Linear Algebra III: Computations in Coordinates

Nathan Ratliff

Nov 1, 2014

Abstract

This document completes the step from abstract vector spaces to coordinates and introduces some computational masterpieces of matrix algebra. It starts with an exploration of the Singular Value Decomposition (SVD), and shows how many fundamental properties of the matrix manifest directly in the structure exposed by the computation. It then progresses to orthogonalization and the QR-decomposition, emphasizing how uniqueness of the decomposition enables us to approach the problem from vastly different directions while still ending up at the same result. And finally, it reviews Eigenvectors and Eigenvalues, and introduces the Power Method through one of its most famous applications, Google's PageRank algorithm. That discussion culminates in the presentation and analysis of the QR-algorithm for estimating the full Eigenspectrum of a matrix.

1 Introduction

We've studied the underlying structure of linear transform in abstract vector spaces. That setting is important in understanding the fundamental basis-independent behavior and geometry of a linear transform, but at the end of the day we often need to choose coordinates in the domain and co-domain and to computationally manipulate the resulting matrix of the linear transform. These notes explore how the coordinate representation of the linear transform's fundamental structure, which manifests as a matrix decomposition we know as the Singular Value Decomposition (SVD), exposes the structure of any matrix in a way that allows one to simply read off many of the underlying properties of the operator directly from the elements of the decomposition.

The SVD is a very powerful tool for understanding and manipulating the matrix, but it's relatively expensive to compute. We additionally explore orthogonalization techniques that simplify the system by representing the matrix as a product of an orthogonal matrix (forming an orthonormal basis for the column space) and an upper triangular matrix (which is just really easy to solve). There are a number of very different algorithms for computing QR-decompositions, all of which are much faster in practice than exposing the full

underlying structure of a matrix through an SVD computation, but we show also that the QR-decomposition is unique as long as you follow a simple sign convention.

This uniqueness demonstrates one of the fascinating aspects of matrix manipulation. No matter how different a QR-decomposition algorithm seems, since the QR-decomposition is unique, in the end you always end up with the same decomposition. We overview two methods for computing the decomposition: the more intuitive Gram-Schmidt algorithm for orthogonalization, and a method called the Householder-QR algorithm which uses Householder reflections to more directly construction the upper triangular factor column by column through the application of a series of orthogonal transforms. Both of these algorithms produce the same result, although their numerical properties and practical complexity may differ.

Finally, we then explore Eigenvalues and Eigenvectors in more depth, focusing on their computation through the strikingly simple but remarkably powerful Power Method and QR-algorithm. As an example, we study the Page Rank algorithm, both in terms of its intuitive construction and it's analysis as a Power Method.

2 SVDs and fundamental spaces

Earlier, between lectures 1 and 2 (Ratliff, 2014a,b), we saw that the fundamental spaces for a linear transform and the corresponding mapping between them are represented clearly in its underlying structural representation $T = \sum_i \sigma_i |u_i\rangle\langle v_i|$. We then saw that this representation, in coordinates, becomes the familiar Singular Value Decomposition (SVD) of a matrix. Every time we compute the SVD of a matrix, we're uncovering the fundamental underlying structure of a linear transform as represented in the chosen coordinate system. This section demonstrates that many of the geometric properties of the matrix (and corresponding underlying transform) can simply be read off directly from the SVD.

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ of rank k be an arbitrary matrix, and let $\boldsymbol{A} = \boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T$ be its "thin" SVD, where $\boldsymbol{U}_{/\!/} \in \mathbb{R}^{m\times k}$ is the matrix of k mutually orthogonal left singular vectors, $\boldsymbol{S} \in \mathbb{R}^{k^2}$ is the diagonal matrix with diagonal entries containing the singular values σ_i, and $\boldsymbol{V}_{/\!/} \in \mathbb{R}^{n\times k}$ is the matrix of k mutually orthogonal right singular vectors. We can expand this representation by adding $m-k$ more left singular vectors in a matrix $\boldsymbol{U}_\perp \in \mathbb{R}^{m-k}$, $n-k$ more right singular vectors in a matrix $\boldsymbol{V}_\perp \in \mathbb{R}^{n-k}$, and some zero padding around $\boldsymbol{S}$ to match the resulting dimensions of the surrounding matrices:

