

Math 6610: Analysis of Numerical Methods, I

Power iteration and the QR algorithm

Department of Mathematics, University of Utah

Fall 2025

Resources: Trefethen and Bau 1997, Lectures 24, 25, 28
Atkinson 1989, Sections 9.1-9.3, 9.5
Süli and Mayers 2003, Sections 5.1, 5.2, 5.7
Salgado and Wise 2022, Sections 8.3, 8.4, 8.6

We now have enough machinery to tackle computing eigenvalues.

Our first set of observations will surround the operation $\mathbf{A} \mapsto \lambda(\mathbf{A})$.

The conceptually straightforward approach to computing eigenvalues is to implement what we do on paper: compute roots of the characteristic polynomial. I.e., given $\mathbf{A} \in \mathbb{C}^{n \times n}$, compute,

$$\lambda(\mathbf{A}) = p_{\mathbf{A}}^{-1}(0), \quad p_{\mathbf{A}}(z) = \det(\mathbf{A} - z\mathbf{I})$$

We now have enough machinery to tackle computing eigenvalues.

Our first set of observations will surround the operation $\mathbf{A} \mapsto \lambda(\mathbf{A})$.

The conceptually straightforward approach to computing eigenvalues is to implement what we do on paper: compute roots of the characteristic polynomial. I.e., given $\mathbf{A} \in \mathbb{C}^{n \times n}$, compute,

$$\lambda(\mathbf{A}) = p_{\mathbf{A}}^{-1}(0), \quad p_{\mathbf{A}}(z) = \det(\mathbf{A} - z\mathbf{I})$$

It turns out numerically implementing this is a bad idea, for at least two reasons.

One main motivation for computing roots is that we typically compute roots by hand using *elementary* operations (arithmetic + rational power).

However, we cannot use elementary operations in general.

Theorem (Abel-Ruffini, or “Abel’s impossibility”)

General polynomials of degree 5 or more have roots that are not expressible through elementary operations on the polynomial coefficients.

If $A \in \mathbb{C}^{n \times n}$, then $\deg p_A = n$. And the coefficients of p_A are explicit (rational) functions of the matrix entries.

However, we cannot use elementary operations in general.

Theorem (Abel-Ruffini, or “Abel’s impossibility”)

General polynomials of degree 5 or more have roots that are not expressible through elementary operations on the polynomial coefficients.

If $A \in \mathbb{C}^{n \times n}$, then $\deg p_A = n$. And the coefficients of p_A are explicit (rational) functions of the matrix entries.

Hence, for matrices of size 5×5 or larger, we simply cannot compute eigenvalues in a finite number of numerical elementary operations.

(This is yet another data point that computing eigenvalues is *significantly* harder than solving linear systems, or orthogonalizing vectors, or computing determinants, etc.)

However, we cannot use elementary operations in general.

Theorem (Abel-Ruffini, or “Abel’s impossibility”)

General polynomials of degree 5 or more have roots that are not expressible through elementary operations on the polynomial coefficients.

If $A \in \mathbb{C}^{n \times n}$, then $\deg p_A = n$. And the coefficients of p_A are explicit (rational) functions of the matrix entries.

Hence, for matrices of size 5×5 or larger, we simply cannot compute eigenvalues in a finite number of numerical elementary operations.

(This is yet another data point that computing eigenvalues is *significantly* harder than solving linear systems, or orthogonalizing vectors, or computing determinants, etc.)

The upshot: Any eigenvalue algorithm must be an iterative scheme that *approximates* eigenvalues.

With this realization, numerically computing roots of characteristic polynomials is fine, but there is no real motivation to stick with this procedure if there’s a better alternative.

However, there is a strong reason to not compute roots of polynomials: it's generally an ill-conditioned operation.

Consider

$$p_{\mathbf{A}}(z) = az^2 + bz + c = z^2 - 2z + 1.$$

With $\mathbf{f}(a, b, c) = p_{\mathbf{A}}^{-1}(0)$, then $\kappa_{\mathbf{f}}(1, -2, 1)$ is infinity, i.e., this explicit operation is *terribly* ill-conditioned.

