Numerical Eigenvalue Algorithms

Jayden Li

Linear Algebra 2024-25

0 Introduction

Let A be any $n \times n$ matrix. Consider the equation $Ax = \lambda x$: calculating eigenvalues λ and eigenvectors x for a large matrix is a challenging problem. The naive algorithm of solving the roots of the characteristic polynomial $\det(A - \lambda I) = 0$ is easy to compute on small matrices, but hard to implement in general. Most polynomial solvers, including NumPy's `roots` function, calculate the eigenvalues of the companion matrix of the coefficients of the characteristic polynomial [Num].

We will derive the Power Method, and show why it breaks down in certain cases. Then, we introduce a very basic QR algorithm, which suffers from similar limitations to the Power Method. Finally, we will see the algorithm actually used by libraries to compute eigenvalues.

In the end, we can compare the performance of all algorithms, in a case where the less advanced algorithms do not break down. Since there are too many variables affecting time, including the choice of starting vector, number of iterations and computer background activity, we will compare by a simple metric: **floating point operations**. Many advanced algorithms save time by performing less floating point operations per iteration, and it seems fair to judge them by this metric.

1 Power Method

Power Method

1. Let x_0 be a random vector in $\mathbb{R}^n$.
2. Define $x_{i+1} = \dfrac{Ax_i}{||Ax_i||}$. Then $x_k \neq A^k x_0$, but are pointing in the same direction.
3. Iterate the above process for m steps. Now, we have have calculated x_m, which is parallel to $A^m x_0$.
4. Calculate the dominant eigenvalue $\lambda_1 = \dfrac{\|AA^m x_0\|}{\|A^m x_0\|} = \dfrac{\|Ax_m\|}{\|x_m\|} = \dfrac{Ax_m \cdot x_m}{x_m \cdot x_m}$.

Reference: [Koc25].

We can try to understand why this algorithm works. Since we are repeating the process for a large number of iterations, let us examine the behavior of this algorithm as the number of iterations tends to infinity. Reference: [Bin16].

Let $A = PDP^{-1}$ be an $n \times n$ diagonalizable matrix. Then:

$$A^k = \left(PDP^{-1}\right)^k = PD^kP^{-1} \implies A^kP = PD^k \tag{1}$$

Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of A, such that $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$.

Let $v_0 \in \mathbb{R}^n$ be a random vector, let $v = Pv_0$, let v_{0_i} be the ith component of v_0, and let p_i be the ith column of P. Since P is invertible, v is essentially another random vector. p_i must be an eigenvector of A, since P is the matrix of eigenvectors. Multiplying on the left of equation (1), we have:

$$A^k P v_0 = PD^k v_0 = A^k v = P \begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{bmatrix} \begin{bmatrix} v_{0_1} \\ \vdots \\ v_{0_n} \end{bmatrix} = \begin{bmatrix} p_1 & \dots & p_n \end{bmatrix} \begin{bmatrix} \lambda_1^k v_{0_1} \\ \vdots \\ \lambda_n^k v_{0_n} \end{bmatrix} = \sum_{i=1}^n \lambda_i^k v_{0_i} p_i$$
$$= \sum_{i=1}^n \lambda_1^k \frac{\lambda_i^k}{\lambda_1^k} v_{0_i} p_i = \lambda_1^k \sum_{i=1}^n \left(\frac{\lambda_i}{\lambda_1}\right)^k v_{0_i} p_i$$

<u>Suppose the dominant eigenvalue λ_1 is greater than all other eigenvalues: $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n|$</u>*, so $0 \leq \lambda_i/\lambda_1 < 1$ for all $1 \leq i \leq n$. Thus, as $k \to \infty$, $(\lambda_i/\lambda_1)^k \to 0$.

$$\lim_{k\to\infty} A^k v = \lim_{k\to\infty} \lambda_1^k \sum_{i=1}^n \left(\frac{\lambda_i}{\lambda_1}\right)^k v_{0_i} p_i = \lim_{k\to\infty} \lambda_1^k \left(\cancel{\left(\frac{\lambda_1}{\lambda_1}\right)^k} v_{0_1} p_1 + \cancel{\sum_{i=2}^n \left(\frac{\lambda_i}{\lambda_1}\right)^k v_{0_i} p_i} \right) = \lim_{k\to\infty} \lambda_1^k v_{0_1} p_1$$

Therefore, as k tends to infinity, $A^k v$ is parallel to p_1 as λ_1 and v_{0_1} are scalar values. Recall that p_1 is an eigenvalue of A, associated with the eigenvalue λ_1.

