# Math 6610: Analysis of Numerical Methods, I
# Rayleigh iteration and the QR algorithm with shifts

Department of Mathematics, University of Utah

Fall 2025

Resources: Trefethen and Bau 1997, Lectures 27, 29
Atkinson 1989, Section 9.6
Süli and Mayers 2003, Sections 5.8, 5.9
Salgado and Wise 2022, Section 8.4

# Recall: power iteration



With $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ and $\boldsymbol{x} \in \mathbb{C}^{n}$, we expect

$$\boldsymbol{A}^k \boldsymbol{x} \propto \lambda_1^k \boldsymbol{v}_1, \qquad k \gg 1,$$

where $(\lambda_1, \boldsymbol{v}_1)$ is the dominant eigenpair of $\boldsymbol{A}$. The generic algorithm that computes $\boldsymbol{A}^k \boldsymbol{x}$ as an approximation to $\boldsymbol{v}_1$ is power iteration.

Convergence is *linear*, i.e., $\|\boldsymbol{A}^k \boldsymbol{x} - c\lambda_1^k \boldsymbol{v}_1\|_2 \sim r^k$, where $r = \sup_{j>1} |\lambda_j/\lambda_1| < 1$.

To compute several eigenpairs, we can use simultaneous (power) iteration, computing $\boldsymbol{A}^k$ for large $k$.

A *stable* numerical algorithm that essentially performs power iteration is the QR algorithm.

# Recall: power iteration



With $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ and $\boldsymbol{x} \in \mathbb{C}^n$, we expect

$$\boldsymbol{A}^k \boldsymbol{x} \propto \lambda_1^k \boldsymbol{v}_1, \qquad\qquad k \gg 1,$$

where $(\lambda_1, \boldsymbol{v}_1)$ is the dominant eigenpair of $\boldsymbol{A}$. The generic algorithm that computes $\boldsymbol{A}^k \boldsymbol{x}$ as an approximation to $\boldsymbol{v}_1$ is power iteration.

Convergence is *linear*, i.e., $\|\boldsymbol{A}^k \boldsymbol{x} - c\lambda_1^k \boldsymbol{v}_1\|_2 \sim r^k$, where $r = \sup_{j>1} |\lambda_j/\lambda_1| < 1$.

To compute several eigenpairs, we can use simultaneous (power) iteration, computing $\boldsymbol{A}^k$ for large $k$.

A *stable* numerical algorithm that essentially performs power iteration is the QR algorithm.

The problem: The constant $r$ above can be very close to 1, making convergence slow, as then $r^k \approx 1 - k(1-r)$.

# Spectral shifts

Here's a simple but powerful observation: if $\lambda_j$ is an eigenvalue of $\boldsymbol{A}$, and $\mu \in \mathbb{C}\backslash\lambda(\boldsymbol{A})$, then

$$\frac{1}{\lambda_j - \mu} \in \lambda(\boldsymbol{B}), \qquad \boldsymbol{B} = (\boldsymbol{A} - \mu \boldsymbol{I})^{-1},$$

Here's a simple but powerful observation: if $\lambda_j$ is an eigenvalue of $\boldsymbol{A}$, and $\mu \in \mathbb{C}\backslash\lambda(\boldsymbol{A})$, then

$$\frac{1}{\lambda_j - \mu} \in \lambda(\boldsymbol{B}), \qquad\qquad \boldsymbol{B} = (\boldsymbol{A} - \mu\boldsymbol{I})^{-1},$$

The main utility of this fact is that if $\mu \approx \lambda_j$, then

$$\max_{k \neq j} \frac{\frac{1}{\lambda_k - \mu}}{\frac{1}{\lambda_j - \mu}} = \max_{k \neq j} \frac{\lambda_j - \mu}{\lambda_k - \mu} \ll 1.$$

The factor on the left is constant "$r$" in the convergence rate of power iteration for $(\boldsymbol{A} - \mu\boldsymbol{I})^{-1}$.

# Spectral shifts

Here's a simple but powerful observation: if $\lambda_j$ is an eigenvalue of $\boldsymbol{A}$, and $\mu \in \mathbb{C} \backslash \lambda(\boldsymbol{A})$, then

$$\frac{1}{\lambda_j - \mu} \in \lambda(\boldsymbol{B}), \qquad \boldsymbol{B} = (\boldsymbol{A} - \mu \boldsymbol{I})^{-1},$$

The main utility of this fact is that if $\mu \approx \lambda_j$, then

$$\max_{k \neq j} \frac{\frac{1}{\lambda_k - \mu}}{\frac{1}{\lambda_j - \mu}} = \max_{k \neq j} \frac{\lambda_j - \mu}{\lambda_k - \mu} \ll 1.$$

The factor on the left is constant "$r$" in the convergence rate of power iteration for $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$.

