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

Computing eigenvalues

We now have enough machinery to tackle computing eigenvalues.

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

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

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

Computing eigenvalues

We now have enough machinery to tackle computing eigenvalues.

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

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

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

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 $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, then $\deg p_{\boldsymbol{A}} = n$. And the coefficients of $p_{\boldsymbol{A}}$ 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 $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, then $\deg p_{\boldsymbol{A}} = n$. And the coefficients of $p_{\boldsymbol{A}}$ 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 $\boldsymbol{A} \in \mathbb{C}^{n \times n}$, then $\deg p_{\boldsymbol{A}} = n$. And the coefficients of $p_{\boldsymbol{A}}$ 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

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

With $\boldsymbol{f}(a,b,c) = p_{\boldsymbol{A}}^{-1}(0)$, then $\kappa_{\boldsymbol{f}}(1,-2,1)$ 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

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

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

A similar example is $p_{\boldsymbol{A}}(z) = z^2$, 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 $\boldsymbol{A} = \boldsymbol{I}$ yields $(a, b, c) = (1, -2, 1)$, yet we know that the (absolute) condition number of $\boldsymbol{I} \mapsto \lambda(\boldsymbol{I})$ is unity. (Bauer-Fike)

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

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

The operation $\boldsymbol{A} \mapsto (a, b, c)$ 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.

Matrix powers

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

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

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

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

Matrix powers

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

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

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

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

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

$$\boldsymbol{x} = \sum_{j\in[n]} c_j \boldsymbol{v}_j, \qquad\qquad \boldsymbol{c} = \boldsymbol{V}^{-1}\boldsymbol{x}, \qquad\qquad \boldsymbol{V} = \begin{pmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \end{pmatrix},$$

Matrix powers

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

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

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

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

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

$$\boldsymbol{x} = \sum_{j\in[n]} c_j \boldsymbol{v}_j, \qquad\qquad \boldsymbol{c} = \boldsymbol{V}^{-1}\boldsymbol{x}, \qquad\qquad \boldsymbol{V} = \begin{pmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \end{pmatrix},$$

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

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

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

Power iteration

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

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

Power iteration

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

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

These leads to the following algorithm, Power iteration:

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

Power iteration

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

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

These leads to the following algorithm, Power iteration:

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

Observations:

- For large k, we expect $(\mu_k, \boldsymbol{x}_k) \approx (\lambda_1, \boldsymbol{v}_1)$.
- The error scales like r^k, where $r = \max_{j \geqslant 2} r_j$.
  (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.

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" $(\lambda_1, \boldsymbol{v}_1)$ from $\boldsymbol{A}$. Such a "removal" procedure is called *deflation*.

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" $(\lambda_1, \boldsymbol{v}_1)$ from $\boldsymbol{A}$. Such a "removal" procedure is called *deflation*.

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

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

so "removing" $(\lambda_1, \boldsymbol{v}_1)$ can be accomplished as:

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

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" $(\lambda_1, \boldsymbol{v}_1)$ from $\boldsymbol{A}$. Such a "removal" procedure is called *deflation*.

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

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

so "removing" $(\lambda_1, \boldsymbol{v}_1)$ can be accomplished as:

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

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

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.)

Simultaneous power iteration, I

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, $(\boldsymbol{x}_1, \boldsymbol{x}_2)$:

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

Simultaneous power iteration, I

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, $(\boldsymbol{x}_1, \boldsymbol{x}_2)$:

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

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

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

Simultaneous power iteration, I

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, $(\boldsymbol{x}_1, \boldsymbol{x}_2)$:

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

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

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

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

$$(\boldsymbol{y}_1 \;\; \boldsymbol{y}_2) \overset{\text{QR}}{\approx} (\boldsymbol{q}_1 \;\; \boldsymbol{q}_2)\,\boldsymbol{R}$$

where $\boldsymbol{q}_1 \approx \boldsymbol{v}_1$, and $\boldsymbol{q}_2$ approximately parallel to $(\boldsymbol{I} - \boldsymbol{P}_1)\boldsymbol{v}_2$, with $\boldsymbol{P}_1$ the orthogonal projector onto $\boldsymbol{v}_1$.