Let us note that if (x, λ) is an eigenpair of A, a scalar multiple cx is also an eigenvector associated with the same eigenvalue λ: $A(cx) = c(Ax) = c(\lambda x) = \lambda(cx)$.

Thus, $p_1 = A^k v$ are eigenvectors of A, both associated with the same eigenvalue λ_1. When performing the Power Method on a matrix, we do not know p_1, but we can calculate $A^k v$, and for higher values of k becomes "almost" parallel to eigenvector p_1.

Suppose $A^k v$ is an eigenvector, then we can easily calculate the eigenvalue λ_1.

$$A\left(A^k v\right) = \lambda_1 \left(A^k v\right) \implies \left\|A\left(A^k v\right)\right\| = \left\|\lambda_1\left(A^k v\right)\right\| = \lambda_1 \left\|A^k v\right\| \implies \lambda_1 = \frac{\|A(A^k v)\|}{\|A^k v\|} = \boxed{\frac{A(A^k v) \cdot (A^k v)}{(A^k v)\cdot(A^k v)}}$$

We will prove the equality used above:

$$\frac{\|A(A^k v)\|}{\|A^k v\|} = \frac{A(A^k v) \cdot (A^k v)}{(A^k v)\cdot(A^k v)} \tag{2}$$
$$\iff \frac{\|A(A^k v)\|}{\|A^k v\|} = \frac{A(A^k v) \cdot (A^k v)}{\|A^k v\|^2} \iff \left\|A\left(A^k v\right)\right\| \left\|A^k v\right\| = A\left(A^k v\right) \cdot \left(A^k v\right)$$

Which is true by the equality case of the Cauchy-Schwarz Inequality, since the two vectors $A\left(A^k v\right)$ and $A^k v$ are linearly dependent. As seen earlier, $A^k v$ is an eigenvector of A as $k \to \infty$, so $A\left(A^k v\right) = \lambda_1 A^k v$ is a scalar multiple of $A^k v$. $\square$

Let us return to the condition marked by an asterisk above. If that assumption is not met, i.e. $|\lambda_1| = |\lambda_2|$. Let us suppose for now that $|\lambda_2| > |\lambda_3|$.

$$\lim_{k\to\infty} A^k v = \lim_{k\to\infty} \lambda_1^k \sum_{i=1}^n \left(\frac{\lambda_i}{\lambda_1}\right)^k v_{0_i} p_i = \lim_{k\to\infty} \lambda_1^k \left(\cancel{\left(\frac{\lambda_1}{\lambda_1}\right)^k} v_{0_1} p_1 + \cancel{\left(\frac{\lambda_2}{\lambda_1}\right)^k} v_{0_2} p_2 + \cancel{\sum_{i=3}^n \left(\frac{\lambda_i}{\lambda_1}\right)^k v_{0_i} p_i} \right)$$
$$= \lim_{k\to\infty} \lambda_1^k \left(v_{0_1} p_1 + v_{0_2} p_2\right)$$

If $\lambda_1 \neq \lambda_2$, $\lim_{k\to\infty} A^k$ is not an eigenvector of A.

Furthermore, let us consider the case where the dominant eigenvalue of a real matrix A is a complex number $a + bi$. Complex eigenvalues appear in complex conjugate pairs (since if $a + bi$ is a root of the characteristic polynomial, so must $a - bi$), so $a - bi$ is also an eigenvalue [Fre]. But notice that $|a + bi| = \sqrt{a^2 + b^2} = |a - bi|$, so the algorithm will break down. In fact, if the dominant eigenvalue is complex, the algorithm will not work.