The punch line: <u>if</u> we can choose $\mu \approx \lambda_j$, we can (considerably) accelerate power iteration's convergence to $\lambda_j$.

The parameter $\mu$ is called a *shift*.

# Inverse iteration

To see how shifting can provide utility, note that even if $|\lambda_1 - \lambda_2|$ is small, if we are able to choose $\mu$ very close to $\lambda_1$, then

$$\left|\frac{\lambda_1 - \mu}{\lambda_2 - \mu}\right| \ll 1, \qquad\qquad \left|\frac{1}{\lambda_1 - \mu}\right| \gg \left|\frac{1}{\lambda_2 - \mu}\right|,$$

and hence power iteration on $(\boldsymbol{A} - \mu \boldsymbol{I})$ has a well-separated dominant eigenvalue of $1/(\lambda_1 - \mu)$, which can be inverted and re-shifted back to $\lambda_1$ since $\mu$ is known.

To see how shifting can provide utility, note that even if $|\lambda_1 - \lambda_2|$ is small, if we are able to choose $\mu$ very close to $\lambda_1$, then

$$\left|\frac{\lambda_1 - \mu}{\lambda_2 - \mu}\right| \ll 1, \qquad \left|\frac{1}{\lambda_1 - \mu}\right| \gg \left|\frac{1}{\lambda_2 - \mu}\right|,$$

and hence power iteration on $(\boldsymbol{A} - \mu \boldsymbol{I})$ has a well-separated dominant eigenvalue of $1/(\lambda_1 - \mu)$, which can be inverted and re-shifted back to $\lambda_1$ since $\mu$ is known.

Performing power iteration on $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ is called (shifted) inverse iteration.

0. Given $\mu \in \mathbb{C}$, define $\boldsymbol{A}^{\mu-} = (\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$. Randomly initialize $\boldsymbol{x}_0$, set $j = 0$.
1. Compute $\boldsymbol{x}_{j+1} = \frac{\boldsymbol{A}^{\mu-}\boldsymbol{x}_j}{\|\boldsymbol{A}^{\mu-}\boldsymbol{x}_j\|_2}$.
2. Compute $\mu_{j+1} = R_{\boldsymbol{A}}(\boldsymbol{x}_{j+1})$.
3. Set $j \leftarrow j + 1$, return to step 1.

NB: This requires linear solves!

# The shift

Note that the choice of shift is integral to the success of inverse iteration, *and* the choice of shift should depend on the "targeted" eigenvalue.

Happily, we have a reasonable guess for a good eigenvalue: it's the Rayleigh quotient $R_{\boldsymbol{A}}(\boldsymbol{x}_j)$.

# The shift



Note that the choice of shift is integral to the success of inverse iteration, *and* the choice of shift should depend on the "targeted" eigenvalue.

Happily, we have a reasonable guess for a good eigenvalue: it's the Rayleigh quotient $R_{\boldsymbol{A}}(\boldsymbol{x}_j)$.

Therefore, one strategy is to actually use this estimate of the eigenvalue as a shift:

$$\boldsymbol{x}_0 \;\rightarrow\; \mu = R_{\boldsymbol{A}}(\boldsymbol{x}_0) \;\rightarrow\; \boldsymbol{x}_1 = \frac{\boldsymbol{A}^{\mu_0 -}\boldsymbol{x}_0}{\|\boldsymbol{A}^{\mu_0 -}\boldsymbol{x}_0\|} \;\rightarrow\; \mu_1 = R_{\boldsymbol{A}}(\boldsymbol{x}_1) \;\rightarrow \ldots.$$

# Rayleigh iteration

This leads to an algorithm called Rayleigh iteration, which uses the Rayleigh quotient to approximate shifts for use in inverse iteration:

0. With $(\mu_0, \boldsymbol{x}_0)$ a guess for an initial eigenpair, set $j = 0$.
1. Compute $\boldsymbol{x}_{j+1} = \frac{\boldsymbol{A}^{\mu_j -}\boldsymbol{x}_j}{\|\boldsymbol{A}^{\mu_j -}\boldsymbol{x}_j\|_2}$.
2. Compute $\mu_{j+1} = R_{\boldsymbol{A}}(\boldsymbol{x}_{j+1})$.
3. Set $j \leftarrow j + 1$, return to step 1.