However, there is a strong reason to not compute roots of polynomials: it's generally an ill-conditioned operation.

Consider

$$p_{\mathbf{A}}(z) = az^2 + bz + c = z^2 - 2z + 1.$$

With $\mathbf{f}(a, b, c) = p_{\mathbf{A}}^{-1}(0)$, then $\kappa_{\mathbf{f}}(1, -2, 1)$ is infinity, i.e., this explicit operation is *terribly* ill-conditioned.

A similar example is $p_{\mathbf{A}}(z) = z^2$, which corresponds to the 2×2 zero matrix. The roots of the perturbed polynomial $z^2 - \epsilon$ are $z = \pm\sqrt{\epsilon}$, and this root perturbation $\sqrt{\epsilon}$ is much greater than the coefficient perturbation ϵ .

All of this is a bit surprising, since $\mathbf{A} = \mathbf{I}$ yields $(a, b, c) = (1, -2, 1)$, yet we know that the (absolute) condition number of $\mathbf{I} \mapsto \lambda(\mathbf{I})$ is unity. (Bauer-Fike)

The operation $\mathbf{A} \mapsto (a, b, c)$ introduces some ill-conditioning artifacts.

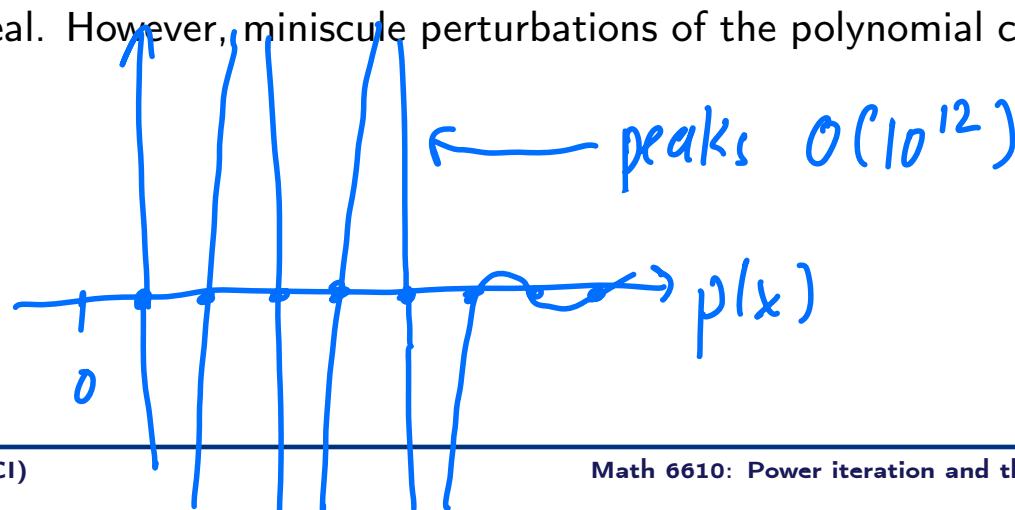
All of this is a bit surprising, since $\mathbf{A} = \mathbf{I}$ yields $(a, b, c) = (1, -2, 1)$, yet we know that the (absolute) condition number of $\mathbf{I} \mapsto \lambda(\mathbf{I})$ is unity. (Bauer-Fike)

The operation $\mathbf{A} \mapsto (a, b, c)$ introduces some ill-conditioning artifacts.

One can achieve similar results more generally: Consider *Wilkinson's polynomial*,

$$p(z) = \prod_{j=1}^{20} (z - j).$$

The roots are explicit, *simple*, and real. However, miniscule perturbations of the polynomial coefficients still lead to large changes in the roots.



All of the this is just to say: Let's not compute roots of characteristic polynomials.