2 Basic QR Algorithm

Schur form

Let A be an $n \times n$ real matrix. The **Schur form** of A is:

$$A = QUQ^T = QUQ^{-1}$$

where Q is orthogonal and U is upper triangular. A and U are similar so they share the same eigenvalues, which are the diagonal entries of U.

For some $n \times n$ matrix A, the QR Algorithms compute the Schur form of A. Once we calculate the triangular matrix T, it is trivial to find the eigenvalues of A.

Basic QR Algorithm

1. Set $A_1 = A$.
2. Define $A_{i+1} = R_i Q_i$, where $Q_i R_i$ is the QR factorization of A_i.
3. Iterate step 2 m times to calculate A_{m+1}.
4. If the eigenvalues of A are real, then for large values of m, A_{m+1} converges to the triangular matrix T in the Schur form $A = QTQ^T$. The eigenvalues of A are the diagonal entries of A_{m+1}.

Reference: [Arb16, p. 64].

Let us try to understand this algorithm in some very specific cases. If A is a real matrix with complex eigenvalues, the Basic QR Algorithm will not converge upon a triangular matrix. Let us examine the case where A is a real matrix, with only real eigenvalues. Reference: [Fal].

We define $A_{i+1} = R_i Q_i$, such that $A_i = Q_i R_i$ is a QR factorization. Since Q_i is orthogonal, $R_i = Q_i^T A_i$.

$$A_{i+1} = Q_i^T A_i Q_i \tag{3}$$

Furthermore, notice $A_i = R_{i-1} Q_{i-1} = Q_{i-1}^T A_{i-1} Q_{i-1}$. Substituting into equation (3):

$$A_{i+1} = Q_i^T Q_{i-1}^T A_{i-1} Q_{i-1} Q_i$$

Let $P_i = Q_1 Q_2 \ldots Q_i$. In general, I claim:

$$A_{i+1} = P_i^T A_1 P_i \tag{4}$$

Proof. We will prove equation (4) by induction.

Base case. When $i = 1$, $P_i = Q_1$. Then $P_i^T A_1 P_i = Q_1^T A_1 Q_1 = R_1 Q_1 = A_2 = A_{i+1}$.

Inductive step. Suppose $A_i = P_{i-1}^T A_1 P_{i-1}$. Let Q_i, R_i be the QR factorization of A_i, such that $A_i = Q_i R_i$, and $R_i = Q_i^T A_i$. Recall the definition:

$$A_{i+1} = R_i Q_i = Q_i^T A_i Q_i = Q_i^T \left(P_{i-1}^T A_1 P_{i-1}\right) Q_i = \left(P_{i-1} Q_i\right)^T A_1 \left(P_{i-1} Q_i\right)$$

By definition, $P_i = Q_1 Q_2 \dots Q_{i-1} Q_i = P_{i-1} Q_i$, so:

$$A_{i+1} = P_i^T A_1 P^i$$

□

We find the diagonalization of $A = X \Lambda X^{-1}$, and the factorization $X = QR$.

$$A = X \Lambda X^{-1} = QR\Lambda R^{-1} Q^{-1} \implies Q^T A Q = R \Lambda R^{-1}$$

The right-hand side $R\Lambda R^{-1}$ is the product of three triangular matrices, and the product is also triangular. Furthermore, let us suppose $\lim\limits_{i\to\infty} P_i = Q$. Then:

$$\lim_{i\to\infty} A_i = \lim_{i\to\infty} P_i^T A P_i = R\Lambda R^{-1}$$

As m, the number of iterations in the Basic QR Algorithm, tends to infinity, the result of the QR iteration A_{m+1} tends to the triangular matrix $R\Lambda R^{-1}$. Notice that A_{m+1} is similar to A, since $\left(P_i^T\right)^{-1} = P_i$ as P_i is orthogonal, and thus **the eigenvalues of A are the diagonal entries of A_{m+1}, as $m \to \infty$.**

