Topic: Eigenvalues

Nick Battista

Date Created: 6/25/2014
Date Modified: 7/27/16

1 REMEMBER EIGENVALUES?

Remember how much fun calculating eigenvalues using paper and pencil was in undergraduate Linear Algebra? There was computing determinants of matrices symbollically, forming a characteristic polynomial only to find it roots, and solving some singular systems of equations. Everything that should translate swimmingly to numerically finding eigenvalues, right? Literally, none of those things are exceedingly computer friendly.

The process of finding eigenvalues on a computer is going to have a very different feel and approach, than how one finds them by hand. Before we dive into the abstract details, we will first flat out state that numerically computing eigenvalues tends to be a messy business. Unlike solving systems of equations (which also have a very much more natural transition from pencil and paper to the *in silica* implementations), we will not be able to state *a priori* the computational cost of finding eigenvalues.

We will not be able to say the explicit cost of finding eigenvalues; however, will only be able to approximate the number of operations required. This is all to say, *numerical eigenvalue solvers need must be iterative*. We can easily observe this since at the heart of each eigenvalue problem lies a characteristic polynomial, to which we must find its roots, and finding roots of equations is an iterative process. Whatever our method, it will require an iterative flavor, and hence will introduce errors, outside of round-off errors.

Before we dive right into the numerical side of things, we will first refresh a bunch of eigenvalue identities, i.e., factorizations, properties, among other things, which we will put into our numerical eigenvalue toolbox.

2 EIGENVALUE TOOLBOX

This section will more or less be a list of eigenvalue identities, definitions, factorizations, and everything that is good…in the eigen-world. Hopefully this looks familiar, if not, please indulge me, and peruse this section with a smile.

2.1 SOME EIGEN-RELATED DEFINITIONS TO KEEP AROUND FOR A RAINY DAY…

Here it goes…let's go out and define!

1. λ is an eigenvalue $\leftrightarrow$ there exists $\mathbf{x}$ such that $\lambda\mathbf{x} - A\mathbf{x} = 0 \leftrightarrow \lambda I - A$ is *singular* $\leftrightarrow \det(\lambda I - A) = 0$.

2. *Definition:* The *geometric multiplicity* of an eigenvalue, λ, is the dimension of the nullspace of $\lambda I - A$, i.e., $(\lambda I - A)$. In short, it is the number of eigenvectors associated with one particular eigenvalue, λ.

3. *Definition:* The *algebraic multiplicity* of an eigenvalue, λ, of a matrix A, is the the multiplicity of λ as a root of the characteristic polynomial, e.g., $p_A(\lambda) = \det(\lambda I - A) = 0$ Furthermore, note that the above two definitions imply that

$$\text{algebraic mult.} \geq \text{geometric mult.}.$$

4. *Definition:* A matrix, A, is said to be *defective* when the algebraic multiplicity is strictly greater than the geometric multiplicity, that is, not enough linearly independent eigenvectors exist for a particular eigenvalue. *Spoiler*, we will not be able to factorize A into an eigenvalue decomposition, if A is defective. On that note, I'll also drop here that all eigenvectors are linearly independent, so if A is full rank, e.g., $(A) = \{\}$, then the eigenspace of A spans the same space as A. Stronger than that, though, we can say that the eigen-space of A spans the same space as A.

5. *Definition:* The determinant of a matrix A can be written as a product of its eigenvalues, e.g.,
$$\det(A) = \prod_{k=1}^{n} \lambda_k.$$

6. *Definition:* Similarly, the *trace* of a matrix A can be written as a sum of its eigenvalues, e.g.,
$$\text{tr}(A) = \sum_{k=1}^{n} \lambda_k.$$
We note the above two definitions may be useful in some applications, but we will not harness them, often, if at all in everyday life. It's best to keep them around in the toolbox, though.

2.2 EIGENVALUE DECOMPOSITIONS: THE BIG GUNS

If the above subsection was nails, screws, and bolts of eigenvalue problems, then this section we learn about the power tools. Matrix factorizations are very useful, as demonstrated when solving linear systems. We will restrict our discussion here to solely eigenvalue-type factorizations, e.g., eigenvalue decompositions. We will mention three different eigenvalue decompositions.

1. *Diagonalization*: We can write A as a product of three matrices,

   $$A = X\Lambda X^{-1}.$$

   if and only A is *non-defective*, meaning all of A's algebraic multiplicities = geometric multiplicities. The matrix X is composed of eigenvectors in their column form. The matrix Λ is a diagonal matrix, where it's diagonal is composed of the eigenvalues of A. We note if $\Lambda_{11} = \lambda_1$, then the first column of X is the eigenvector associated with λ_1, and so on.

   You are probably familiar with this eigenvalue decomposition from undergraduate Linear Algebra and for this, I apologize.

2. If $A \in \mathbb{R}^{n\times n}$ and has n orthogonal eigenvectors, then A has a *unitary diagonalization*, and we can write

   $$A = Q\Lambda Q^*.$$

   Similarly to matrix X above, Q is a matrix, whose columns are all the eigenvectors of A; however, in this case they are orthonormal, hence we have $QQ^* = Q^*Q = I$. Furthermore, the matrix Λ is a diagonal matrix, which contains the eigenvalues on the diagonal.

   We will note a few things.

   - This counts as both an eigenvalue decomposition and SVD, i.e.,

     $$A = Q\Lambda Q^* = U\sum V^{-1},$$

     where we let $U = Q$, $\sum = \Lambda$, and $V^{-1} = \text{sign}(\Lambda)Q^*$.
   - Hermitian matrices, i.e., $A = A^*$, are unitary diagonalizable; e.g., all of their eigenvectors are orthogonal (orthonormal) and eigenvalues are real-valued! Let's prove these claims!