A simple, naïve alternative is power iteration. For $\mathbf{A} \in \mathbb{C}^{n \times n}$, let's assume simple eigenvalues with a dominant eigenvalue:

$$\{\lambda_1, \dots, \lambda_n\} = \lambda(\mathbf{A}), \quad |\lambda_1| > \max_{j=2, \dots, n} |\lambda_j|.$$

Given a vector $\mathbf{x}$, we'll compute an eigenvector by analyzing $\mathbf{A}^k \mathbf{x}$ for $k \gg 1$.

All of the this is just to say: Let's not compute roots of characteristic polynomials.

A simple, naïve alternative is power iteration. For $A \in \mathbb{C}^{n \times n}$, let's assume simple eigenvalues with a dominant eigenvalue:

$$\{\lambda_1, \dots, \lambda_n\} = \lambda(A), \quad |\lambda_1| > \max_{j=2, \dots, n} |\lambda_j|.$$

Given a vector x , we'll compute an eigenvector by analyzing $A^k x$ for $k \gg 1$.

The motivation of this approach is the following: let A be diagonalizable, and let x have an expansion in a basis comprised of eigenvectors of A :

$$x = \sum_{j \in [n]} c_j v_j, \quad c = V^{-1}x, \quad V = (v_1 \ v_2 \ \cdots \ v_n),$$

$$A^k x = \sum_{j \in [n]} c_j A^k v_j = \sum_{j \in [n]} c_j \lambda_j^k v_j = \lambda_1^k \left[c_1 v_1 + \sum_{j=2}^n c_j v_j \left(\frac{\lambda_j}{\lambda_1} \right)^k \right]$$

All of the this is just to say: Let's not compute roots of characteristic polynomials.

A simple, naïve alternative is power iteration. For $\mathbf{A} \in \mathbb{C}^{n \times n}$, let's assume simple eigenvalues with a dominant eigenvalue:

$$\{\lambda_1, \dots, \lambda_n\} = \lambda(\mathbf{A}), \quad |\lambda_1| > \max_{j=2, \dots, n} |\lambda_j|.$$

Given a vector $\mathbf{x}$, we'll compute an eigenvector by analyzing $\mathbf{A}^k \mathbf{x}$ for $k \gg 1$.

The motivation of this approach is the following: let $\mathbf{A}$ be diagonalizable, and let $\mathbf{x}$ have an expansion in a basis comprised of eigenvectors of $\mathbf{A}$:

$$\mathbf{x} = \sum_{j \in [n]} c_j \mathbf{v}_j, \quad \mathbf{c} = \mathbf{V}^{-1} \mathbf{x}, \quad \mathbf{V} = (\mathbf{v}_1 \ \mathbf{v}_2 \ \cdots \ \mathbf{v}_n),$$

If we assume that $c_1 \neq 0$, (i.e., $\mathbf{x} \notin \text{span}\{\mathbf{v}_2, \dots, \mathbf{v}_n\}$) then

$$\mathbf{A}^k \mathbf{x} = \sum_{j \in [n]} c_j \lambda_j^k \mathbf{v}_j \implies \frac{1}{\lambda_1^k} \mathbf{A}^k \mathbf{x} = c_1 \mathbf{v}_1 + \sum_{j=2}^n c_j r_j^k \mathbf{v}_j, \quad |r_j| = \left| \frac{\lambda_j}{\lambda_1} \right| < 1,$$

and therefore if $k \gg 1$, then $\mathbf{A}^k \mathbf{x} \approx \lambda_1^k \mathbf{v}_1$.

As long as $|\lambda_1| > |\lambda_j|$ for $j \geq 2$, then $\mathbf{A}^k \mathbf{x} \approx \mathbf{v}_1$.

- $\mathbf{A}^k \mathbf{x}$ for large k can be a vector with huge norm. Iteratively normalizing would fix this.
- If ~~$\mathbf{v}$~~ is approximately an eigenvector of $\mathbf{A}$, then $R_{\mathbf{A}}(\mathbf{x}) = \frac{\mathbf{x}^* \mathbf{A} \mathbf{x}}{\|\mathbf{x}\|_2}$ is approximately its corresponding eigenvalue.