(Alternatively, just initialize $\boldsymbol{x}_0$ and set $\mu_0 = \boldsymbol{R}_{\boldsymbol{A}}(\boldsymbol{x}_0)$.

# Rayleigh iteration

This leads to an algorithm called Rayleigh iteration, which uses the Rayleigh quotient to approximate shifts for use in inverse iteration:

0. With $(\mu_0, \boldsymbol{x}_0)$ a guess for an initial eigenpair, set $j = 0$.
1. Compute $\boldsymbol{x}_{j+1} = \frac{\boldsymbol{A}^{\mu_j -}\boldsymbol{x}_j}{\|\boldsymbol{A}^{\mu_j -}\boldsymbol{x}_j\|_2}$.
2. Compute $\mu_{j+1} = R_{\boldsymbol{A}}(\boldsymbol{x}_{j+1})$.
3. Set $j \leftarrow j + 1$, return to step 1.

(Alternatively, just initialize $\boldsymbol{x}_0$ and set $\mu_0 = \boldsymbol{R}_{\boldsymbol{A}}(\boldsymbol{x}_0)$. How well does Rayleigh iteration work? Consider first power iteration with $(\lambda, \boldsymbol{v})$ a dominant eigenpair:

$$\boldsymbol{x}_0 = \boldsymbol{v} + \boldsymbol{\delta}, \qquad \left\|\frac{\boldsymbol{A}}{\lambda}\boldsymbol{\delta}\right\|_2 \leqslant r\|\boldsymbol{\delta}\|_2, \qquad r = \max_{\lambda \in \Lambda(\boldsymbol{A})\backslash\{\lambda\}} \left|\frac{\lambda_j}{\lambda}\right| < 1,$$

This leads to an algorithm called Rayleigh iteration, which uses the Rayleigh quotient to approximate shifts for use in inverse iteration:

0. With $(\mu_0, \boldsymbol{x}_0)$ a guess for an initial eigenpair, set $j = 0$.
1. Compute $\boldsymbol{x}_{j+1} = \frac{\boldsymbol{A}^{\mu_j -} \boldsymbol{x}_j}{\|\boldsymbol{A}^{\mu_j -} \boldsymbol{x}_j\|_2}$.
2. Compute $\mu_{j+1} = R_{\boldsymbol{A}}(\boldsymbol{x}_{j+1})$.
3. Set $j \leftarrow j + 1$, return to step 1.

(Alternatively, just initialize $\boldsymbol{x}_0$ and set $\mu_0 = \boldsymbol{R}_{\boldsymbol{A}}(\boldsymbol{x}_0)$. How well does Rayleigh iteration work? Consider first power iteration with $(\lambda, \boldsymbol{v})$ a dominant eigenpair:

$$\boldsymbol{x}_0 = \boldsymbol{v} + \boldsymbol{\delta}, \qquad \left\| \frac{\boldsymbol{A}}{\lambda} \boldsymbol{\delta} \right\|_2 \leqslant r \|\boldsymbol{\delta}\|_2, \qquad r = \max_{\lambda \in \Lambda(\boldsymbol{A}) \backslash \{\lambda\}} \left| \frac{\lambda_j}{\lambda} \right| < 1,$$

where we've assumed $\boldsymbol{A}$ is diagonalizable. Then,

$$\left\| \frac{1}{\lambda^k} \boldsymbol{A}^k \boldsymbol{x}_0 - \boldsymbol{v} \right\|_2 = \left\| \frac{\boldsymbol{A}}{\lambda} \underbrace{\left( \frac{\boldsymbol{A}^{k-1} \boldsymbol{x}_0}{\lambda^{k-1}} - \boldsymbol{v} \right)}_{\propto \boldsymbol{\delta}} \right\|_2 \leqslant r \left\| \frac{1}{\lambda^{k-1}} \boldsymbol{A}^{k-1} \boldsymbol{x}_0 - \boldsymbol{v} \right\|_2.$$

I.e., the error at stage $k$ is essentially the error at stage $k-1$ times the factor $r < 1$.