     Proof. We will now prove that all eigenvalues are real and that their associated eigenvectors are orthogonal.

a) First we will prove that all the eigenvalues of a Hermitian matrix are real. To do this we will recall the infamous eigenvalue equation,

$$A\mathbf{x} = \lambda\mathbf{x}.$$

To isolate λ by itself we multiply both sides by $\mathbf{x}^*$, and solve for λ, e.g.,

$$\lambda = \frac{\mathbf{x}^* A\mathbf{x}}{\mathbf{x}^*\mathbf{x}}.$$

Now we are ready for the final step, ,

$$\lambda^* = \left(\frac{\mathbf{x}^* A\mathbf{x}}{\mathbf{x}^*\mathbf{x}}\right)^* = \frac{\mathbf{x}^* A^*\mathbf{x}}{\mathbf{x}^*\mathbf{x}} = \frac{\mathbf{x}^* A\mathbf{x}}{\mathbf{x}^*\mathbf{x}} = \lambda,$$

and hence $\lambda \in \mathbb{R}$.

b) We now will prove all the eigenvectors are orthogonal. Assume thatλ and μ are distinct eigenvalues associated eigenvectors $\mathbf{x}$ and $\mathbf{y}$, respectively. Hence we will now show that $<\mathbf{x},\mathbf{y}>=0$

To begin we consider,

$$\lambda <\mathbf{x},\mathbf{y}>=<\lambda\mathbf{x},\mathbf{y}>=<A\mathbf{x},\mathbf{y}>.$$

Now using properties of inner products, we can write

$$<A\mathbf{x},\mathbf{y}>=<\mathbf{x},A^*\mathbf{y}>.$$

However, since $A = A^*$, we get

$$<\mathbf{x},A^*\mathbf{y}>=<\mathbf{x},A\mathbf{y}>=<\mathbf{x},\mu\mathbf{y}>$$

Using another property of inner products, we can pull out a scalar by

$$<\mathbf{x},\mu\mathbf{y}>=\mu^* <\mathbf{x},\mathbf{y}>=\mu<\mathbf{x},\mathbf{y}>.$$

Hence we have

$$\lambda <\mathbf{x},\mathbf{y}>=\mu<\mathbf{x},\mathbf{y}>,$$

but since we assumed $\mu \neq \lambda$, this implies that $<\mathbf{x},\mathbf{y}>=0$.

I have to admit this is a more long winded way to show this property of Hermitian matrices. Since we are assuming A is full rank, we can diagonalize A, e.g., let

$$A = X\Lambda X^{-1},$$

and now since $A = A^*$, we have

$$A^* = (X\Lambda X^{-1})^* = (X^{-1})^*\Lambda^* X^* = (X\Lambda X^{-1})^* = (X^{-1})^*\Lambda X^*,$$

since all the eigenvalues are real. However this must be equal to the original diagonalization of A because A is Hermitian. Therefore we have that

$$X = (X^{-1})^* \quad \text{and} \quad X^{-1} = X^*.$$

The only way this above statement could be true is if $X^* = X^{-1}$, which makes X an orthogonal matrix, and our result falls out.

□

In fact, we can extend this to all matrices that are normal!

Theorem 1. *A matrix is unitary diagonalizable if and only if it is normal.*

Recall that a matrix, A, is *normal* if $AA^* = A^*A$.

3. Regardless of the spectral properties of A, we can write it in its *Schur form.* The Schur form of a matrix is where A is decomposed as,

$$A = QTQ^*.$$

where Q is an orthogonal matrix and T is an upper-triangular matrix. Please note that Q is not, in general, composed of eigenvectors, as in the unitary diagonalization case. Moreover, upper-triangular matrix T is in the *Jordan form*, but still maintains all the eigenvalues of A along its main diagonal. This is because A and T are *similar*. We now define what that means.

- *Definition*: Two matrices, A and B, are similar if and only if there exists non-singular matrix P, such that we can write $A = PBP^{-1}$. Furthermore if two matrices are similar, then they have the same eigenvalues. A clear example of this is the diagonalization of a matrix.

Moreover, because of the *Jordan form* of T, every matrix has a Schur Factorization. That's important enough to write as a theorem, give me a second.

Theorem 2. *Every square matrix has a Schur Factorization.*

Boom. Why is this so important? Well as we approach numerically finding eigenvalues, we will almost exclusively use the Schur Factorization. The only time we won't is typically when A is Hermitian, and we see the Schur Factorization becomes a Unitary Diagonalization of A, that is, T is diagonal. Also, whenever we have orthogonal matrices involved, computations tend to have nice stability properties, from the backward stability of how one will compute Q, i.e., using Householder Reflectors or Givens Rotations.

Before jumping head first into a bunch of algorithms, we will give a brief overview of where every algorithm fits into the mix. Please do not be intimidated by the names, but simply sit back and welcome some new key words and connections into your brain.

3 THE OVERVIEW: THE LONG AND THE SHORT OF IT

This section's goal is not to confuse or intimidate you, if that happens, I am all apologies, but to simply put all the methods into a context and where they fit into the world of standard eigenvalue schemes. We will not go into detail about each method here, but will just say one or two useful notes that differentiate them from each other. Here comes the list!

1. The *Rayleigh Quotient*, or *RQ*, can be used to find an exact eigenvalue, if the true eigenvector is known. Furthermore, if an approximation to an eigenvector is known, then we can use the *RQ* to find the eigenvalue that would be associated with it, that is, we can find an approximate eigenvalue if we have an approximate eigenvector. s

2. There are two methods that can only find one eigenvector - the *Power Iteration* and *Inverse Power Iteration*.
   a) The *Power Method* only finds the eigenvector associated with the largest eigenvalue in magnitude. That is the only eigenvector it can find.
   b) The *Inverse Power Method* only, too, finds one eigenvector at a time; however, it finds the eigenvector associated with eigenvalue that is nearest to the shift, μ, that is an input to the algorithm. In short, we have *a priori* information that μ is close to an eigenvalue, so using μ we are able to find the eigenvector associated with the eigenvalue closest to μ. However, we have find the whole spectrum of a matrix using this method...with enough patience and persistence.
   c) Once we have an approximate eigenvector, coming from either the Power Method or Inverse Power Iteration, we employ the *RQ* to find the associated approximate eigenvalue.

3. The *QR Algorithm* is able to find all the eigenvalues of a matrix at once. However, it can be both computationally expensive and slow, if not performed optimally. There are two routes one can use when using the *QR Algorithm*, which we will briefly put into context here.

Route 1 : Apply QR algorithm directly to the matrix $A \in \mathbb{R}^{N\times N}$. This is *slow* and *expensive*. Basically the overall cost will be computing the QR factorization of a matrix at each iteration, which costs $\mathcal{O}(N^3)$, so if the algorithm iterates m-times, the cost be will $\sim \frac{4}{3}mN^3$. This may not seem like a big deal, but m could potentially be rather large, making the potential overall cost for the algorithm $\mathcal{O}(N^4)$.

Route 2 : Before applying the QR algorithm right away to A, we will massage it into upper-Hessenberg form, H, and then apply the QR algorithm in two steps, e.g.,

Step 1 : Massage the matrix to upper Hessenberg form, H, using Householder Reflectors or Givens Rotations, which costs $\mathcal{O}(N^3)$. Note, if A is exceptionally large, one can use a Krylov method, i.e., the *Arnoldi* iteration, or *Lanczos* iteration, if the matrix is Hermitian, to transform the matrix to upper Heisenberg form. Furthermore, this step is backward stable!