$$\boldsymbol{A} = \underbrace{\begin{pmatrix} \boldsymbol{U}_{/\!/} & \boldsymbol{U}_\perp \end{pmatrix}}_{\boldsymbol{U}\in\mathbb{R}^{m^2}} \underbrace{\begin{pmatrix} \boldsymbol{S} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}}_{\widetilde{\boldsymbol{S}}\in\mathbb{R}^{m\times n}} \underbrace{\begin{pmatrix} \boldsymbol{V}_{/\!/}^T \\ \boldsymbol{V}_\perp^T \end{pmatrix}}_{\boldsymbol{V}^T\in\mathbb{R}^{n^2}}. \tag{1}$$

From this "full" SVD, $\boldsymbol{A} = \boldsymbol{U}\widetilde{\boldsymbol{S}}\boldsymbol{V}^T$ with both $\boldsymbol{U}$ and $\boldsymbol{V}$ now as fully orthogonal matrices, we can read off a lot of properties of the matrix. We give

a sketch of the rational behind these properties here and leave the rest to the reader to mull over:

1. **Column space.** The column space is spanned by the orthonormal vectors in $\boldsymbol{U}_{/\!/}$. This can be easily seen because if $\boldsymbol{v}$ lies in the column space, then there is some vector $\boldsymbol{x}$ such that $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{x}$. Using the thin SVD, we have $\boldsymbol{v} = \boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T\boldsymbol{x} = \boldsymbol{U}_{/\!/}\boldsymbol{\beta}$ where $\boldsymbol{\beta} = \boldsymbol{S}\boldsymbol{V}_{/\!/}^T\boldsymbol{x}$. The reverse direction is easy to show as well. This space is the coordinate representation of what we referred to earlier as the left fundamental space of underlying linear map (Ratliff, 2014b).

2. **Row space.** The row space is spanned by the orthonormal vectors in $\boldsymbol{V}_{/\!/}$. This property can be shown using essentially the same argument as we used for the column space. This space is the coordinate representation of what we referred to earlier as the right fundamental space of underlying linear map (Ratliff, 2014b).

3. **Left null space.** $\boldsymbol{U}_\perp$ spans the left null space. This property we can see by examining the operation of matrix multiplication. If $\boldsymbol{x}^T\boldsymbol{A} = \boldsymbol{x}^T\boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T = \boldsymbol{0}^T$, it must be that $\boldsymbol{x}^T\boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T(\boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}) = \boldsymbol{x}^T\boldsymbol{U}_{/\!/} = \boldsymbol{0}^T$. This means $\boldsymbol{x}$ is in the orthogonal compliment to the space spanned by $\boldsymbol{U}_{/\!/}$, which is to say $\boldsymbol{x} \in \mathbf{span}(\boldsymbol{U}_\perp)$.

4. **Right null space.** $\boldsymbol{V}_\perp$ spans the right null space. We can show this using a similar argument to the one we use for the left null space.

Moreover, we can easily see that the linear map represented by the matrix is bijective between the row space and the column space. Any point already in the row space is of the form $\boldsymbol{x} = \boldsymbol{V}_{/\!/}\boldsymbol{\alpha}$ for some vector of coefficients $\boldsymbol{\alpha} \in \mathbb{R}^k$. Pushing that vector through the matrix gives $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T\boldsymbol{V}_{/\!/}\boldsymbol{\alpha} = \boldsymbol{U}_{/\!/}(\boldsymbol{S}\boldsymbol{\alpha}) = \boldsymbol{U}_{/\!/}\boldsymbol{\beta}$ where $\boldsymbol{\beta} = \boldsymbol{S}\boldsymbol{\alpha}$. This vector $\boldsymbol{U}_{/\!/}\boldsymbol{\beta}$ is, by definition, an element of the column space. Now, to get back to where it came from, we just need to get rid of $\boldsymbol{U}_{/\!/}$ and $\boldsymbol{S}$ and then re-apply the resulting coefficients to the basis $\boldsymbol{V}_{/\!/}$. In full, those operations imply