# Rayleigh iteration analysis



$$\left\| \frac{1}{\lambda_1^k} \boldsymbol{A}^k \boldsymbol{x}_0 - \boldsymbol{v} \right\|_2 \leqslant r \left\| \frac{1}{\lambda^{k-1}} \boldsymbol{A}^{k-1} \boldsymbol{x}_0 - \boldsymbol{v} \right\|_2 \implies \|\boldsymbol{x}_{k+1} - \boldsymbol{v}\|_2 \leqslant r \|\boldsymbol{x}_k - \boldsymbol{v}\|$$

The next part we need to analyze is the Rayleigh quotient, which is how we compute shifts for the next iteration.

Recall that if $(\lambda, \boldsymbol{v})$ is an eigenpair of $\boldsymbol{A}$, then $R_{\boldsymbol{A}}(\boldsymbol{v}) = \lambda$.

# Rayleigh iteration analysis



$$\left\| \frac{1}{\lambda_1^k} \boldsymbol{A}^k \boldsymbol{x}_0 - \boldsymbol{v} \right\|_2 \leqslant r \left\| \frac{1}{\lambda^{k-1}} \boldsymbol{A}^{k-1} \boldsymbol{x}_0 - \boldsymbol{v} \right\|_2 \implies \|\boldsymbol{x}_{k+1} - \boldsymbol{v}\|_2 \leqslant r \|\boldsymbol{x}_k - \boldsymbol{v}\|$$

The next part we need to analyze is the Rayleigh quotient, which is how we compute shifts for the next iteration.

Recall that if $(\lambda, \boldsymbol{v})$ is an eigenpair of $\boldsymbol{A}$, then $R_{\boldsymbol{A}}(\boldsymbol{v}) = \lambda$.

A nice additional property is that $(\nabla_{\boldsymbol{x}} R_{\boldsymbol{A}}(\boldsymbol{x}))\big|_{\boldsymbol{x}=\boldsymbol{v}} = 0$.
I.e., eigenvectors $\boldsymbol{v}$ are stationary points of $R_{\boldsymbol{A}}$.

Hence, by a Taylor series argument, when $\boldsymbol{x} \approx \boldsymbol{v}$, we have,

$$R_{\boldsymbol{A}}(\boldsymbol{x}) = R_{\boldsymbol{A}}(\boldsymbol{v}) + \mathcal{O}(\|\boldsymbol{x} - \boldsymbol{v}\|_2^2) = \lambda + \mathcal{O}(\|\boldsymbol{x} - \boldsymbol{v}\|_2^2).$$

# Rayleigh iteration analysis

$$\left\| \frac{1}{\lambda_1^k} \boldsymbol{A}^k \boldsymbol{x}_0 - \boldsymbol{v} \right\|_2 \leqslant r \left\| \frac{1}{\lambda^{k-1}} \boldsymbol{A}^{k-1} \boldsymbol{x}_0 - \boldsymbol{v} \right\|_2 \implies \|\boldsymbol{x}_{k+1} - \boldsymbol{v}\|_2 \leqslant r \|\boldsymbol{x}_k - \boldsymbol{v}\|$$

The next part we need to analyze is the Rayleigh quotient, which is how we compute shifts for the next iteration.

Recall that if $(\lambda, \boldsymbol{v})$ is an eigenpair of $\boldsymbol{A}$, then $R_{\boldsymbol{A}}(\boldsymbol{v}) = \lambda$.

A nice additional property is that $(\nabla_{\boldsymbol{x}} R_{\boldsymbol{A}}(\boldsymbol{x}))\Big|_{\boldsymbol{x}=\boldsymbol{v}} = 0$.
I.e., eigenvectors $\boldsymbol{v}$ are stationary points of $R_{\boldsymbol{A}}$.