As long as $|\lambda_1| > |\lambda_j|$ for $j \geq 2$, then $\mathbf{A}^k \mathbf{x} \approx v_1$.

- $\mathbf{A}^k \mathbf{x}$ for large k can be a vector with huge norm. Iteratively normalizing would fix this.
- If ~~$\mathbf{v}$~~ is approximately an eigenvector of $\mathbf{A}$, then $R_{\mathbf{A}}(\mathbf{x}) = \frac{\mathbf{x}^* \mathbf{A} \mathbf{x}}{\|\mathbf{x}\|_2}$ is approximately its corresponding eigenvalue.

These leads to the following algorithm, Power iteration:

0. Randomly initialize $\mathbf{x}_0$, set $j = 0$.
1. Compute $\mathbf{x}_{j+1} = \frac{\mathbf{A} \mathbf{x}_j}{\|\mathbf{A} \mathbf{x}_j\|_2}$.
2. Compute $\mu_{j+1} = R_{\mathbf{A}}(\mathbf{x}_{j+1})$.
3. Set $j \leftarrow j + 1$, return to step 1.

As long as $|\lambda_1| > |\lambda_j|$ for $j \geq 2$, then $\mathbf{A}^k \mathbf{x} \approx \mathbf{v}_1$.

- $\mathbf{A}^k \mathbf{x}$ for large k can be a vector with huge norm. Iteratively normalizing would fix this.
- If $\mathbf{v}$ is approximately an eigenvector of $\mathbf{A}$, then $R_{\mathbf{A}}(\mathbf{x}) = \frac{\mathbf{x}^* \mathbf{A} \mathbf{x}}{\|\mathbf{x}\|_2}$ is approximately its corresponding eigenvalue.

These leads to the following algorithm, Power iteration:

0. Randomly initialize $\mathbf{x}_0$, set $j = 0$.
1. Compute $\mathbf{x}_{j+1} = \frac{\mathbf{A} \mathbf{x}_j}{\|\mathbf{A} \mathbf{x}_j\|_2}$.
2. Compute $\mu_{j+1} = R_{\mathbf{A}}(\mathbf{x}_{j+1})$.
3. Set $j \leftarrow j + 1$, return to step 1.

Observations:

$$r \approx 1, r < 1 \Rightarrow r^k \sim 1 - (1-r)k$$

- For large k , we expect $(\mu_k, \mathbf{x}_k) \approx (\lambda_1, \mathbf{v}_1)$.
- The error scales like r^k , where $r = \max_{j \geq 2} r_j$.
(In the language of iterative methods, this is *linear* convergence, i.e., the exponent is linear in k .)

This algorithm is fine, but is generally slow if r is close to 1.

Power iteration allows us to compute, in principle, a *single* eigenpair.

Computing the rest using this simple approach would essentially require that we “remove” $(\lambda_1, \mathbf{v}_1)$ from $\mathbf{A}$. Such a “removal” procedure is called *deflation*.

Power iteration allows us to compute, in principle, a *single* eigenpair.

Computing the rest using this simple approach would essentially require that we “remove” $(\lambda_1, \mathbf{v}_1)$ from $\mathbf{A}$. Such a “removal” procedure is called *deflation*.

Here’s a simple strategy for deflation of normal matrices. When $\mathbf{A}$ is normal, then,

$$\mathbf{A} = \sum_{j \in [n]} \lambda_j \mathbf{v}_j \mathbf{v}_j^*,$$

so “removing” $(\lambda_1, \mathbf{v}_1)$ can be accomplished as:

$$\mathbf{A}_2 := \mathbf{A} - \lambda_1 \mathbf{v}_1 \mathbf{v}_1^* = \sum_{j=2}^n \lambda_j \mathbf{v}_j \mathbf{v}_j^*,$$

Power iteration allows us to compute, in principle, a *single* eigenpair.