In general, QR iteration converges to a *block triangular* matrix:

$$\lim_{i\to\infty} A_i = \begin{bmatrix} B_1 & 0 & \dots & 0 \\ 0 & B_2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & B_k \end{bmatrix}$$

where $B_1, \dots B_k$ are square matrices. The proof is difficult and is outside the scope of this project. $\lim\limits_{i\to\infty} A_i$ is the triangular matrix in the Schur form of A depends on the assumption $\lim\limits_{i\to\infty} P_i = Q$, which is true if:

$$|\lambda_1| > |\lambda_2| > \dots > |\lambda_n| \tag{5}$$

where $\lambda_1, \dots \lambda_n$ are the eigenvalues of A. The Basic QR Algorithm converges to the upper triangular matrix U in the Schur form of A when the magnitudes of each eigenvalue are different.

It is interesting that the Basic QR Algorithm may produce the correct Schur form (and eigenvalues) for matrices not satisfying property (5), but may also break down (Section 2.2 and 2.3 of Jupyter Notebook).

3 New and Improved QR Algorithm

This is the algorithm used by the Linear Algebra PACKage (LAPACK), which is in turn used by libraries like NumPy and SciPy to calculate eigenvalues and eigenvectors.

Hessenberg matrix

A Hessenberg matrix H is one that is "almost" triangular: it is a triangular matrix with nonzero entries in the diagonal just above or just below its main diagonal. A **lower** Hessenberg matrix H satisfies $H_{ij} = 0$ for all $j > i+1$, and an **upper** Hessenberg matrix H satisfies $H_{ij} = 0$ for all $i > j+1$.

For example, H_l and H_u are 5×5 lower and upper Hessenberg matrices, respectively.

$$H_l = \begin{bmatrix} \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \qquad H_u = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}$$

Lapack Algorithm

1. Reduce A to an *upper Hessenberg* form by calculating $A = QHQ^T$, where Q is orthogonal and H is an upper Hessenberg matrix. H and A are similar and share the same eigenvalues.
2. Calculate the Schur form of H.

Reference: [Bla99].

By definition, Hessenberg matrices are more *sparse* (containing more zero entries) than the general matrix A, so QR factorizing and multiplying a Hessenberg matrix will be less computationally intensive. In fact, QR iteration on a Hessenberg matrix is only order $\mathcal{O}(n^2)$, compared to $\mathcal{O}(n^3)$ for general matrices.

Real Schur form

Let A be an $n \times n$ real matrix. The **real Schur form** of A is:

$$A = QUQ^T = QUQ^{-1}$$

where Q is orthogonal and U is an *upper quasi-triangular* matrix in block form:

$$U = \begin{bmatrix} B_1 & \times & \times & \times \\ 0 & B_2 & \times & \times \\ \vdots & \vdots & \ddots & \times \\ 0 & 0 & \dots & B_k \end{bmatrix}$$

where $B_1, B_2, \dots B_k$ are 1×1 or 2×2 real submatrices. A 1×1 submatrix contains a real eigenvalue, while a 2×2 submatrix B is in form:

$$B = \begin{bmatrix} a & b \\ -b & a \end{bmatrix}$$

which corresponds to the eigenvalues $a + bi$ and its conjugate $a - bi$, which is also an eigenvalue.

Step 2 (Schur decomposition) is an intricate and complicated procedure performed by Lapack's `xHSEQR` function. In particular, it uses the Francis double shift QR algorithm, which essentially "shifts" the intermediate matrix A_i from QR iteration by λI and its complex conjugate $\overline{\lambda} I$ every step [Bye07; Bin12]. The Francis double shift algorithm converges to the *real Schur form*, from where both real and complex eigenvalues can be easily calcualted.

Householder transformation

In the real numbers, an $n \times n$ Householder matrix is in the form

$$H = I_n - 2vv^T$$

where I_n is the $n \times n$ identity, and v is a unit vector in $\mathbb{R}^n$.