Hence, by a Taylor series argument, when $\boldsymbol{x} \approx \boldsymbol{v}$, we have,

$$R_{\boldsymbol{A}}(\boldsymbol{x}) = R_{\boldsymbol{A}}(\boldsymbol{v}) + \mathcal{O}(\|\boldsymbol{x} - \boldsymbol{v}\|_2^2) = \lambda + \mathcal{O}(\|\boldsymbol{x} - \boldsymbol{v}\|_2^2).$$

Therefore, if $(\mu_k, \boldsymbol{x}_k)$ is "close" to an eigenpair $(\lambda, \boldsymbol{v})$ of $\boldsymbol{A}$, then $1/(\lambda - \mu)$ is the dominant eigenpair of $\boldsymbol{A}_{\mu^-}$, so,

$$\begin{aligned} r &= \max_{\lambda \in \Lambda(\boldsymbol{A}) \backslash \{\lambda\}} \frac{\mu_k - \lambda}{\mu_k - \lambda} \\ &= \mathcal{O}(\mu_k - \lambda) = \mathcal{O}(R_{\boldsymbol{A}}(\boldsymbol{x}_k) - R_{\boldsymbol{A}}(\boldsymbol{v})) = \mathcal{O}(\|\boldsymbol{x}_k - \boldsymbol{v}\|_2^2). \end{aligned}$$

# Rayleigh iteration convergence

We have "approximately" proved the following:

**Theorem**

*Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$. For almost every initialization $(\mu_0, \boldsymbol{x}_0) \in \mathbb{C} \times \mathbb{C}^n$ of Rayleigh iteration, there is some eigenpair $(\lambda, \boldsymbol{v})$ of $\boldsymbol{A}$ such that,*

$$\|\boldsymbol{x}_{j+1} - \boldsymbol{v}\|_2 = \mathcal{O}(\|\boldsymbol{x}_j - \boldsymbol{v}\|_2^3), \qquad \|\mu_{j+1} - \lambda\|_2 = \mathcal{O}(\|\mu_j - \lambda\|_2^3)$$

# Rayleigh iteration convergence

We have "approximately" proved the following:

**Theorem**

*Let* $\boldsymbol{A} \in \mathbb{C}^{n \times n}$. *For almost every initialization* $(\mu_0, \boldsymbol{x}_0) \in \mathbb{C} \times \mathbb{C}^n$ *of Rayleigh iteration, there is some eigenpair* $(\lambda, \boldsymbol{v})$ *of* $\boldsymbol{A}$ *such that,*

$$\|\boldsymbol{x}_{j+1} - \boldsymbol{v}\|_2 = \mathcal{O}(\|\boldsymbol{x}_j - \boldsymbol{v}\|_2^3), \qquad \|\mu_{j+1} - \lambda\|_2 = \mathcal{O}(\|\mu_j - \lambda\|_2^3)$$

The result: convergence is <u>cubic</u>. (!!!!!)

I.e., if we reach a stage where $(\mu_j, \boldsymbol{v}_j)$ has one digit of accuracy, then $(\mu_{j+1}, \boldsymbol{v}_{j+1})$ will have 3 digits of accuracy, and $(\mu_{j+2}, \boldsymbol{v}_{j+2})$ will have *nine*. (!!!!!)

We have "approximately" proved the following:

**Theorem**

*Let* $\boldsymbol{A} \in \mathbb{C}^{n \times n}$. *For almost every initialization* $(\mu_0, \boldsymbol{x}_0) \in \mathbb{C} \times \mathbb{C}^n$ *of Rayleigh iteration, there is some eigenpair* $(\lambda, \boldsymbol{v})$ *of* $\boldsymbol{A}$ *such that,*

$$\|\boldsymbol{x}_{j+1} - \boldsymbol{v}\|_2 = \mathcal{O}(\|\boldsymbol{x}_j - \boldsymbol{v}\|_2^3), \qquad \|\mu_{j+1} - \lambda\|_2 = \mathcal{O}(\|\mu_j - \lambda\|_2^3)$$

The result: convergence is cubic. (!!!!!)

I.e., if we reach a stage where $(\mu_j, \boldsymbol{v}_j)$ has one digit of accuracy, then $(\mu_{j+1}, \boldsymbol{v}_{j+1})$ will have 3 digits of accuracy, and $(\mu_{j+2}, \boldsymbol{v}_{j+2})$ will have *nine*. (!!!!!)

Hence, Rayleigh iteration is an *exceptional* algorithm for computing the spectrum.

Paired with deflation, one can compute the whole spectrum very efficiently.

# A "Rayleigh" version of the QR algorithm?



Power iteration was our first algorithm for computing eigenvalues.
The QR algorithm was a stable way to perform (simultaneous) power iteration.

We now have a "better" algorithm, Rayleigh iteration.
Is there a stable way to perform inverse iteration (+shifts), like the QR algorithm?