Step 2 : Apply the QR algorithm to the upper Heisenberg matrix, H. Each step will now cost $\mathcal{O}(N^2)$, making these two steps computationally less expensive and faster than simply applying the QR algorithm to the original matrix A.

The long and the short of it is, if you only care about one particular eigenvalue, use the Power Iteration or Inverse Power Iteration, but if you need the full spectrum of A, use the QR algorithm and use *Route 2*, that is the two step eigenvalue solving scheme.

We will now proceed with the first algorithms anyone learns when computing eigenvalues and eigenvectors numerically - *Rayleigh Quotient*, *Power Iteration*, and *Inverse Power Iteration*.

4 Rayleigh Quotient, Power Iteration, and Inverse Power Iteration - oh my!

Let's begin the algorithmic discussion by analyzing the methods that can only give you one eigenvector, or eigenvalue, either at a time, or in total. We will perform error analysis, as well, so get ready for some fun! Our journey begins with the Rayleigh Quotient.

4.1 Rayleigh Quotient

The Rayleigh Quotient pops right out of the eigenvalue equation, $A\mathbf{x} = \lambda\mathbf{x}$, and solving for λ, i.e.,

$$\begin{aligned} \lambda\mathbf{x} &= A\mathbf{x} \\ \lambda(\mathbf{x}^*\mathbf{x}) &= \mathbf{x}^* A\mathbf{x} \\ \lambda &= \frac{\mathbf{x}^* A\mathbf{x}}{\mathbf{x}^*\mathbf{x}}. \end{aligned}$$

The last line is what we define the Rayleigh Quotient to be,

$$r(\mathbf{x}) = \frac{\mathbf{x}^* A\mathbf{x}}{\mathbf{x}^*\mathbf{x}}. \tag{4.1}$$

However, like mentioned before, the RQ is not only useful for computing the exact eigenvalue when one knows the exact eigenvector. Furthermore, the RQ minimizes the 2-norm of $||A\mathbf{x} -$

$\alpha\mathbf{x}||_2$. To do this, we begin by minimizing $r(x)$, e.g.,

$$\frac{\partial r(x)}{\partial x_j} = \frac{\frac{\partial}{\partial x_j}\left(\mathbf{x}^* A\mathbf{x}\right)}{\mathbf{x}^*\mathbf{x}} - \frac{\left(\mathbf{x}^* A\mathbf{x}\right)\frac{\partial}{\partial x_j}(\mathbf{x}^*\mathbf{x})}{\mathbf{x}^*\mathbf{x}}$$

$$= \frac{2(A\mathbf{x})_j}{\mathbf{x}^*\mathbf{x}} - \frac{\left(\mathbf{x}^* A\mathbf{x}\right)(2x_j)}{(\mathbf{x}^*\mathbf{x})^2}$$

$$= \frac{2}{\mathbf{x}^*\mathbf{x}}\left[(A\mathbf{x})_j - \left(\frac{\mathbf{x}^* A\mathbf{x}}{\mathbf{x}^*\mathbf{x}}\right)x_j\right]$$

$$= \frac{2}{\mathbf{x}^*\mathbf{x}}\Big(A\mathbf{x} - r(\mathbf{x})\mathbf{x}\Big)_j$$

and hence we have that

$$\nabla r(\mathbf{x}) = \frac{2}{\mathbf{x}^*\mathbf{x}}\big[A\mathbf{x} - r(\mathbf{x})\mathbf{x}\big].$$

Therefore $r(x)$ is minimized when $\mathbf{x}$ is an eigenvector and $r(\mathbf{x})$ is its associated eigenvalue. However, if $\mathbf{x}$ is not a true eigenvector of A, but an approximation to one, how accurate will the RQ be? It turns out that if $\mathbf{x}$ is sufficiently close to a true eigenvector, then the RQ is quadratically accurate. We will go ahead and prove this now!

4.1.1 Proof RQ is Quadratically Accurate

Proof. To begin this proof, we will assume that $\{\mathbf{v}_j\}$ are the true eigenvectors and $\mathbf{x}$ is our approximated eigenvector. We can expand $\mathbf{x}$ in terms of the eigenvector basis,

$$\mathbf{x} = \sum_j a_j \mathbf{v}_j.$$

Now we begin. Where we are going, we don't need roads, I mean we will assume that if $\mathbf{x}$ is sufficiently close to the eigenvector $\mathbf{v}_P$, then we have $\left|\frac{a_k}{a_P}\right| < \epsilon$ for all $k \neq P$. Before we use this fact, we must first do some computations. Let's begin with RQ,

$$r(\mathbf{x}) = \frac{\sum_j a_j \mathbf{v}_j^* A \sum_k a_k \mathbf{v}_k}{\sum_j a_j \mathbf{v}_j^* \sum_k a_k \mathbf{v}_k}$$

$$= \frac{\sum_j a_j \mathbf{v}_j^* \sum_k a_k (A\mathbf{v}_k)}{\sum_j \sum_k a_j a_k \mathbf{v}_j^* \mathbf{v}_k}$$

$$= \frac{\sum_j a_j \mathbf{v}_j^* \sum_k a_k (\lambda_k \mathbf{v}_k)}{\sum_j \sum_k a_j a_k \mathbf{v}_j^* \mathbf{v}_k}$$

$$= \frac{\sum_j \sum_k a_j a_k \lambda_k \mathbf{v}_j^* \mathbf{v}_k}{\sum_j \sum_k a_j a_k \mathbf{v}_j^* \mathbf{v}_k}$$

Now we proceed to subtract λ_P from $r(\mathbf{x})$, e.g.,

$$r(\mathbf{x}) - \lambda_P = \frac{\sum_j \sum_k a_j a_k \lambda_k \mathbf{v}_j^* \mathbf{v}_k}{\sum_j \sum_k a_j a_k \mathbf{v}_j^* \mathbf{v}_k} - \lambda_P$$

$$= \frac{\sum_{j \neq P} \sum_{k \neq P} \left(\frac{a_j a_k}{a_P^2} \right) \lambda_k \mathbf{v}_j^* \mathbf{v}_k + \lambda_P}{\sum_{j \neq P} \sum_{k \neq P} \left(\frac{a_j a_k}{a_P^2} \right) \mathbf{v}_j^* \mathbf{v}_k + 1} - \lambda_P$$