For all vectors $w \in \mathbb{R}^n$ and $w' \in \mathbb{R}^n$ where $\|w\| = \|w'\|$, there exists an $n \times n$ Householder matrix H such that $Hw = w'$. That is, there exists a Householder matrix mapping any vector w to another vector w of the same norm as w.

For a proof and reference for the above theorem, see [GH17].

3.1 Hessenberg Reduction

Step 1 (Hessenberg reduction) is somewhat simpler[1], so we can try explaining its steps. Reference: [Wik].

Let A be an $n \times n$ real matrix. Then, let A' be an $(n-1) \times n$ matrix consisting of the 2nd to nth row of A, and let $a'_1 \in \mathbb{R}^{n-1}$ be the first column of A'.

Now, we need to calculate the $(n-1) \times (n-1)$ matrix V_1 that maps a'_1 to a vector with only one component in the form $(\times, 0, 0, \ldots, 0)$. As long as the new vector has the same norm as a'_1, we can find a Householder matrix V_1 to accomplish this mapping.

$$V_1 = I_{n-1} - 2vv^T \qquad v = \frac{\operatorname{sgn}(a'_{11}) \|a'_1\| e_1 + a'_1}{\|\operatorname{sgn}(a'_{11}) \|a'_1\| e_1 + a'_1\|} \tag{6}$$

where I_{n-1} is the $(n-1) \times (n-1)$ identity, e_1 is the standard basis vector for $\mathbb{R}^{n-1}$: $(1, 0, 0, \ldots, 0)$, or the first row/column of the $(n-1) \times (n-1)$ identity, and a'_{11} is the first component of the vector a'_1. If we apply the transformation V_1 to a'_1, we will have a scalar multiple of e_1.

Note that V_1 is an $(n-1) \times (n-1)$ matrix. Define the following $n \times n$ block matrix U_1,

$$U_1 = \begin{bmatrix} 1 & 0 \\ 0 & V_1 \end{bmatrix}$$

The first column will be in the form $\begin{bmatrix} \times & \times & 0 & 0 & \ldots & 0 \end{bmatrix}^T$. This is what we want in a Hessenberg matrix, as for the first columns, there are no nonzero entries below the diagonal below the main diagonal ("the subdiagonal").

Now, we may repeat the process. However, remove **two** rows from the top of A, and one column from the left of A, and call this new matrix A'', which is $(n-2) \times (n-1)$. Then, take $a''_1 \in \mathbb{R}^{n-2}$ as the first column of A''. Calculate the Householder matrix V_2 using formula (6), and calculate U_2 in a fashion:

$$U_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & V_2 \end{bmatrix}$$

Then the first and second columns of $U_2 U_1 A$ has no nonzero entries below the subdiagonal.

Continue for a total of $n-2$ steps. Then, $H = U_{n-2} U_{n-3} \ldots U_2 U_1 A$ has no nonzero entries below the subdiagonal on all columns, and we have reduced a general matrix A to the the Hessenberg form. Also, the following is true:

$$H = U_{n-2} U_{n-3} \ldots U_2 U_1 A = U_{n-2} U_{n-3} \ldots U_2 U_1 A U_1 U_2 \ldots U_{n-3} U_{n-2}$$

[1]admittedly, "simpler than the Francis double-step algorithm" is a very low bar

And therefore, the Hessenberg matrix H is similar to A, because $U_i = U_i^{-1}$. Any QR Algorithm will take less floating point operations per step on a Hessenberg matrix, which has more zeros, than a general matrix A.

The implementation in the Jupyter notebook is slightly different and is based on an algorithm in [Per06].

4 Comparison

First, a breakdown of the outcome of each algorithm. It is easy to recover **every** eigenvalue of a matrix from its Schur form, since they are all listed on its diagonal.