# A warmup: The QR algorithm for inverse iteration, I



Recall the ("pure") QR algorithm with $\boldsymbol{A}_0 = \boldsymbol{A}$:

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

and that,

$$\boldsymbol{A}_k = (\boldsymbol{Q}^{(k)})^* \boldsymbol{A} \boldsymbol{Q}^{(k)}, \qquad \boldsymbol{A}^k \overset{\text{QR}}{=} \boldsymbol{Q}^{(k)} \boldsymbol{R}^{(k)}$$

# A warmup: The QR algorithm for inverse iteration, I

Recall the ("pure") QR algorithm with $\boldsymbol{A}_0 = \boldsymbol{A}$:

$$\boldsymbol{A}_j \stackrel{\mathrm{QR}}{=} \boldsymbol{Q}_j \boldsymbol{R}_j, \qquad \boldsymbol{A}_{j+1} = \boldsymbol{R}_j \boldsymbol{Q}_j = \boldsymbol{Q}_j^* \boldsymbol{A}_j \boldsymbol{Q}_j,$$

and that,

$$\boldsymbol{A}_k = (\boldsymbol{Q}^{(k)})^* \boldsymbol{A} \boldsymbol{Q}^{(k)}, \qquad \boldsymbol{A}^k \stackrel{\mathrm{QR}}{=} \boldsymbol{Q}^{(k)} \boldsymbol{R}^{(k)}$$

By taking inverses, we can actually see how this is related to inverse iteration:

$$\boldsymbol{A}^{-k} = (\boldsymbol{R}^{(k)})^{-1} (\boldsymbol{Q}^{(k)})^* \quad \Longrightarrow \quad (\boldsymbol{A}^{-T})^k = \boldsymbol{Q}_*^{(k)} (\boldsymbol{R}^{(k)})^{-T}$$

where $\boldsymbol{Q}_*^{(k)}$ is the componentwise complex conjugate of $\boldsymbol{Q}^{(k)}$, and like $\boldsymbol{Q}^{(k)}$ is unitary.

# A warmup: The QR algorithm for inverse iteration, II

We now introduce a permutation matrix $\mathbf{\Pi} \in \mathbb{C}^{n \times n}$ with ones on the antidiagonal: $(\Pi)_{j,n-j+1} = 1$.

I.e., $\mathbf{\Pi}\boldsymbol{x}$ reverses (geometrically flips) the entries of $\boldsymbol{x}$.
$\mathbf{\Pi}\boldsymbol{x} = (x_n, x_{n-1}, \ldots, x_2, x_1)$.

And we also have that $\mathbf{\Pi}^{-1} = \mathbf{\Pi}^* = \mathbf{\Pi}$, i.e., $\mathbf{\Pi}^2 = \boldsymbol{I}$.

# A warmup: The QR algorithm for inverse iteration, II

We now introduce a permutation matrix $\mathbf{\Pi} \in \mathbb{C}^{n \times n}$ with ones on the antidiagonal: $(\Pi)_{j,n-j+1} = 1$.

I.e., $\mathbf{\Pi}\boldsymbol{x}$ reverses (geometrically flips) the entries of $\boldsymbol{x}$.
$\mathbf{\Pi}\boldsymbol{x} = (x_n, x_{n-1}, \ldots, x_2, x_1)$.

And we also have that $\mathbf{\Pi}^{-1} = \mathbf{\Pi}^* = \mathbf{\Pi}$, i.e., $\mathbf{\Pi}^2 = \boldsymbol{I}$.

With this matrix, we have:

$$(\boldsymbol{A}^{-T})^k \mathbf{\Pi} = \underbrace{\boldsymbol{Q}_*^{(k)} \mathbf{\Pi}}_{\text{unitary}} \underbrace{\mathbf{\Pi} (\boldsymbol{R}^{(k)})^{-T} \mathbf{\Pi}}_{\text{upper triangular}}$$

# A warmup: The QR algorithm for inverse iteration, II

We now introduce a permutation matrix $\mathbf{\Pi} \in \mathbb{C}^{n \times n}$ with ones on the antidiagonal: $(\Pi)_{j,n-j+1} = 1$.

I.e., $\mathbf{\Pi}\boldsymbol{x}$ reverses (geometrically flips) the entries of $\boldsymbol{x}$.
$\mathbf{\Pi}\boldsymbol{x} = (x_n, x_{n-1}, \ldots, x_2, x_1)$.