Computing the rest using this simple approach would essentially require that we “remove” $(\lambda_1, \mathbf{v}_1)$ from $\mathbf{A}$. Such a “removal” procedure is called *deflation*.

Here’s a simple strategy for deflation of normal matrices. When $\mathbf{A}$ is normal, then,

$$\mathbf{A} = \sum_{j \in [n]} \lambda_j \mathbf{v}_j \mathbf{v}_j^*,$$

so “removing” $(\lambda_1, \mathbf{v}_1)$ can be accomplished as:

$$\mathbf{A}_2 := \mathbf{A} - \lambda_1 \mathbf{v}_1 \mathbf{v}_1^* = \sum_{j=2}^n \lambda_j \mathbf{v}_j \mathbf{v}_j^*,$$

Now if $|\lambda_2| > |\lambda_j|$ for $j \geq 3$, we can perform power iteration on $\mathbf{A}_2$ to compute an approximation to $(\lambda_2, \mathbf{v}_2)$.

This in principle gives us a concrete (implementable) strategy for computing the full eigendecomposition of a normal matrix: perform power iteration, deflate, perform power iteration, deflate, etc.

(This deflation strategy is called *Hotelling deflation*, and is generally not numerically stable.)

The whole restarting of power iteration after deflation seems a little wasteful.

We can do a little better using *simultaneous* power iteration. We'll illustrate with 2 vectors, $(\mathbf{x}_1, \mathbf{x}_2)$:

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{V}(\mathbf{V}^{-1}\mathbf{x}_1) = \sum_{j \in [n]} c_{j,1} \mathbf{v}_j \\ \mathbf{x}_2 &= \mathbf{V}(\mathbf{V}^{-1}\mathbf{x}_2) = \sum_{j \in [n]} c_{j,2} \mathbf{v}_j\end{aligned}$$

The whole restarting of power iteration after deflation seems a little wasteful.

We can do a little better using *simultaneous* power iteration. We'll illustrate with 2 vectors, $(\mathbf{x}_1, \mathbf{x}_2)$:

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{V}(\mathbf{V}^{-1}\mathbf{x}_1) = \sum_{j \in [n]} c_{j,1} \mathbf{v}_j \\ \mathbf{x}_2 &= \mathbf{V}(\mathbf{V}^{-1}\mathbf{x}_2) = \sum_{j \in [n]} c_{j,2} \mathbf{v}_j\end{aligned}$$

Now consider $\mathbf{y}_j = \mathbf{A}^k \mathbf{x}_j$ for $k \gg 1$:

$$\mathbf{y}_1 = c_{1,1} \lambda_1^k \mathbf{v}_1 + c_{2,1} \lambda_2^k \mathbf{v}_2 + \cdots,$$

$$= \sum_{j \in [n]} c_{j,1} \lambda_j^k \mathbf{v}_j$$

$$\mathbf{y}_2 = c_{1,2} \lambda_1^k \mathbf{v}_1 + c_{2,2} \lambda_2^k \mathbf{v}_2 + \cdots$$

$$= \sum_{j \in [n]} c_{j,2} \lambda_j^k \mathbf{v}_j$$

$$\underline{\text{If}} \quad |\lambda_1| > |\lambda_2| > |\lambda_j|, \quad j \geq 3 \Rightarrow \mathbf{y}_i \approx c_{i,1} \lambda_1^k \mathbf{v}_1 + c_{i,2} \lambda_2^k \mathbf{v}_2$$

The whole restarting of power iteration after deflation seems a little wasteful.