Algorithm	Advantages	Disadvantages
Power Method	• Simple.	• Breaks down if there two eigenvalues with the same magnitude. • Only finds dominant eigenvalues. • Unable to find complex eigenvalues. • Need to use shifts to address above, which are complicated. • Convergence speed depends on starting vector.
Basic QR	• Finds all eigenvalues. • No starting vector to worry about.	• Needs modifications to find complex eigenvalues. • Similar problems to Power Method: no handling of multiple dominant eigenvalues.
Improved QR	• Finds all eigenvalues. • Works on all matrices. • Faster: lower big-$\mathcal{O}$.	• Complicated.

Table showing advantages and disadvantages of each algorithm.

Next, a comparison of the time taken. The Power Method is so drastically different from the QR Algorithms we will only compare the Basic and Improved Algorithms. Since the Francis double-step algorithm is too complicated, we will modify ("un-improve") the Improved Algorithm by performing a Hessenberg reduction, then applying the Basic QR Algorithm to the Hessenberg matrix.

We will perform the following procedure for a constant m number of iterations:

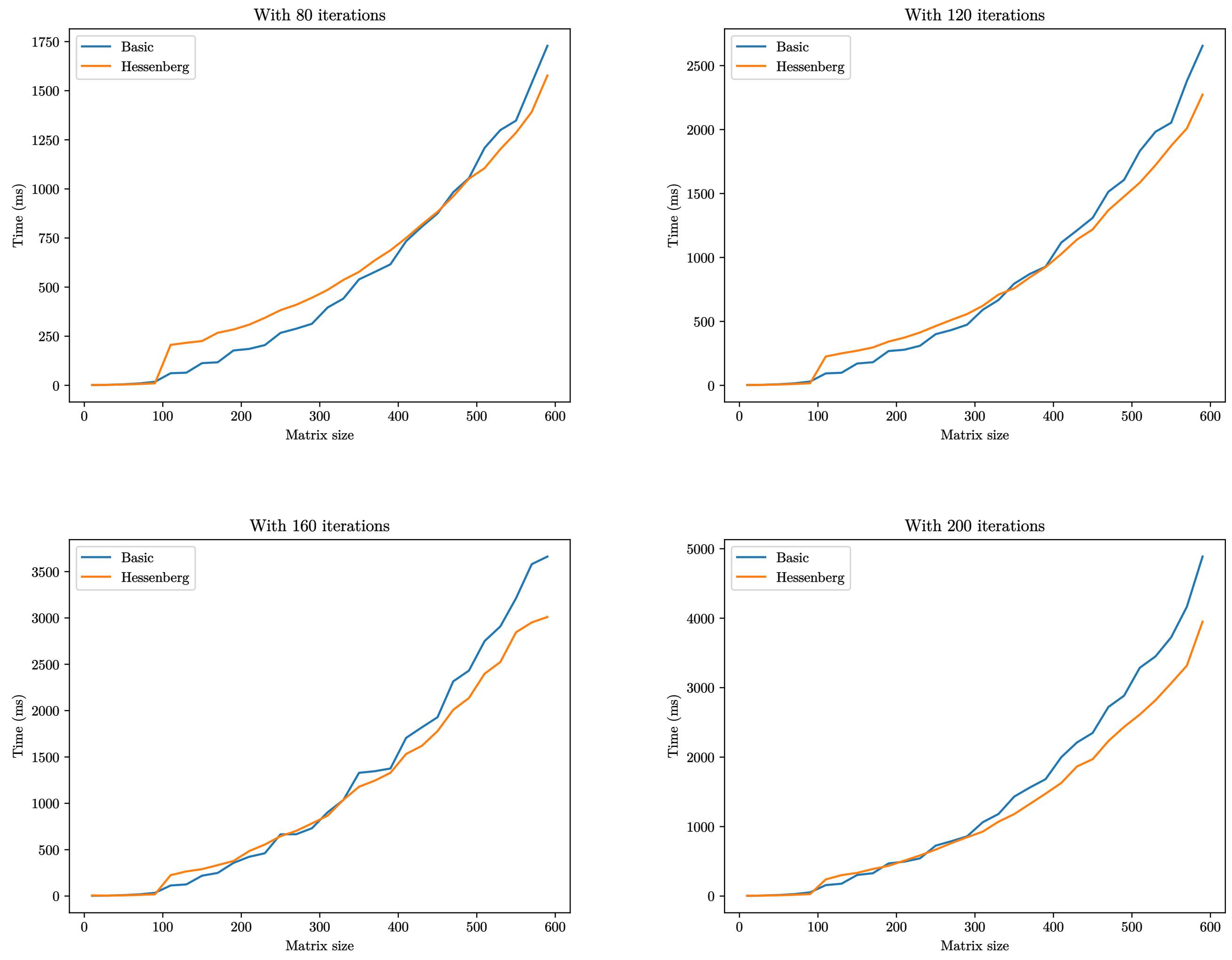
1. Set A to some random $n \times n$ matrix.