Using one of our favorite Taylor Series, $\frac{1}{1+x} = 1 - x + x^2 - x^3 + \ldots$, since $\left| \frac{a_k}{a_P} \right| < \epsilon$, we get

$$= \left(\sum_{j \neq P} \sum_{k \neq P} \left(\frac{a_j a_k}{a_P^2} \right) \lambda_k \mathbf{v}_j^* \mathbf{v}_k + \lambda_P \right) \left[1 - \left(\sum_{j \neq P} \sum_{k \neq P} \left(\frac{a_j a_k}{a_P^2} \right) \mathbf{v}_j^* \mathbf{v}_k \right) + \ldots \right] - \lambda_P$$

$$= \left[\sum_{j \neq P} \sum_{k \neq P} \left(\frac{a_j a_k}{a_P^2} \right) \lambda_k \mathbf{v}_j^* \mathbf{v}_k + \lambda_P - \lambda_P \sum_{j \neq P} \sum_{k \neq P} \left(\frac{a_j a_k}{a_P^2} \right) \mathbf{v}_j^* \mathbf{v}_k + \lambda_P \right] + \textit{higher order terms}$$

$\approx \mathcal{O}(\epsilon^2)$.

□

Hence we find that $r(\mathbf{x}) - \lambda_P = \mathcal{O}(\epsilon^2)$. Welp, now that we know how accurate RQ is, let's dive into some methods that produce approximate eigenvectors so we can put the RQ to use! We will introduce the Power Method, followed closely behind by it's cousin, the Inverse Power Iteration.

4.2 Power Iteration

The Power Iteration, a method appropriately named as we will be hammering an initial guess of an eigenvector by the matrix A, over and over. That is, we will just continually multiply A onto the vector, $\mathbf{x}^{(0)}$. It turns out that by continually multiply A onto $\mathbf{x}^{(0)}$, it will begin to align itself in the direction of the eigenvector associated with the largest eigenvalue in magnitude.

The algorithm can be written simply as,

$$
\begin{aligned}
&\mathbf{x}^{(0)} \text{ s.t. } \left|\left| v^{(0)} \right|\right| = 1 \\
&\text{for } k = 1, 2, \ldots \\
&\quad \mathbf{w} = A\mathbf{x}^{(k-1)} \\
&\quad \mathbf{x}^{(k)} = \frac{\mathbf{w}}{||\mathbf{w}||} \\
&\quad \lambda^{(k)} = \left(\mathbf{x}^{(k)}\right)^* A\mathbf{x}^{(k)}
\end{aligned}
$$

Well, how does this actually work? It turns out the analysis is not terribly complicated. Let's show why this converges to the eigenvector associated with the largest eigenvalue in magnitude. First we expand the initial guess vector in terms of the eigenbasis,

$$\mathbf{x}^{(0)} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_N\mathbf{v}_N.$$

Next after iterating k times, i.e., $k-$multiples of A onto $\mathbf{x}^{(0)}$ and letting c_k be a normalization parameter and assuming that $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_N|$, we get

$$
\begin{aligned}
\mathbf{x}^{(k)} &= c_k A^k \mathbf{x}^{(0)} \\
&= c_k\left(a_1 A^k\mathbf{v}_1 + a_2 A^k\mathbf{v}_2 + \ldots + a_N A^k\mathbf{v}_N\right) \\
&= c_k\left(a_1\lambda_1^k\mathbf{v}_1 + a_2\lambda_2^k\mathbf{v}_2 + \ldots + a_N\lambda_N^k\mathbf{v}_N\right) \\
&= c_k\lambda_1^k\left(a_1\mathbf{v}_1 + a_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\mathbf{v}_2 + \ldots + a_N\left(\frac{\lambda_N}{\lambda_1}\right)^k\mathbf{v}_N\right).
\end{aligned}
$$

As $k \to \infty$, we see this process will converge to $\mathbf{x}_\infty = c_\infty\lambda^\infty a_1\mathbf{v}_1$. If you forgive that abusive notation, it is clear that the Power Method will be able to capture the single eigenvector associated with the largest eigenvalue in magnitude. Unfortunately, this process will not be able

to produce any others for us, but that is where *Inverse Power Iteration* comes in.

Before we get to that algorithm, we will discuss the convergence of the Power Method to the eigenvector as well as the convergence to its associated eigenvalue, using RQ.

4.2.1 Convergence of Power Method to Dominant Eigenvector

Here we will show that the convergence of the Power Method to the dominant eigenvector is

$$\left|\left|\mathbf{x}^{(k)} - (\pm\mathbf{v}_1)\right|\right| = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$

Proof. We will begin with $\mathbf{x}^{(k)} - (\pm\mathbf{v}_1)$ and massage it until we get our result. Let's take a look.

$$\mathbf{x}^{(k)} - (\pm\mathbf{v}_1) = c_k\left(\sum_{j=1}^{N} a_j\lambda_j^k\mathbf{v}_j\right) - (\pm\mathbf{v}_1)$$

$$= c_k\lambda_1^k a_1\mathbf{v}_1 + c_k\sum_{j=2}^{N} a_j\left(\frac{\lambda_j}{\lambda_1}\right)^k\mathbf{v}_j - (\pm\mathbf{v}_1).$$

Letting the normalization factor, be $c_k = \frac{1}{\lambda^k a_1}$, we get

$$\mathbf{x}^{(k)} - (\pm\mathbf{v}_1) = c_k\sum_{j=2}^{N} a_j\left(\frac{\lambda_j}{\lambda_1}\right)^k\mathbf{v}_j,$$

and the largest error term will be from the $j = 2$ term, and hence we obtain the final result,

$$\left|\left|\mathbf{x}^{(k)} - (\pm\mathbf{v}_1)\right|\right| = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$

□

Welp, we got the result, although the convergence is only linear, but hey, it still converges! Not too bad for such a simple algorithm. What does this mean for the convergence its associated eigenvalue?

4.2.2 Convergence of Power Method for Dominant Eigenvalue

We will show that using the Power Method for finding the dominant eigenvalue converges as,

$$\left|\left|r(\mathbf{x}^{(k)} - \lambda_1\right|\right| = \mathcal{O}\left(\epsilon^{2k}\right).$$

Proof. This is a rather simple proof, in which we will just simply use our previous results for the accuracy of the RQ and the convergence of the Power Method to the dominant eigenvector. First from the accuracy of the RQ we get

$$r(\mathbf{x}^{(k)} - r(\mathbf{v}_1) = \mathcal{O}\left(\epsilon^k\right),$$

and hence

$$r(\mathbf{x}^{(k)} - r(\mathbf{v}_1) = \mathcal{O}\left(\left|\left|\mathbf{x}^{(k)} - \mathbf{v}_1\right|\right|^2\right).$$

Now using the convergence of the Power Method to dominant eigenvector, we find that

$$r(\mathbf{x}^{(k)} - r(\mathbf{v}_1) = \mathcal{O}\left(\left|\left|\mathbf{x}^{(k)} - \mathbf{v}_1\right|\right|^2\right) = \mathcal{O}\left(\left(\epsilon^k\right)^2\right) = \mathcal{O}\left(\epsilon^{2k}\right).$$

□

Now that we have beat the Power Method to death, and it can only give us the dominant eigenvector approximation, let's set our sights a bit higher…finding any eigenvector we want, by guessing different eigenvalue approximations. Yup, it's time for the *Inverse Power Iteration*!