$$\underbrace{\left(\boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}_{/\!/}^T\right)}_{\boldsymbol{A}^\dagger}\underbrace{\boldsymbol{U}_{/\!/}\overbrace{(\boldsymbol{S}\boldsymbol{\alpha})}^{\boldsymbol{\beta}}}_{\boldsymbol{A}\boldsymbol{x}} = \boldsymbol{V}_{/\!/}\boldsymbol{\alpha} = \boldsymbol{x}. \tag{2}$$

The indicated matrix $\boldsymbol{A}^\dagger = \boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}_{/\!/}^T$ is known as the pseudoinverse of $\boldsymbol{A}$. The pseudoinverse, which we understand geometrically through the SVD, forms the inverse map that undoes the operation of $\boldsymbol{A}$ between fundamental spaces (the column space and the row space). The existence of this pseudoinverse, which is actually the inverse of $\boldsymbol{A}$ when we restrict the domain and co-domain to the row space and column space, respectively, shows that $\boldsymbol{A}$ does indeed form a bijection between those two fundamental subspaces.

Now what happens off of those spaces. Since the matrix is only of rank k, it simply doesn't have the capacity to represent components of vectors lying in the

null spaces. For an arbitrary $\boldsymbol{x}$ in the domain, the forward mapping $\boldsymbol{A}$ operates by throwing away any component orthogonal to $\boldsymbol{U}_{/\!/}$, effectively projecting the point onto $\boldsymbol{U}_{/\!/}$, before mapping it. Explicitly, we can decompose the point as $\boldsymbol{x} = \boldsymbol{V}_{/\!/}\boldsymbol{\alpha} + \boldsymbol{V}_{\perp}\boldsymbol{\beta}$ for some $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Applying $\boldsymbol{A}$ has the effect

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}\left(\boldsymbol{V}_{/\!/}\boldsymbol{\alpha} + \boldsymbol{V}_{\perp}\boldsymbol{\beta}\right) = \boldsymbol{A}(\boldsymbol{V}_{/\!/}\boldsymbol{\alpha}) + \boldsymbol{U}_{/\!/}\boldsymbol{S}\underbrace{\boldsymbol{V}_{/\!/}^T\boldsymbol{V}_{\perp}}_{=\boldsymbol{0}}\boldsymbol{\beta} = \boldsymbol{A}(\boldsymbol{V}_{/\!/}\boldsymbol{\alpha}). \tag{3}$$

And similarly in the reverse direction, the pseudoinverse $\boldsymbol{A}^{\dagger} = \boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}_{/\!/}^T$ can't represent any component of a vector lying in the left null space, so it just throws it away, effectively projecting it onto the column space.

Thus, symmetrically, the forward map throws away right null space components and then transforms the resulting row space vector through the bijection between fundamental spaces. And the reverse map (the pseudoinverse) throws away any left null space components and then transforms the resulting column space vector through the inverse map of the bijection between fundamental spaces.

Of note, this pseudoinverse solves the **least squares** problem. If we have a vector $\boldsymbol{b}$ that doesn't lie in the column space of a matrix, then the best we can do is find the $\boldsymbol{x}$ that's closest to the desired $\boldsymbol{b}$:

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|^2. \tag{4}$$

Since every vector of the form $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{x}$ lies in the column space of $\boldsymbol{A}$, we can re-read this problem statement as saying that we want to find the point in the column space that's closest to the point $\boldsymbol{b}$, and then ultimately find the $\boldsymbol{x}$ that constructs that column space point. Said another way, we want to project $\boldsymbol{b}$ onto the column space, and then transform the resulting column space vector through the inverse map of the bijection between fundamental spaces. This is precisely the operation performed by the pseudoinverse $\boldsymbol{A}^{\dagger} = \boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}_{/\!/}^T$ as described above.