And we also have that $\mathbf{\Pi}^{-1} = \mathbf{\Pi}^* = \mathbf{\Pi}$, i.e., $\mathbf{\Pi}^2 = \boldsymbol{I}$.

With this matrix, we have:

$$(\boldsymbol{A}^{-T})^k \mathbf{\Pi} = \underbrace{\boldsymbol{Q}_*^{(k)} \mathbf{\Pi}}_{\text{unitary}} \underbrace{\mathbf{\Pi} (\boldsymbol{R}^{(k)})^{-T} \mathbf{\Pi}}_{\text{upper triangular}}$$

I.e., the pure QR algorithm actually computes the Q factor for $\boldsymbol{A}^{-T}$, i.e., it performs inverse iteration as well!

(More precisely, the componentwise conjugate of $\boldsymbol{Q}^{(k)}$, with its columns reversed, is the same as the $Q$ factor for inverse iteration on $\boldsymbol{A}^T$.)

In particular, the last column of $\boldsymbol{Q}^{(k)}$ has the (conjugated) first column of inverse iteration on $\boldsymbol{A}^T$.

# A second step: Inverse iteration with shifts



We have learned that the QR algorithm is both power iteration and (essentially) also inverse iteration.

How do we build in shifts? Suppose that $\mu_1, \mu_2, \ldots,$ are *a priori* known shifts. Consider the following *shifted* QR algorithm:

0. Let $j = 1$, $\boldsymbol{A}_1 = \boldsymbol{A}$.
1. Define $\boldsymbol{A}_j - \mu_j \boldsymbol{I} \stackrel{\mathrm{QR}}{=} \boldsymbol{Q}_j \boldsymbol{R}_j$.
2. Compute $\boldsymbol{A}_{j+1} := \boldsymbol{R}_j \boldsymbol{Q}_j + \mu_j \boldsymbol{I}$
3. Set $j \leftarrow j + 1$, return to step 1.

# A second step: Inverse iteration with shifts

We have learned that the QR algorithm is both power iteration and (essentially) also inverse iteration.

How do we build in shifts? Suppose that $\mu_1, \mu_2, \ldots,$ are *a priori* known shifts. Consider the following *shifted* QR algorithm:

0. Let $j = 1$, $\boldsymbol{A}_1 = \boldsymbol{A}$.
1. Define $\boldsymbol{A}_j - \mu_j \boldsymbol{I} \stackrel{\text{QR}}{=} \boldsymbol{Q}_j \boldsymbol{R}_j$.
2. Compute $\boldsymbol{A}_{j+1} := \boldsymbol{R}_j \boldsymbol{Q}_j + \mu_j \boldsymbol{I}$
3. Set $j \leftarrow j + 1$, return to step 1.

Some exercises, similar to what is done for the pure QR algorithm, reveal:

$$\boldsymbol{A}_{j+1} = \boldsymbol{Q}_j^* \boldsymbol{A}_j \boldsymbol{Q}_j, \qquad \boldsymbol{Q}^{(k)} = \prod_{j=1}^{k} \boldsymbol{Q}_j = \boldsymbol{Q}_1 \cdots \boldsymbol{Q}_k, \qquad \boldsymbol{R}^{(k)} = \prod_{j=k}^{1} \boldsymbol{R}_j = \boldsymbol{R}_k \cdots \boldsymbol{R}_1,$$

where $\boldsymbol{Q}^{(k)}$ and $\boldsymbol{R}^{(k)}$ satsify,

$$\prod_{j=k}^{1} (\boldsymbol{A} - \mu_j \boldsymbol{I}) = (\boldsymbol{A} - \mu_k \boldsymbol{I}) \cdots (\boldsymbol{A} - \mu_1 \boldsymbol{I}) = \boldsymbol{Q}^{(k)} \boldsymbol{R}^{(k)},$$

# QR for shifted inverse iteration

We have established that:

- The QR algorithm perform inverse iteration
- The shifted QR algorithm therefore performs shifted inverse iteration (on a transpose of $\boldsymbol{A}$)

Recall that our goal, Rayleigh iteration, simply defines the shifts through the Rayleigh quotient. How do we compute Rayleigh quotients?

## QR for shifted inverse iteration



We have established that:

- The QR algorithm perform inverse iteration
- The shifted QR algorithm therefore performs shifted inverse iteration (on a transpose of $\boldsymbol{A}$)

Recall that our goal, Rayleigh iteration, simply defines the shifts through the Rayleigh quotient. How do we compute Rayleigh quotients?