4.3 Inverse Power Iteration

The idea with *Inverse Power Iteration*, IPI, is that if we have background knowledge of what the spectrum of A looks like, we will be able to use that information to pull out what the exact eigenvalues, well eigenvectors, are. We will do this by introducing shifts, μ, into the Power Method. However, rather than apply perform the Power Method by continually multiplying A onto an initial guess, we will continually multiply the matrix $(A-\mu I)^{-1}$.

The reason for this comes from relations between this shifted matrix, $(A-\mu I)^{-1}$ and A. We note two things,

1. The eigenvectors of $(A-\mu I)^{-1}$ are the same as A's except they have eigenvalues $\left\{\frac{1}{\lambda_j-\mu}\right\}$, rather than $\{\lambda_j\}$.
2. If μ is close to λ_j, then $\frac{1}{\lambda_j-\mu} >> \frac{1}{\lambda_k-\mu}$ for $k \neq j$.

We can easily see the first of the above statements is true, i.e.,

Proof. We begin with the eigenvalue equation and go from there,

$$A\mathbf{v} = \lambda\mathbf{v}.$$

Now subtract μI from both sides,

$$(A-\mu I)\mathbf{v} = (\lambda-\mu)\mathbf{v}.$$

Finally we just need to multiply both sides by $(A-\mu I)^{-1}$ and divide both sides by $(\lambda-\mu)$, to get

$$(A-\mu I)^{-1}\mathbf{v} = \left(\frac{1}{\lambda-\mu}\right)\mathbf{v}.$$

Hence we see that A and $(A-\mu I)^{-1}$ have the same eigenvectors, and the eigenvalues of $(A-\mu I)^{-1}$ are $\left\{\frac{1}{\lambda_j-\mu}\right\}$. □

Similar to the Power Method, the convergence of the IPI is only linear. Furthermore, the algorithm looks awfully similar to that of the Power Method; however, each step is going to require solving a linear system, now

$$\begin{aligned}
&\mathbf{x}^{(0)} \text{ s.t. } \left|\left|v^{(0)}\right|\right| = 1\\
&\text{for } k = 1,2,\ldots\\
&\quad Solve: \left(A-\mu I\right)\mathbf{w} = \mathbf{x}^{(k-1)}\\
&\quad \mathbf{x}^{(k)} = \frac{\mathbf{w}}{||\mathbf{w}||}\\
&\quad \lambda^{(k)} = \left(\mathbf{x}^{(k)}\right)^* A\mathbf{x}^{(k)}
\end{aligned}$$

One question you may be asking yourself is, if we're computing $\lambda^{(k)}$ each iteration with the RQ, why aren't we using that information anywhere? Also, does the shift, μ, need to stay the same as the computation progresses? The answer is, lets use the extra information to update our shift each iteration! This should cause the algorithm to converge faster, too (and it does!)!

This process is now called the *Rayleigh Quotient Iteration*!

4.3.1 Rayleigh Quotient Iteration

The *Rayleigh Quotient Iteration* is virtually the same as the Inverse Power Iteration, except each iteration with update the shift with $\mu^{(k)} = \lambda^{(k)}$. The algorithm then becomes

$$\begin{aligned}
&\mathbf{x}^{(0)} \text{ s.t. } \left|\left|v^{(0)}\right|\right| = 1\\
&\lambda^{(0)} = \left(\mathbf{v}^{(0)}\right)^* A\mathbf{v}^{(0)}\\
&\text{for } k = 1,2,\ldots\\
&\quad Solve: \left(A-\lambda^{(k-1)} I\right)\mathbf{w} = \mathbf{x}^{(k-1)}\\
&\quad \mathbf{x}^{(k)} = \frac{\mathbf{w}}{||\mathbf{w}||}\\
&\quad \lambda^{(k)} = \left(\mathbf{x}^{(k)}\right)^* A\mathbf{x}^{(k)}
\end{aligned}$$

We get the following convergence properties,

$$\left|\left|\mathbf{x}^{(k+1)} - (\pm\mathbf{v}_P)\right|\right| = \mathcal{O}\left(\left|\left|\mathbf{x}^{(k)} - (\pm\mathbf{v}_P)\right|\right|^3\right)$$

$$\left|\lambda^{(k+1)} - \lambda_P\right| = \mathcal{O}\left(\left|\lambda^{(k)} - \lambda_P\right|^3\right)$$

that is, we have cubic convergence! These proofs are very similar to those from the Power Method. The sketch of the proof is the following. If we assume $\mathbf{x}^{(k)}$ is sufficiently close to the true eigenvector, $\mathbf{v}_P$, then we have

$$\left|\left|\mathbf{x}^{(k)} - (\pm\mathbf{v}_P)\right|\right| \leq \epsilon,$$

and also from RQ that

$$\left|\lambda^{(k)} - \lambda_P\right| = \mathcal{O}\left(\epsilon^2\right).$$

Now if we go ahead and do one more iteration of the Rayleigh Quotient Iteration we get,

$$\left|\left|\mathbf{x}^{(k+1)} - (\pm\mathbf{v}_P)\right|\right| = \mathcal{O}\left(\left|\lambda^{(k)} - \lambda_P\right|\left|\left|\mathbf{x}^{(k)} - (\pm\mathbf{v}_P)\right|\right|\right) = \mathcal{O}\left(\epsilon^3\right).$$

One last thing to note is the computational expense of the Rayleigh Quotient Iteration (or Inverse Power Iteration for that matter). Due to solving a linear system at each iteration, this could become very expensive, rather quickly…making Rayleigh Quotient Iteration definitely the ideal choice over the Inverse Power Iteration, since it exhibits cubic convergence. The expense is as follows,

1. If A is a dense matrix, then solving $\left(A - \lambda^{(k-1)}I\right)\mathbf{w} = \mathbf{x}^{(k-1)}$ will most likely take $\mathcal{O}\left(N^3\right)$ operations.
2. If A is upper-Hessenberg, then each step would take $\mathcal{O}\left(N^2\right)$ operations.
3. If A is tridiagonal, then each step takes $\mathcal{O}(N)$ operations.