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

$$\boldsymbol{A}^k(\boldsymbol{x}_1 \;\; \boldsymbol{x}_2) = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)}, \qquad (\boldsymbol{v}_1 \;\; \boldsymbol{v}_2) = \boldsymbol{Q}\boldsymbol{R},$$

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

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

$$\boldsymbol{A}^k(\boldsymbol{x}_1 \;\; \boldsymbol{x}_2) = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)}, \qquad (\boldsymbol{v}_1 \;\; \boldsymbol{v}_2) = \boldsymbol{Q}\boldsymbol{R},$$

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

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

$$\boldsymbol{A}^k\boldsymbol{X} = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)} \qquad \boldsymbol{V} = \boldsymbol{Q}\boldsymbol{R},$$

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

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

$$\boldsymbol{A}^k(\boldsymbol{x}_1 \;\; \boldsymbol{x}_2) = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)}, \qquad (\boldsymbol{v}_1 \;\; \boldsymbol{v}_2) = \boldsymbol{Q}\boldsymbol{R},$$

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

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

$$\boldsymbol{A}^k\boldsymbol{X} = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)} \qquad \boldsymbol{V} = \boldsymbol{Q}\boldsymbol{R},$$

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

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

Power iteration recap

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.

Power iteration recap

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 $\boldsymbol{X} = \boldsymbol{I}$ for simultaneous power iteration, so that the matrix under consideration is just $\boldsymbol{A}^k$.

For both iterations, we are essentially computing $\boldsymbol{A}^k$ (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.

Some key realizations

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

A compilation of some observations:

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

Some key realizations

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

A compilation of some observations:

- If $\boldsymbol{A}^k = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)}$, then $\boldsymbol{Q}^{(k)} \approx \boldsymbol{Q}$, where $\boldsymbol{V} = \boldsymbol{Q}\boldsymbol{R}$.
- We can compute $\boldsymbol{Q}^{(k)}$ as the product of other unitary matrices: define $\boldsymbol{A}_1 = \boldsymbol{A}$.
  - Compute $\boldsymbol{A}_j = \boldsymbol{Q}_j\boldsymbol{R}_j$
  - Define $\boldsymbol{A}_{j+1} = \boldsymbol{R}_j\boldsymbol{Q}_j$
  - Repeat

  Then: $\boldsymbol{Q}^{(k)} = \boldsymbol{Q}_1\boldsymbol{Q}_2\ldots\boldsymbol{Q}_k \approx \boldsymbol{Q}$.

Some key realizations

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

A compilation of some observations:

- If $\boldsymbol{A}^k = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)}$, then $\boldsymbol{Q}^{(k)} \approx \boldsymbol{Q}$, where $\boldsymbol{V} = \boldsymbol{Q}\boldsymbol{R}$.
- We can compute $\boldsymbol{Q}^{(k)}$ as the product of other unitary matrices: define $\boldsymbol{A}_1 = \boldsymbol{A}$.
  - Compute $\boldsymbol{A}_j = \boldsymbol{Q}_j\boldsymbol{R}_j$
  - Define $\boldsymbol{A}_{j+1} = \boldsymbol{R}_j\boldsymbol{Q}_j$
  - Repeat

  Then: $\boldsymbol{Q}^{(k)} = \boldsymbol{Q}_1\boldsymbol{Q}_2\ldots\boldsymbol{Q}_k \approx \boldsymbol{Q}$.
- The matrix $\boldsymbol{Q}$ identifies a similarity transform that triangularizes $\boldsymbol{A}$.

Some key realizations

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

A compilation of some observations:

- If $\boldsymbol{A}^k = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)}$, then $\boldsymbol{Q}^{(k)} \approx \boldsymbol{Q}$, where $\boldsymbol{V} = \boldsymbol{Q}\boldsymbol{R}$.
- We can compute $\boldsymbol{Q}^{(k)}$ as the product of other unitary matrices: define $\boldsymbol{A}_1 = \boldsymbol{A}$.
  - Compute $\boldsymbol{A}_j = \boldsymbol{Q}_j\boldsymbol{R}_j$
  - Define $\boldsymbol{A}_{j+1} = \boldsymbol{R}_j\boldsymbol{Q}_j$
  - Repeat

  Then: $\boldsymbol{Q}^{(k)} = \boldsymbol{Q}_1\boldsymbol{Q}_2\ldots\boldsymbol{Q}_k \approx \boldsymbol{Q}$.
- The matrix $\boldsymbol{Q}$ identifies a similarity transform that triangularizes $\boldsymbol{A}$.
- $\boldsymbol{A}_{j+1} = \boldsymbol{R}_j\boldsymbol{Q}_j = \boldsymbol{Q}_j^*\boldsymbol{Q}_j\boldsymbol{R}_j\boldsymbol{Q}_j = \boldsymbol{Q}_j^*\boldsymbol{A}_j\boldsymbol{Q}_j$
  I.e., $\boldsymbol{A}_{j+1} = \left(\boldsymbol{Q}_j^*\cdots\boldsymbol{Q}_1^*\right)\boldsymbol{A}\left(\boldsymbol{Q}_1\cdots\boldsymbol{Q}_j\right)$.