We can do a little better using *simultaneous* power iteration. We'll illustrate with 2 vectors, (x_1, x_2) :

$$x_1 = V(V^{-1}x_1) = \sum_{j \in [n]} c_{j,1} v_j$$

$$x_2 = V(V^{-1}x_2) = \sum_{j \in [n]} c_{j,2} v_j$$

Now consider $y_j = A^k x_j$ for $k \gg 1$:

$$y_1 = c_{1,1} \lambda_1^k v_1 + c_{2,1} \lambda_2^k v_2 + \dots,$$

$$y_2 = c_{1,2} \lambda_1^k v_1 + c_{2,2} \lambda_2^k v_2 + \dots$$

If we assume $|\lambda_1| > |\lambda_2| > |\lambda_j|$ for $j \geq 3$, then

$$(y_1 \ y_2) \stackrel{\text{QR}}{\approx} (q_1 \ q_2) R$$

where $q_1 \approx v_1$, and q_2 approximately parallel to $(I - P_1)v_2$, with P_1 the orthogonal projector onto v_1 .

$$(I - P_1)y_1 = 0$$

$$y_2 \leftarrow y_2 - v_1 (y_2^* v_1)$$

$$y_2^* v_1 = c_{1,2} \lambda_1^k + c_{2,2} \lambda_2^k v_2^* v_1$$

Hence, a QR decomposition of simultaneous power iteration yields a Q matrix that is approximately the Q matrix in a QR decomposition of the eigenvector matrix. I.e., by performing thin QR decompositions:

$$A^k(x_1 \ x_2) = Q^{(k)} R^{(k)},$$

$$(v_1 \ v_2) = QR,$$

then $Q^{(k)} \approx Q$ for large k , where each matrix has 2 columns.

If $k \gg \gg 1$: $(v_1 \ v_2) \overset{\text{thin QR}}{=} \tilde{Q} \tilde{R}$
 then $Q^{(k)} \approx \tilde{Q}$

$$Q = \begin{bmatrix} \frac{v_1}{\|v_1\|_2} & \frac{(I - P_1)v_2}{\|(I - P_1)v_2\|_2} \end{bmatrix}$$

Hence, a QR decomposition of simultaneous power iteration yields a Q matrix that is approximately the Q matrix in a QR decomposition of the eigenvector matrix. I.e., by performing thin QR decompositions:

$$A^k \begin{pmatrix} x_1 & x_2 \end{pmatrix} = Q^{(k)} R^{(k)}, \quad \begin{pmatrix} v_1 & v_2 \end{pmatrix} = QR,$$

then $Q^{(k)} \approx Q$ for large k , where each matrix has 2 columns.

Generalizing this a bit: assume $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$. Then for a generic full-rank $n \times n$ matrix X :

$$A^k X = Q^{(k)} R^{(k)} \quad V = QR,$$

yields $Q^{(k)} \approx Q$.

$\nearrow$
eigenvector matrix

Hence, a QR decomposition of simultaneous power iteration yields a Q matrix that is approximately the Q matrix in a QR decomposition of the eigenvector matrix. I.e., by performing thin QR decompositions:

$$A^k \begin{pmatrix} x_1 & x_2 \end{pmatrix} = Q^{(k)} R^{(k)}, \quad \begin{pmatrix} v_1 & v_2 \end{pmatrix} = QR,$$

then $Q^{(k)} \approx Q$ for large k , where each matrix has 2 columns.

Generalizing this a bit: assume $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$. Then for a generic full-rank $n \times n$ matrix X :

$$A^k X = Q^{(k)} R^{(k)} \quad V = QR,$$

yields $Q^{(k)} \approx Q$. For, e.g., normal matrices, Q is unitary, so that $Q^{(k)}$ is actually a matrix of eigenvectors. (!)

So, for normal matrices, this is yet another algorithm: compute the QR decomposition of $A^k X$, which approximates eigenvectors, then use the Rayleigh quotient to approximate eigenvalues.

$$R_A(v_j) = \lambda_j$$

Whether we do “standard” or simultaneous power iteration:

- For normal matrices, both are directly implementable algorithms to compute the full spectrum
- These methods work in principle for eigenvalues with well-separated magnitudes.
- They generally fail when there are eigenvalues with the same magnitude.

Whether we do “standard” or simultaneous power iteration:

- For normal matrices, both are directly implementable algorithms to compute the full spectrum
- These methods work in principle for eigenvalues with well-separated magnitudes.
- They generally fail when there are eigenvalues with the same magnitude.