2. Calculate the time to run m iterations of the Basic QR Algorithm on A.

3. Calculate the time to calculate the Hessenberg matrix H from A, and to run m iterations of the Basic QR Algorithm on H.

4. Repeat steps 2 and 3 some number of times, then take the average of each elapsed time.

5. Increment n by some amount, then start back from step 1.



Time for m iterations of both QR Algorithms for different matrix sizes.

As seen above, the Hessenberg QR Algorithm is indeed faster for larger matrices and iteration counts. At lower iterations, the cost of Hessenberg reduction is higher than the savings of having a sparser matrix.

References

[Arb16] Peter Arbenz. *Lecture Notes on Solving Large Scale Eigenvalue Problems.* 2016. URL: `https://people.inf.ethz.ch/arbenz/ewp/Lnotes/lsevp.pdf`.

[Bin12] David Bindel. *Week 11: Monday, Oct 29.* Oct. 29, 2012. URL: `https://www.cs.cornell.edu/~bindel/class/cs6210-f12/notes/lec28.pdf`.

[Bin16] David Bindel. *Notes for 2016-10-17.* Oct. 17, 2016. URL: `https://www.cs.cornell.edu/~%20bindel/class/cs6210-f16/lec/2016-10-17.pdf`.

[Bla99] Susan Blackford. *Eigenvalues, Eigenvectors and Schur Factorization.* Oct. 1, 1999. URL: `https://www.netlib.org/lapack/lug/node50.html`.

[Bye07] Ralph Byers. *LAPACK 3.1 xHSEQR: Tuning and Implementation Notes on the Small Bulge Multi-shift QR Algorithm with Aggressive Early Deflation.* May 2007. URL: `https://www.netlib.org/lapack/lawnspdf/lawn187.pdf`.

[Fal] Richard Falk. *Math 574 Lecture Notes: Convergence of the QR algorithm.* URL: `https://sites.math.rutgers.edu/~falk/math574/lecture9.pdf`.

[Fre] Alex Freire. *Complex Eigenavlues of Real Matrices.* URL: `https://web.math.utk.edu/~freire/complex-eig2005`.

[GH17] Stephan Ramon Garcia and Roger A. Horn. *A Second Course in Linear Algebra.* Cambridge University Press, 2017.

[Koc25] Gregory Koch. *Lab #9: Power Method for Approximating Eigenvalues.* Feb. 10, 2025. URL: `https://peddie.instructure.com/courses/7772/assignments/182563`.

[Num] Numpy Documentation. *numpy.roots.* URL: `https://numpy.org/doc/2.2/reference/generated/numpy.roots.html`.

[Per06] Per-Olof Persson. *Lecture 14 Hessenberg/Tridiagonal Reduction.* Oct. 26, 2006. URL: `https://dspace.mit.edu/bitstream/handle/1721.1/75282/18-335j-fall-2006/contents/lecture-notes/lec14.pdf`.

[Wik] Wikipedia. *Hessenberg matrix.* URL: `https://en.wikipedia.org/wiki/Hessenberg_matrix`.