Some key realizations

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

A compilation of some observations:

- If $\boldsymbol{A}^k = \boldsymbol{Q}^{(k)}\boldsymbol{R}^{(k)}$, then $\boldsymbol{Q}^{(k)} \approx \boldsymbol{Q}$, where $\boldsymbol{V} = \boldsymbol{Q}\boldsymbol{R}$.
- We can compute $\boldsymbol{Q}^{(k)}$ as the product of other unitary matrices: define $\boldsymbol{A}_1 = \boldsymbol{A}$.
  - Compute $\boldsymbol{A}_j = \boldsymbol{Q}_j\boldsymbol{R}_j$
  - Define $\boldsymbol{A}_{j+1} = \boldsymbol{R}_j\boldsymbol{Q}_j$
  - Repeat

  Then: $\boldsymbol{Q}^{(k)} = \boldsymbol{Q}_1\boldsymbol{Q}_2\ldots\boldsymbol{Q}_k \approx \boldsymbol{Q}$.
- The matrix $\boldsymbol{Q}$ identifies a similarity transform that triangularizes $\boldsymbol{A}$.
- $\boldsymbol{A}_{j+1} = \boldsymbol{R}_j\boldsymbol{Q}_j = \boldsymbol{Q}_j^*\boldsymbol{Q}_j\boldsymbol{R}_j\boldsymbol{Q}_j = \boldsymbol{Q}_j^*\boldsymbol{A}_j\boldsymbol{Q}_j$
  I.e., $\boldsymbol{A}_{j+1} = \left(\boldsymbol{Q}_j^*\cdots\boldsymbol{Q}_1^*\right)\boldsymbol{A}\left(\boldsymbol{Q}_1\cdots\boldsymbol{Q}_j\right)$.

I.e.: $\boldsymbol{A}_k$ **"should" be close to** $\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}$ **for large** k**.**

We have motivated the proof of the following:

Theorem

Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ have eigenvalues that satisfy $|\lambda_j| > |\lambda_{j+1}|$ for $j \in [n-1]$. With $\boldsymbol{A}_1 = \boldsymbol{A}$, define the sequence of matrices,

$$\boldsymbol{A}_j \overset{\text{QR}}{=} \boldsymbol{Q}_j \boldsymbol{R}_j, \qquad \boldsymbol{A}_{j+1} = \boldsymbol{R}_j \boldsymbol{Q}_j, \qquad j \geqslant 1$$

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

The QR algorithm

We have motivated the proof of the following:

Theorem

Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ have eigenvalues that satisfy $|\lambda_j| > |\lambda_{j+1}|$ for $j \in [n-1]$. With $\boldsymbol{A}_1 = \boldsymbol{A}$, define the sequence of matrices,

$$\boldsymbol{A}_j \overset{\text{QR}}{=} \boldsymbol{Q}_j \boldsymbol{R}_j, \qquad \boldsymbol{A}_{j+1} = \boldsymbol{R}_j \boldsymbol{Q}_j, \qquad j \geqslant 1$$

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

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:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \implies \boldsymbol{A} \leftarrow \boldsymbol{R}\boldsymbol{Q} = \boldsymbol{Q}^* \boldsymbol{A} \boldsymbol{Q}.$$

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

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

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \implies \boldsymbol{A} \leftarrow \boldsymbol{R}\boldsymbol{Q} = \boldsymbol{Q}^* \boldsymbol{A} \boldsymbol{Q}.$$

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

Recall that the "orthogonal triangularization" strategy of QR decompositions computes $\boldsymbol{R}$ in $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$ 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 $\boldsymbol{A}$.

Stability

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

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \implies \boldsymbol{A} \leftarrow \boldsymbol{R}\boldsymbol{Q} = \boldsymbol{Q}^* \boldsymbol{A} \boldsymbol{Q}.$$

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

Recall that the "orthogonal triangularization" strategy of QR decompositions computes $\boldsymbol{R}$ in $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$ 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 $\boldsymbol{A}$.

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/CBO9780511801181.


Trefethen, Lloyd N. and David Bau (1997). *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics. ISBN: 0-89871-361-7.