Algebraically, we can work out this result by brute force minimization of the least squares problem:

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|^2 = \frac{1}{2}\boldsymbol{x}^T\left(\boldsymbol{A}^T\boldsymbol{A}\right)\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{A}\boldsymbol{x} + \frac{1}{2}\boldsymbol{b}^T\boldsymbol{b}. \tag{5}$$

Setting the gradient of $f(\boldsymbol{x})$ to zero gives

$$\nabla f(\boldsymbol{x}) = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^T\boldsymbol{b} = 0. \tag{6}$$

Now, expanding $\boldsymbol{A} = \boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T$ we get

$$\begin{aligned}
&\left(\boldsymbol{V}_{/\!/}\boldsymbol{S}\boldsymbol{U}_{/\!/}^T\right)\left(\boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T\right)\boldsymbol{x} - \boldsymbol{V}_{/\!/}\boldsymbol{S}\boldsymbol{U}_{/\!/}^T\boldsymbol{b} = \boldsymbol{V}_{/\!/}\boldsymbol{S}^2\boldsymbol{V}_{/\!/}^T\boldsymbol{x} - \boldsymbol{V}_{/\!/}\boldsymbol{S}\boldsymbol{U}_{/\!/}^T\boldsymbol{b} = 0 \\
&\Rightarrow \boldsymbol{V}_{/\!/}\boldsymbol{S}^2\boldsymbol{V}_{/\!/}^T\boldsymbol{x} - \boldsymbol{V}_{/\!/}\boldsymbol{S}\boldsymbol{U}_{/\!/}^T\boldsymbol{b} = 0 \\
&\Rightarrow \boldsymbol{x} = \underbrace{\boldsymbol{V}_{/\!/}\boldsymbol{S}^{-1}\boldsymbol{U}_{/\!/}^T}_{\boldsymbol{A}^{\dagger}}\boldsymbol{b} + \boldsymbol{V}_{\perp}\boldsymbol{\beta},
\end{aligned} \tag{7}$$

for any $\boldsymbol{\beta}$. The final solution here augments the particular pseudoinverse solution to be the entire null space offset by that particular pseudoinverse solution. Note that when $\boldsymbol{A}$ has full column rank, the matrix $\boldsymbol{V}_{/\!/}$ must be a full square matrix $\boldsymbol{V}_{/\!/} \in \mathbb{R}^{n^2}$ since the dimensionalities of the row space and column space must be the same. That leaves no room for extra columns of a Null space, so there simply isn't one. The above expression still holds, but there's only a single unique solution to the least squares problem since the Null space is empty.

3 Orthogonalization: Two flavors of QR-decomposition

Our analysis of least squares problems above demonstrated how least squares problems are easier to solve in an orthonormal basis. But do we need to go all out and fully compute the SVD of the matrix, which can be expensive? Triangular systems are also easy to solve, so what if we were to just reduce our matrix to a product of matrix $\boldsymbol{Q} \in \mathbb{R}^{m \times n}$ with mutually orthogonal columns and an upper triangular matrix $\boldsymbol{R} \in \mathbb{R}^{n^2}$ so that $\boldsymbol{A} = \boldsymbol{QR}$. For over-constrained least squares problems (those with a single unique solution), we need to solve $\boldsymbol{A}^T\boldsymbol{Ax} = \boldsymbol{A}^T\boldsymbol{b}$ as indicated by Equation 6. Using this decomposition, we get

$$\begin{aligned} &(\boldsymbol{QR})^T(\boldsymbol{QR})\boldsymbol{x} = \boldsymbol{R}^T(\boldsymbol{Q}^T\boldsymbol{Q})\boldsymbol{Rx} = \boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{b} \\ &\Rightarrow \boldsymbol{Rx} = \boldsymbol{Q}^T\boldsymbol{b} \\ &\text{or} \quad \boldsymbol{x} = \boldsymbol{R}^{-1}\boldsymbol{Q}^T\boldsymbol{b}. \end{aligned} \tag{8}$$

This last expression, which states the solution, is relatively easy to compute. We just project onto a set of orthogonal columns and then solve a triangular system. So lets take a look at what it requires to compute such a convenient decomposition.