The somewhat surprising answer: we essentially get them for free:

$$(A_k)_{n,n} = \boldsymbol{e}_n^* \boldsymbol{A}_k \boldsymbol{e}_n = \boldsymbol{e}_n^* (\boldsymbol{Q}^{(k)})^* \boldsymbol{A} \boldsymbol{Q}^{(k)} \boldsymbol{e}_n = \boldsymbol{e}_n^T (\boldsymbol{Q}^{(k)})^T \boldsymbol{A}^T (\boldsymbol{Q}^{(k)})_* \boldsymbol{e}_n = R_{\boldsymbol{A}^T}(\boldsymbol{y}),$$

where $\boldsymbol{y}$ is the last column of the componentwise conjugate of $\boldsymbol{Q}^{(k)}$, which is precisely shifted inverse iteration's estimate of the dominant eigenvector.

# The QR algorithm with shifts

The following is the QR algorithm with *Rayleigh quotient* shifts:

0. Let $j = 1$, $\boldsymbol{A}_1 = \boldsymbol{A}$.
1. Define $\mu_j = (A_j)_{n,n}$.
2. Define $\boldsymbol{A}_j - \mu_j \boldsymbol{I} \stackrel{\mathrm{QR}}{=} \boldsymbol{Q}_j \boldsymbol{R}_j$.
3. Compute $\boldsymbol{A}_{j+1} := \boldsymbol{R}_j \boldsymbol{Q}_j + \mu_j \boldsymbol{I}$
4. Set $j \leftarrow j + 1$, return to step 1.

This algorithm performs (very well-disguised) Rayleigh iteration.

# The QR algorithm with shifts



The following is the QR algorithm with *Rayleigh quotient* shifts:

0. Let $j = 1$, $\boldsymbol{A}_1 = \boldsymbol{A}$.
1. Define $\mu_j = (A_j)_{n,n}$.
2. Define $\boldsymbol{A}_j - \mu_j \boldsymbol{I} \stackrel{\mathrm{QR}}{=} \boldsymbol{Q}_j \boldsymbol{R}_j$.
3. Compute $\boldsymbol{A}_{j+1} := \boldsymbol{R}_j \boldsymbol{Q}_j + \mu_j \boldsymbol{I}$
4. Set $j \leftarrow j + 1$, return to step 1.

This algorithm performs (very well-disguised) Rayleigh iteration.

This algorithm can fail (to converge) if the shift happens to be equisdistant between dominant eigenvalues. There are other (slightly more complicated) choices of shifts that are much more robust.

# The QR algorithm with shifts

The following is the QR algorithm with *Rayleigh quotient* shifts:

0. Let $j = 1$, $\boldsymbol{A}_1 = \boldsymbol{A}$.
1. Define $\mu_j = (A_j)_{n,n}$.
2. Define $\boldsymbol{A}_j - \mu_j \boldsymbol{I} \stackrel{\text{QR}}{=} \boldsymbol{Q}_j \boldsymbol{R}_j$.
3. Compute $\boldsymbol{A}_{j+1} := \boldsymbol{R}_j \boldsymbol{Q}_j + \mu_j \boldsymbol{I}$
4. Set $j \leftarrow j + 1$, return to step 1.

This algorithm performs (very well-disguised) Rayleigh iteration.

This algorithm can fail (to converge) if the shift happens to be equisdistant between dominant eigenvalues. There are other (slightly more complicated) choices of shifts that are much more robust.

This algorithm requires deflation for practicality (otherise everything continues to focus on the dominant eigenvalue).

When the dominant eigenvalue is found, then the last row of $\boldsymbol{A}_j$ is zero, except for the last entry (which contains the eigenvalue).

To deflate, one may slice the last row + column off and continue.

# Beyond the QR algorithm

The QR algorithm (with wisely chosen shifts) was the gold standard for eigenvalue computation in the 20th century.

Recently, new types of algorithms (divide-and-conquer algorithms and relatively robust representation, RRR, algorithms) are also used.

The main advantage of these alternatives was typically in either complexity or storage, but the gains are not in terms of asymptotic behavior, and instead in terms of decreasing the constants in those asymptotic behaviors.

Neverthless, LAPACK, the set of routines used nearly ubiquitously for numerical computing, still has polished QR-with-shifts eigenvalue algortihms that are still used.

# References I




- 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.