Now that we've discussed how to find one eigenvector and eigenvalue at a time, is it possible to find all the eigenvalues at once? Or at least a method so there is as much guess and checking of shift values, μ, to capture the whole spectrum? The answer is…yes! And the algorithm goes by the name of the QR algorithm.

5 The QR Algorithm

Here you will finally be introduced to one of top ten algorithms developed in the 20^{th} century - the QR algorithm. It was the most popular method for finding all the eigenvalues of a matrix. Although there are a few variants of the QR algorithm, the heart of the algorithm can easily be portrayed in a couple words. That is, find the QR factorization of a matrix, multiply the factors in reverse order, and continue the iteration. We will begin our discussion by the analyzing the *pure* QR algorithm.

5.0.2 THE "PURE" QR ALGORITHM

The "pure" in this distinction refers to the algorithm not using any additional information to formulate it, i.e., shifts of any kind, etc. Again, at the heart of this algorithm is a series of continually finding the QR factorization of a matrix and then multiplying the factors in reverse order. Let's go ahead and list the algorithm now.

"Pure" QR Algorithm

Initialize $A^{(0)} = A$
for $k = 1, 2, \ldots$
  Compute QR factorization of $A^{(k-1)}$, i.e., $Q^{(k)}R^{(k)} = A^{(k-1)}$
  Recombine in reverse order $A^{(k)} = R^{(k)}Q^{(k)}$
end

There are a few key ideas behind this algorithm that may offer some intuition to how it works. Let's explore those.

Idea 1 : Each step triangularizes $A^{(k)}$. This is seen as from the above algorithm it is clear that

$$R^{(k)} = \left(Q^{(k)}\right)^* A^{(k-1)}.$$

- Idea 2: Once we have triangularized the previous iterate, we can multiply on the right by $\left(Q^{(k)}\right)$ to get

  $$R^{(k)}Q^{(k)} = \left(Q^{(k)}\right)^* A^{(k-1)}Q^{(k)},$$

  and hence we have that

  $$A^{(k)} = \left(Q^{(k)}\right)^* A^{(k-1)}Q^{(k)}.$$

  We therefore see that $A^{(k)}$ and $A^{(k-1)}$ are constructed to be similar matrices at each step, preserving the spectrum of the matrix A through the computation. Also, the diagonal elements of $A^{(k)}$ are Rayleigh Quotients (in the case of $A = A^T$) or at least eigenvalue approximations of a general matrix. Moreover, this algorithm is backward stable because of the orthogonal similarity transformations used in each step.

We note that the algorithm will not produce an upper triangular matrix (or if A is symmetric, even a diagonal matrix) in one step. However, as a basis of iteration it will eventually converge to what we'd expect - the Schur form of the matrix A. If we eventually arrive at the Schur form, then it is obvious that the eigenvalues lie along the main dragonal of that upper triangular matrix.

Furthermore, if $A = A^T$, we know that all the eigenvalues are real and all the eigenvectors of A are orthogonal (or orthonormal, upon normalization), and hence upon convergence of the QR factorization to the Schur form, we will have all the eigenvalues, along the main diagonal

of diagonal matrix, but mores we will also have all the eigenvectors in Q.

This algorithm, although simple to wrote out and say, can be considerably expensive, as stated previously when giving an overview of eigenvalue solvers. Every step requires the QR factorization of a matrix, which typically costs $\mathcal{O}\left(N^3\right)$ operations. However, with a few modifications, the algorithm can experience drastic speed up and lower computational cost. These 3 modifications include,

1. Before starting the QR algorithm, massage the matrix to upper-Hessenberg for (or tridiagonal form if A is symmetric).

2. Use a shifted matrix at each step, $A^{(k)} - \mu^{(k)} I$, where $\mu^{(k)}$ is a shift, as in the Inverse Power Iteration.

3. Whenever it is possible, i.e., when an eigenvalue is "found". *deflate* the matrix by breaking $A^{(k)}$ into submatrices.

We also note that one can view the QR algorithm as a beefed up version of the Power Method, applied to a basis of vectors all at once for the case of real, symmetric matrices, except we will use the QR factorization to renormalize and orthogonalized at each step. This is called *simultaneous iteration.*
The algorithm for simultaneous iteration for the case of $A = A^T$, is

Simultaneous Iteration

$Q^{(0)} \in \mathbb{R}^{N\times N}$ i.e., the identity is usually acceptable
for $k = 1, 2, \ldots$
  Multiply by A, $Z = A^{(k-1)}$
  Find QR Fact. of $ZQ^{(k)}R^{(k)} = Z$
end

As $k \to \infty$, $Q^{(k)}$ converges to a matrix of the eigenvectors of real, symmetric matrix A. However, the convergence will be linear.

Theorem 3. *The "pure" QR algorithm applied the a real, symmetric matrix A, where we can order the spectrum $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_N|$., and whose eigenvectors matrix is full rank, then as $k \to \infty$, $A^{(k)}$ converges* linearly *to the diagonal matrix* $\text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$ *and $Q^{(k)}$ converges linearly, as well.*

Okay, so the convergence rate really isn't anything to write home about; however, it still converges! We will see that by adding shifts into the QR algorithm, the algorithm will be able to achieve cubic convergence. We might as well peer into that algorithm, you know, for completeness, but mostly fun.

5.0.3 *QR* Algorithm with Shifts

As before, the shifts we choose are eigenvalue approximations at each step. To be clear, we are introducing the shifts into the QR algorithm simply to speed up the convergence rate. We will discuss a couple popular options for choosing shifts soon, but first let's become acquainted with the algorithm at hand.

The standard QR algorithm with shifts is as follows,

Standard QR Algorithm With Shifts

$A^{(0)} = A$
for $k = 1, 2, \ldots$
  pick $\mu^{(k)}$
  perform QR fact. of shifted matrix $Q^{(k)}R^{(k)} = A^{(k-1)} - \mu^{(k)}I$
  recombine $A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}I$.