First, lets ask what it means to be a QR-decomposition. Suggesting that our matrix $\boldsymbol{A}$ can be constructed by the product of an orthogonal and an upper triangular matrix $\boldsymbol{A} = \boldsymbol{QR}$ means that there needs to be an orthogonal basis $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n \in \mathbb{R}^m$ for the column space with the particular property that each column $\boldsymbol{a}_k$ of $\boldsymbol{A}$ can be constructed from just the first k of the orthogonal basis vectors. We can see this by simply multiplying out the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$:

$$\begin{pmatrix} | & & | \\ \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_n \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_n \\ | & & | \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} | & | & & | & \\ r_{11}\boldsymbol{q}_1 & r_{12}\boldsymbol{q}_1 + r_{22}\boldsymbol{q}_2 & \cdots & \sum_{i=1}^{k} r_{ik}\boldsymbol{q}_i & \cdots \\ | & | & & | & \end{pmatrix}$$

That simple calculation shows that a decomposition consisting of a product of an orthogonal matrix and an upper triangular matrix necessarily has the property

$$\boldsymbol{a}_k = \sum_{i=1}^{k} r_{ik}\boldsymbol{q}_i \quad \in \ \mathbf{span}(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k).$$

We can conclude two properties from this observation. First, the span of the first k orthonormal vectors (i.e. the first k columns of $\boldsymbol{Q}$) must match the span of the first k columns of $\boldsymbol{A}$:

$$\mathbf{span}(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k) \quad = \quad \mathbf{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k),$$

And second, the specific values of r_{ij} are uniquely determined by

$$r_{ij} = \begin{cases} \boldsymbol{a}_i^T \boldsymbol{q}_j & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases}, \tag{9}$$

because the coefficients $r_{1k}, \ldots, r_{kk}$ need to form the expansion of $\boldsymbol{a}_k$ in terms of the first k basis elements $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k$.

Moreover, the simple stipulation that the span of the first k orthonormal vectors match the span of the first k columns of $\boldsymbol{A}$ forces our hand. There's only one orthonormal basis with that property. We can see this simply by examining the matching spans property. It's certainly true for $k = 1$. The one-dimensional span of $\boldsymbol{q}_1$ must match the one dimensional span of $\boldsymbol{a}_1$. So the only possibility is that $\boldsymbol{q}_1 \propto \boldsymbol{a}_1$. Choosing it to be positively aligned with $\boldsymbol{a}_1$ defines it uniquely. Now assume that in general the first k orthonormal vectors are uniquely defined. Then we need to find a new vector direction $\boldsymbol{q}_{k+1}$ that is both orthogonal to all the previous directions and for which $\boldsymbol{a}_{k+1} \in \mathbf{span}(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k, \boldsymbol{q}_{k+1})$. The only possible solution is to choose $\boldsymbol{q}_{k+1} \propto \boldsymbol{a}_{k+1} - \boldsymbol{P}_{1:k}\boldsymbol{a}_{k+1}$, where $\boldsymbol{P}_{1:k}$ is the projection matrix that finds the orthogonal projection of a vector onto the span of the first k basis vectors. Choosing $\boldsymbol{q}_{k+1}$ to be positively aligned with that residual vector uniquely defines it.

Thus, by induction, it must be that the entire orthonormal basis is uniquely defined by the property that the intermediate k-dimensional subspaces match, along with the stipulation that each new orthonormal vector $\boldsymbol{q}_{k+1}$ be positively aligned with the residual of the projection of $\boldsymbol{a}_{k+1}$ onto the previous span of vectors. That point's important enough that it deserves a theorem.

Theorem 1. The QR-decomposition is unique. *Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be any matrix with linearly independent columns. Then the decomposition $\boldsymbol{A} = \boldsymbol{QR}$, where $\boldsymbol{Q} \in \mathbb{R}^{m \times n}$ is a matrix whose columns consist of orthonormal vectors and $\boldsymbol{R} \in \mathbb{R}^{n^2}$ is an upper triangular matrix whose diagonal elements are all positive, is unique. The columns of $\boldsymbol{Q}$ form what the unique **orthogonalization** of the vectors constituting the columns of $\boldsymbol{A}$.*

The additional requirement that the diagonal elements of $\boldsymbol{R}$ be positive is simply a restatement of the stipulation that each new orthogonal vector be positively aligned with the projection residual.