One could choose $X = I$ for simultaneous power iteration, so that the matrix under consideration is just A^k .

For both iterations, we are essentially computing A^k (or its action on some vector). This matrix can be terribly ill-conditioned if $k \gg 1$.

Therefore, while these procedures conceptually work, they probably aren't well-conditioned.

We have seen that the Q factor in a QR decomposition of A^k (i.e., $A^k I$) is an object of interest.

A compilation of some observations:

- If $A^k = Q^{(k)} R^{(k)}$, then $Q^{(k)} \approx Q$, where $V = QR$.

We have seen that the Q factor in a QR decomposition of A^k (i.e., $A^k I$) is an object of interest.

A compilation of some observations:

- If $A^k = Q^{(k)} R^{(k)}$, then $Q^{(k)} \approx Q$, where $V = QR$.
- We can compute $Q^{(k)}$ as the product of other unitary matrices: define $A_1 = A$.
 - ▶ Compute $A_j = Q_j R_j$
 - ▶ Define $A_{j+1} = R_j Q_j$
 - ▶ Repeat

Then: $Q^{(k)} = Q_1 Q_2 \dots Q_k \approx Q$.

We have seen that the Q factor in a QR decomposition of A^k (i.e., $A^k I$) is an object of interest.

A compilation of some observations:

- If $A^k = Q^{(k)} R^{(k)}$, then $Q^{(k)} \approx Q$, where $V = QR$.
- We can compute $Q^{(k)}$ as the product of other unitary matrices: define $A_1 = A$.
 - ▶ Compute $A_j = Q_j R_j$
 - ▶ Define $A_{j+1} = R_j Q_j$
 - ▶ Repeat

Then: $Q^{(k)} = Q_1 Q_2 \dots Q_k \approx Q$.

- The matrix Q identifies a similarity transform that triangularizes A .

We have seen that the Q factor in a QR decomposition of A^k (i.e., $A^k I$) is an object of interest.

A compilation of some observations:

- If $A^k = Q^{(k)} R^{(k)}$, then $Q^{(k)} \approx Q$, where $V = QR$.
- We can compute $Q^{(k)}$ as the product of other unitary matrices: define $A_1 = A$.
 - ▶ Compute $A_j = Q_j R_j$
 - ▶ Define $A_{j+1} = R_j Q_j$
 - ▶ Repeat

Then: $Q^{(k)} = Q_1 Q_2 \dots Q_k \approx Q$.

- The matrix Q identifies a similarity transform that triangularizes A .
- $A_{j+1} = R_j Q_j = Q_j^* Q_j R_j Q_j = Q_j^* A_j Q_j \Rightarrow A_1 = Q_1 R_1$
 i.e., $A_{j+1} = (Q_j^* \dots Q_1^*) A (Q_1 \dots Q_j)$.
 $A_2 = R_1 Q_1 = Q_1^* A_1 Q_1$

$\Downarrow$

$$A_{j+1} = (Q^{(j)})^* A Q^{(j)} \approx Q^* A Q$$

We have seen that the Q factor in a QR decomposition of A^k (i.e., $A^k I$) is an object of interest.

A compilation of some observations:

- If $A^k = Q^{(k)} R^{(k)}$, then $Q^{(k)} \approx Q$, where $V = QR$.
- We can compute $Q^{(k)}$ as the product of other unitary matrices: define $A_1 = A$.
 - ▶ Compute $A_j = Q_j R_j$
 - ▶ Define $A_{j+1} = R_j Q_j$
 - ▶ Repeat

Then: $Q^{(k)} = Q_1 Q_2 \dots Q_k \approx Q$.

- The matrix Q identifies a similarity transform that triangularizes A .
- $A_{j+1} = R_j Q_j = Q_j^* Q_j R_j Q_j = Q_j^* A_j Q_j$
 i.e., $A_{j+1} = (Q_j^* \dots Q_1^*) A (Q_1 \dots Q_j)$.

i.e.: A_k “should” be close to $Q^* A Q$ for large k .

