\[
= \lambda_{n-p}^k \left( \sum_{j=n-p+1}^{n} \left( \frac{\lambda}{\lambda_{n-p}} \right)^k \alpha_{i,j} v_j \right) + \sum_{j=1}^{n-p} \left( \frac{\lambda_j}{\lambda_{n-p}} \right)^k \alpha_{i,j} v_j
\]

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 \( \text{span}\{v_{n-p+1}, \ldots, v_n\} \)
The QR Algorithm

In fact, we don’t just get a basis for \( \text{span}\{v_{n-p+1}, \ldots, v_n\} \), 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 \( \text{span}\{v_{n-p+1}, \ldots, v_n\} \)

Simultaneous iteration approximates eigenvectors, we obtain eigenvalues from the Rayleigh quotient \( \hat{Q}^T A \hat{Q} \approx \text{diag}(\lambda_1, \ldots, \lambda_n) \)
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 the $k^{th}$ Rayleigh quotient matrix: $A_k \equiv Q_k^T A Q_k$, 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 $Q_0 = I \in \mathbb{R}^{n \times n}$, then in the first simultaneous iteration we obtain $X_1 = A$ and $Q_1 R_1 = A$

It follows that $A_1 = Q_1^T A Q_1 = Q_1^T (Q_1 R_1) Q_1 = R_1 Q_1$

Also $Q_1 R_1 = A_0 = Q_0^T A Q_0 = A$, so that $Q_1 = Q_1$, $R_1 = R_1$, and $A_1 = R_1 Q_1$

---

$^2$We now we use the full, rather than the reduced, QR factorization hence we omit $^\wedge$ notation
The QR Algorithm

In the second simultaneous iteration, we have $X_2 = AQ_1$, and we compute the QR factorization $Q_2 R_2 = X_2$

Also, using our QR factorization of $A_1$ gives

$$X_2 = AQ_1 = (Q_1 Q_1^T) A Q_1 = Q_1 A_1 = Q_1 (Q_2 R_2),$$

which implies that $Q_2 = Q_1 Q_2 = Q_1 Q_2$ and $R_2 = R_2$

Hence

$$A_2 = Q_2^T A Q_2 = Q_2^T Q_1^T A Q_1 Q_2 = Q_2^T A_1 Q_2 = Q_2^T Q_2 R_2 Q_2 = R_2 Q_2$$
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 \( k \to \infty \))
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

Python demo: Compute eigenvalues and eigenvectors of

\[
A = \begin{pmatrix}
2.9766 & 0.3945 & 0.4198 & 1.1159 \\
0.3945 & 2.7328 & -0.3097 & 0.1129 \\
0.4198 & -0.3097 & 2.5675 & 0.6079 \\
1.1159 & 0.1129 & 0.6079 & 1.7231
\end{pmatrix}
\]

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

\(^3\)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 Python/MATLAB eig, handle all of these subtleties for us.