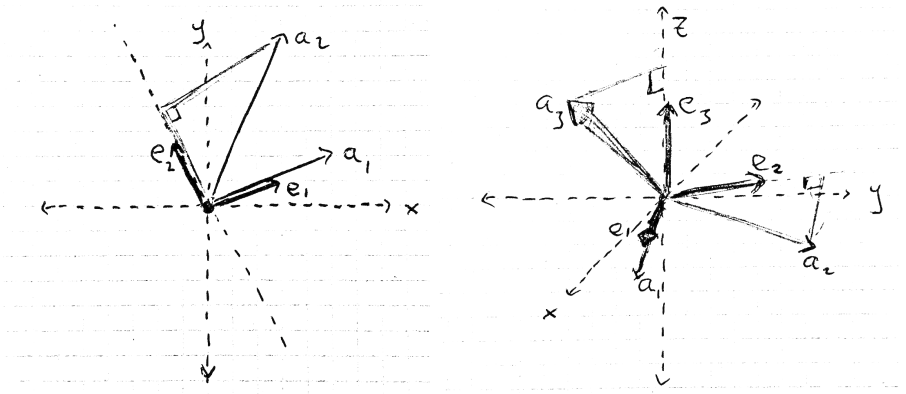

Figure 1: An example of Gram-Schmidt orthogonalization for three vectors. **Left:** The 2D subspace spanned by the first two vectors. The first orthogonal vector $\boldsymbol{e}_1$ is chosen to be proportional to $\boldsymbol{a}_1$. The second is chosen by subtracting off the component of $\boldsymbol{a}_2$ along $\boldsymbol{e}_1$. **Right:** A representation of the 3D space spanned by the third vector $\boldsymbol{a}_3$ in addition to the first two. $\boldsymbol{e}_3$ is chosen to be orthogonal to the plane spanned by $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ (equivalently, $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$).

The uniqueness of the QR-decomposition means that no matter how you compute it, independent of how strangely unintuitive the algorithm seems with regard to the subspace requirements outlined above, if you end up with an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{R}$ whose product reconstructs the matrix in question $\boldsymbol{A} = \boldsymbol{QR}$, you'll always come up with the same $\boldsymbol{Q}$ and $\boldsymbol{R}$.

3.1 Gram-Schmidt orthogonalization

The most intuitive algorithm for constructing $\boldsymbol{Q}$ and the corresponding $\boldsymbol{R}$ is the Gram-Schmidt process. The Gram-Schmidt process proceeds by directly implementing the span requirement $\mathbf{span}(\{\boldsymbol{q}_i\}_{i=1}^k) = \mathbf{span}(\{\boldsymbol{a}_i\}_{i=1}^k)$. For each k in sequence, it computes

$$\begin{aligned}\boldsymbol{q}_k &\propto \boldsymbol{a}_k - \boldsymbol{P}_{1:k-1}\boldsymbol{a}_k \\ &= \boldsymbol{a}_k - (\boldsymbol{q}_1^T\boldsymbol{a}_k)\boldsymbol{q}_1 - (\boldsymbol{q}_2^T\boldsymbol{a}_k)\boldsymbol{q}_2 - \ldots - (\boldsymbol{q}_{k-1}^T\boldsymbol{a}_k)\boldsymbol{q}_{k-1}.\end{aligned}$$

The proportionality is there because each final vector $\boldsymbol{q}_k$ needs to be normalized. This process, by construction, automatically chooses $\boldsymbol{q}_k$ to have the correct sign. Figure 1 visualizes this process for three vectors.