We have motivated the proof of the following:

Theorem

Let $A \in \mathbb{C}^{n \times n}$ have eigenvalues that satisfy $|\lambda_j| > |\lambda_{j+1}|$ for $j \in [n-1]$. With $A_1 = A$, define the sequence of matrices,

$$A_j \stackrel{\text{QR}}{=} Q_j R_j, \quad A_{j+1} = R_j Q_j, \quad j \geq 1$$

Then A_j converges to the (upper triangular) Schur form of A , and hence its diagonal entries contain $\lambda(A)$.

We have motivated the proof of the following:

Theorem

Let $A \in \mathbb{C}^{n \times n}$ have eigenvalues that satisfy $|\lambda_j| > |\lambda_{j+1}|$ for $j \in [n-1]$. With $A_1 = A$, define the sequence of matrices,

$$A_j \stackrel{\text{QR}}{=} Q_j R_j, \quad A_{j+1} = R_j Q_j, \quad j \geq 1$$

Then A_j converges to the (upper triangular) Schur form of A , and hence its diagonal entries contain $\lambda(A)$.

The mechanical, iterative procedure defined above is the **QR Algorithm**. (Not the QR decomposition.)

The QR algorithm is celebrated because it is numerically stable: each iteration of the QR algorithm performs:

$$A = QR \implies A \leftarrow RQ = Q^* A Q.$$

I.e., it is just a sequence of *unitary* similarity transforms on A , so we expect numerical stability.

The QR algorithm is celebrated because it is numerically stable: each iteration of the QR algorithm performs:

$$\mathbf{A} = \mathbf{QR} \implies \mathbf{A} \leftarrow \mathbf{RQ} = \mathbf{Q}^* \mathbf{A} \mathbf{Q}.$$

I.e., it is just a sequence of *unitary* similarity transforms on $\mathbf{A}$, so we expect numerical stability.

Recall that the “orthogonal triangularization” strategy of QR decompositions computes $\mathbf{R}$ in $\mathbf{A} = \mathbf{QR}$ through a sequence of left-applications of unitary transformations (Givens rotations or Householder reflectors).

To implement the QR algorithm, one simply re-uses these transforms on the right of $\mathbf{A}$.

The QR algorithm is celebrated because it is numerically stable: each iteration of the QR algorithm performs:

$$\mathbf{A} = \mathbf{QR} \implies \mathbf{A} \leftarrow \mathbf{RQ} = \mathbf{Q}^* \mathbf{A} \mathbf{Q}.$$

I.e., it is just a sequence of *unitary* similarity transforms on $\mathbf{A}$, so we expect numerical stability.

Recall that the “orthogonal triangularization” strategy of QR decompositions computes $\mathbf{R}$ in $\mathbf{A} = \mathbf{QR}$ through a sequence of left-applications of unitary transformations (Givens rotations or Householder reflectors).

To implement the QR algorithm, one simply re-uses these transforms on the right of $\mathbf{A}$.

NB: We have *not* solved all our problems! The QR algorithm is implicitly just power iteration in disguise, and power iteration isn't really that great....

-  Atkinson, Kendall (1989). *An Introduction to Numerical Analysis*. New York: Wiley. ISBN: 978-0-471-62489-9.
-  Salgado, Abner J. and Steven M. Wise (2022). *Classical Numerical Analysis: A Comprehensive Course*. Cambridge: Cambridge University Press. ISBN: 978-1-108-83770-5. DOI: 10.1017/9781108942607.
-  Süli, Endre and David F. Mayers (2003). *An Introduction to Numerical Analysis*. Cambridge: Cambridge University Press. ISBN: 978-0-521-00794-8. DOI: 10.1017/CB09780511801181.
-  Trefethen, Lloyd N. and David Bau (1997). *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics. ISBN: 0-89871-361-7.