Again, it is not difficult to prove that A and $A^{(k)}$ are similar matrices. We begin with the last iterate, $A^{(k)}$,

Proof.

$$A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}I = \left(Q^{(k)}\right)^* \left(A^{(k)} - \mu^{(k)}I\right)Q^{(k)} + +\mu^{(k)}I = \left(Q^{(k)}\right)^* A^{(k-1)}Q^{(k)}.$$

Now simply repeated this process $k-1$ more times, it's fairly trivial to see that $A^{(k)}$ and A are similar. □

The clear question - how does one choose the shifts, $\mu^{(k)}$? There are a bunch of ways. The first obvious one is choosing them to be the Rayleigh Quotients, i.e.,

$$\text{If using Rayleigh Quotients: } \mu^{(k)} = A^{(k)}_{NN},$$

that is, the last diagonal element of $A^{(k)}$. Another other popular choice of a shift is called the *Wilkinson Shift*. This shift was created because in cases where there are symmetric eigenvalues, using the Rayleigh Quotients as shifts may fail. This shift seeks to remedy that problem. The shift is defined as

$$\text{If using Wilkinson Shifts: } \mu = -\operatorname{sign}(\delta)\frac{b_{m-1}^2}{|\delta| + \sqrt{\delta^2 + b_{m-1}^2}},$$

where $\delta = \frac{a_{m-1} - a_m}{2}$ and $B = \begin{bmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{bmatrix}$ is the lower-right sub matrix of $A^{(k)}$.

Like all things in math, there is also a mixed-type shift, that is

$$\text{If using Mixed-type Shifts: } \mu^{(k)} = \begin{cases} \text{RQ-Shift} & \text{if } \xi b_{n-2} \geq b_{n-1} \\ \text{Wilkinson Shift} & \text{if } \xi b_{n-2} < b_{n-1} \end{cases},$$

where ξ is some parameter that has yet to have been determined. However, regardless of choice of shift, there is a slight variant on the QR algorithm with shifts, which includes one addition step to accelerate convergence. Basically whenever an eigenvalue is "found", you will deflate the matrix by breaking it into submatrices. It has been called the "Practice" QR Algorithm With Shifts. Take a glance.

Practical QR Algorithm With Shifts

$A^{(0)} = A$
for $k = 1, 2, \ldots$
  pick $\mu^{(k)}$
  perform QR fact. of shifted matrix $Q^{(k)} R^{(k)} = A^{(k-1)} - \mu^{(k)} I$
  recombine $A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I$.
  If $\left| A^{(k)}_{j,j+1} \right| < \epsilon$, set $A^{(k)}_{j,j+1} = A^{(k)}_{j+1,k} = 0$
    This creates two submatrices, $A^{(k)}_1$ and $A^{(k)}_2$, *i.e.*,

$$A^{(k)} = \begin{bmatrix} A^{(k)}_1 & 0 \\ 0 & A^{(k)}_2 \end{bmatrix}$$

The last thing we have to do to finish our discussion of eigenvalues, was turn our sights back onto the first step of the two for numerical eigenvalue schemes. That is, we need to say something about how to massage matrices into upper-Hessenberg form when they are impressively large and dense. These methods are called the *Arnoldi Iteration* and *Lanczos Iteration*.

6 Introducing Mr. Arnoldi and Mr. Lanczos; some Krylov friends

If A is an exceptionally large matrix, it may not be computationally feasible or efficient to completely transform it into its upper-Hessenberg form. This where the *Krylov Subspace* methods come into play. What will happen is that rather than going after the complete transformation, we will make our goal to find partial results after a relatively small number of iterations. If that sounds rather cryptic, just picture what the problem at hand is too difficult so what we will try to do is find our solutions in a smaller subspace of the original problem.

In the first step of an eigenvalue solver, transforming the matrix to upper-Hessenberg form, there are two algorithms that will do just that - the *Arnoldi Iteration* and the *Lanczos Iteration*. Note that the Lanczos algorithm is simply an efficient version of Arnoldi applied to symmetric matrices. Furthermore, if $A = A^T$, we obtain a tri-diagonal matrix with either Arnoldi or Lanczos, but Lanczos will be more efficient.

We will motivate the main idea using the Arnoldi Iteration and then look into where Lanzcos fits into the picture.

6.1 The Arnoldi Iteration

If you recall the Power Method, it produced a sequence of vectors $\{v, Av, A^2v, A^3v, \ldots\}$. The idea was that eventually this sequence would least to the eigenvector associated with the dominant eigenvalue, say λ_1. A lot of effort was wasted in performing this iteration to only use the last vector found, $A^n v$. Rather, let's save all of those vectors, and form a matrix from them,

$$K_n = \left[v \, Av \, A^2 v \, A^3 v \ldots A^n v \right],$$

which is called the *Krylov Matrix*. In general the columns of this matrix will not be orthogonal; however, in principle we can extract an orthogonal basis from it, which spans the same space. This orthogonalized subspace is what is referred to as the *Krylov Subspace*, $\mathcal{K}_n$. These vectors could be easily orthogonalized using a standard Gram-Schmidt orthogonalization process, too!

Forming these subspaces, we expect could that the vectors of $\mathcal{K}_n$ give good approximations of the eigenvectors corresponding to the $n+1$ largest eigenvalues. It is then makes sense why we would expect $A^n v$ to give a good approximation to the dominant eigenvector.

Before diving right into the algorithm, let's try to develop a little intuition and motivation behind this madness. The main thing that we begin with are orthogonal similarity transformations,

$$A = QHQ^*,$$

and hence

$$AQ = HQ.$$

However, like we mentioned before, it may be in our best interest not to form H explicitly since it would be too costly. Hence we will try to build up successive subspaces to approximate Q and H. Let's define those!

- We define $Q_m \in \mathbb{C}^{n \times m}$ to be a matrix whose columns are the first m-columns of Q.

- We definte $\tilde{H}_n$ to be the $(m+1)\times m$ upper left section of the matrix H. Hence $\tilde{H}_m$ will be have the following form

$$\tilde{H}^m = \begin{bmatrix} h_{11} & h_{12} & \cdots & & h_{1m} \\ h_{21} & h_{22} & \cdots & & \vdots \\ & & \ddots & & \vdots \\ & & & h_{m,m+1} & h_{mm} \\ & & & & h_{m,m+1} \end{bmatrix}$$

The main idea is rather than consider $AQ = QH$, we consider

$$AQ_m = Q_{m+1}\tilde{H}_m.$$

Upon doing this we will be able to build up the Krylov subspace using basic Gram-Schmidt Orthogonalization. Therefore one can find the recurrence relation in the $(m+1)^{st}$ column of $\tilde{H}_m$ to be

$$A\mathbf{q}_n = h_{1m}\mathbf{q}_1 + h_{2m}\mathbf{q}_2 + \ldots + h_{mm}\mathbf{q}_m + h_{m+1,m}\mathbf{q}_{m+1}.$$

We can then solve this recurrence relation for $\mathbf{q}_{m+1}$ using the previous Krylov vectors! All of the coefficients. $\{h_{jk}\}$ are the standard inner products from a Gram-Schmidt process. This is essentially the way we will be able to build up $\mathscr{K}_n$. We are now in a position to check out the algorithm itself!

Arnoldi Iteration

$\mathbf{b}$: arbitrary, $\mathbf{q}_1 = \dfrac{\mathbf{b}}{||\mathbf{b}||}$

for $k = 1, 2, \ldots$

$\mathbf{v} = A\mathbf{q}_n$

for $j = 1, 2, \ldots$

$h_{jn} = \mathbf{q}_j^* \mathbf{v}$

$\mathbf{v} = \mathbf{v} - h_{jn}\mathbf{q}_j$

$h_{n,n+1} = ||\mathbf{v}||$

$\mathbf{q}_{n+1} = \dfrac{\mathbf{v}}{h_{n,n+1}}$

At the heart of this iteration, we are finding a basis of successive *Krylov Subspaces* using Gram-Schmidt Orthogonalization. Hence we are creating

$$\mathscr{K}_n = \langle \mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, A^3\mathbf{b}, \ldots, A^{n-1}\mathbf{b} \rangle = \langle \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \ldots, \mathbf{q}_{n-1} \rangle.$$

We have the following theorem relating the QR factorization and the Arnoldi Iteration.

Theorem 4. *The matrices Q_n generated by the Arnoldi Iteration are reduced QR factors of the Krylov Matrix, $K_n = Q_n R_n$. Moreover, the Hessenberg matrices, $\tilde{H}_n$, are the corresponding projections $\tilde{H}_n = Q_n^* A Q^n$, and successive iterations related by $AQ_n = Q_{n+1}\tilde{H}_n$.*

We note that if either K_n, or R_n for that matter, are found explicitly, the overall scheme will be unstable, as the columns of K_n would approximate the same dominant eigenvector of A. This is why we need to orthogonalize the subspace at each iteration. Moreover, the QR factorization is expected to reveal information about the largest eigenvalues by peeling off one approximated eigenvector after another, starting with the dominant one.

On that note, since $\tilde{H}_n$ is a projection of A, we have that the eigenvalues of $\tilde{H}_n$ are called the *Ritz Values*, e.g., Arnoldi eigenvalue estimates, at step n. This can be seen easily if at some point at entry, $h_{n+1,n} = 0$ is encountered. Let's show that this is true!

Proof. To show that each eigenvalue of H_n is an eigenvalue of A in the case an entry $h_{n+1,n} = 0$ is encounter during the algorithm, we start with

$$AQ_n = Q_n \tilde{H}_n.$$

Letting $\mathbf{v}$ be an eigenvector of $\tilde{H}_n$ and λ its associated eigenvalue, we have

$$AQ_n\mathbf{v} = Q_n \tilde{H}_n\mathbf{v} = Q_n(\lambda\mathbf{v}) = \lambda Q_n\mathbf{v}.$$

Now substituting $\mathbf{y} = Q_n\mathbf{v}$, we have

$$A\mathbf{y} = \lambda\mathbf{y}.$$

□

Furthermore, out of the realm of eigenvalues, we can show that if A is non-singular, then the solution $\mathbf{x}$ to $A\mathbf{x} = \mathbf{b}$ lies in $\mathcal{K}_n$! This point becomes extremely relevant when we use the Krylov Subspaces for large linear systems, i.e., *GMRES*. Let's prove this!

Proof. To show that if A is non-singular, then the solution $\mathbf{x}$ to $A\mathbf{x} = \mathbf{b}$ lies in $\mathcal{K}_n$, we begin with

$$\mathbf{x} = A^{-1}\mathbf{b}.$$

Now note that we have $\mathbf{b} = Q_n\mathbf{q}_1||\mathbf{b}||$, by definition of how we started the Arnoldi iteration. Therefore we have

$$\mathbf{x} = A^{-1}\mathbf{b} = A^{-1}Q_n\mathbf{q}_1||\mathbf{b}|| = A^{-1}Q_n\left(\tilde{H}_n^{-1}\tilde{H}_n\right)\mathbf{q}_1||\mathbf{b}||.$$

Substituting $AQ_n = Q_n\tilde{H}_n$,we have that

$$\mathbf{x} = A^{-1}(AQ_n)\,\tilde{H}_n^{-1}\mathbf{q}_1||\mathbf{b}|| = Q_n\tilde{H}_n^{-1}\mathbf{q}_1||\mathbf{b}||.$$

Hence we have

$$\mathbf{x} = Q_n\left(\tilde{H}_n^{-1}\mathbf{q}_1||\mathbf{b}||\right) \in \mathcal{K}_n.$$

□

We will now turn our sights to the Arnoldi Iteration's kid brother, the Lanczos algorithm. This algorithm is an efficient implementation of Arnoldi for real, symmetric matrices, A.

6.2 LANCZOS ITERATION

As said before, the Lanczos Iteration does the same thing of trying to reduce a large matrix A to upper-Hessenberg form, but only if A is real and symmetric. Furthermore, rather than obtaining an upper-Hessenberg matrix, because of the symmetry of A, we get a tri-diagonal matrix. However, if you remember one thing, it is just that Lanczos is the Arnoldi Iteration for real, symmetric matrices.

The reason we obtain a tri-diagonal matrix can be seen directly from the matrix decomposition. Recall we have $A = Q_n \tilde{H}_N Q_n^T$. Now if $A = A^T$, we have

$$A^T = \left(Q_n \tilde{H}_N Q_n^T\right)^T = Q_n \tilde{H}_n^T Q_n^T,$$

and hence it in necessary that $\tilde{H}_n = \tilde{H}_n^T$, to which we can induce $\tilde{H}_n$ is both symmetric and tri-diagonal.

Let's take a look at the algorithm!

Lanczos Iteration

$\beta_0 = 0,\ \mathbf{q}_0 = \mathbf{0}$

$\mathbf{b}$: arbitrary, $\mathbf{q}_1 = \dfrac{\mathbf{b}}{||\mathbf{b}||}$

for $k = 1, 2, \ldots$

$\mathbf{v} = A\mathbf{q}_n$

$\alpha_n = \mathbf{q}_n^T \mathbf{v}$

$\mathbf{v} = \mathbf{v} - \beta_{n-1}\mathbf{q}_{n-1} - \alpha_n \mathbf{q}_n$

$\beta_n = ||\mathbf{v}||$

$\mathbf{q}_{n+1} = \dfrac{\mathbf{v}}{\beta_n}$