Now that we have a collection of basis vectors with the desired subspace span property, we can construct the upper triangular matrix $\boldsymbol{R}$ per Equation 9 which, as discussed above, simply represents how each $\boldsymbol{a}_i$ is reconstructed from the basis vectors in $\boldsymbol{Q}$. Thus, we have $\boldsymbol{A} = \boldsymbol{QR}$, the desired QR-decomposition.

It turns out that directly implementing this procedure is numerically unstable, particularly for high dimensional spaces, because the kth construction has to subtract off so many projections. In practice, the algorithm is typically implemented using a provably more numerically robust variant that instead computes the projection components incrementally:

$$\begin{aligned}
\boldsymbol{u_k} &\leftarrow \boldsymbol{a}_k \\
\boldsymbol{u_k} &\leftarrow \boldsymbol{u}_k - (\boldsymbol{q}_1^T \boldsymbol{u}_k)\boldsymbol{q}_1 \\
\boldsymbol{u_k} &\leftarrow \boldsymbol{u}_k - (\boldsymbol{q}_2^T \boldsymbol{u}_k)\boldsymbol{q}_2 \\
&\vdots \\
\boldsymbol{u_k} &\leftarrow \boldsymbol{u}_k - (\boldsymbol{q}_{k-1}^T \boldsymbol{u}_k)\boldsymbol{q}_{k-1},
\end{aligned}$$

finally setting $\boldsymbol{q}_k = \frac{\boldsymbol{u}_k}{\|\boldsymbol{u}_k\|}$.

Note that if $m > n$ the Gram-Schmidt process still computes what is sometimes referred to as the "thin" QR-decomposition $\boldsymbol{A} = \boldsymbol{Q}_{m\times n}\boldsymbol{R}$ where $\boldsymbol{Q}_{m\times n} \in \mathbb{R}^{m\times n}$ consists of n mutually orthonormal columns and $\boldsymbol{R} \in \mathbb{R}^{n^2}$ is an upper triangular matrix.

3.2 Householder reflections

The above Gram-Schmidt orthogonalization procedure is the obvious procedure given the theoretical requirements of the QR-decomposition. However, even the more numerically robust variant doesn't have the best numerical properties (Golub and Van Loan, 1996). Here we take a vastly different approach. Rather than attempting in any way to orthogonalize the vectors, we simply try to construct a collection of orthogonal transformations that reduce the matrix to an upper triangular form. If we're successful, since the QR-decomposition is unique, the final result must be the same as what we'd get from performing the more intuitive Gram-Schmidt algorithm (as long as we follow the sign convention).

First, we ask what does it mean to transform a vector $\boldsymbol{a}_1$ so that all of its entries are zero except the first? Such a vector is simply a multiple of the canonical basis vector $\boldsymbol{b}_1 = (1, 0, \ldots, 0)^T$. So somehow, we want to transform $\boldsymbol{a}_1$ so that it aligns with $\boldsymbol{b}_1$.

In general, how might we do that? Suppose we have a vector $\boldsymbol{a}$ and we want to transform it, using an orthogonal transformation, to align with a normalized vector $\boldsymbol{b}$. Consider the following. If we re-scale $\boldsymbol{b}$ to have the same length as $\boldsymbol{a}$ via $\widetilde{\boldsymbol{b}} = \|\boldsymbol{a}\|\boldsymbol{b}$, the two vectors $\boldsymbol{a}$ and $\widetilde{\boldsymbol{b}}$ end up forming two sides of an isosceles triangle with the difference vector $\boldsymbol{v} = \boldsymbol{a} - \widetilde{\boldsymbol{b}}$ forming the base of the triangle. (See Figure 2.) If we were to reflect the entire space across the subspace orthogonal to $\boldsymbol{v}$ that would implement the desired operation. $\boldsymbol{a}$ would go to where $\boldsymbol{b}$ was and $\boldsymbol{b}$ would go to where $\boldsymbol{a}$ was.

Mathematically, such a reflection is the result of a transform called a **Householder reflection,** formally defined as

$$\boldsymbol{H} = \boldsymbol{I} - 2\widehat{\boldsymbol{v}}\widehat{\boldsymbol{v}}^T,$$

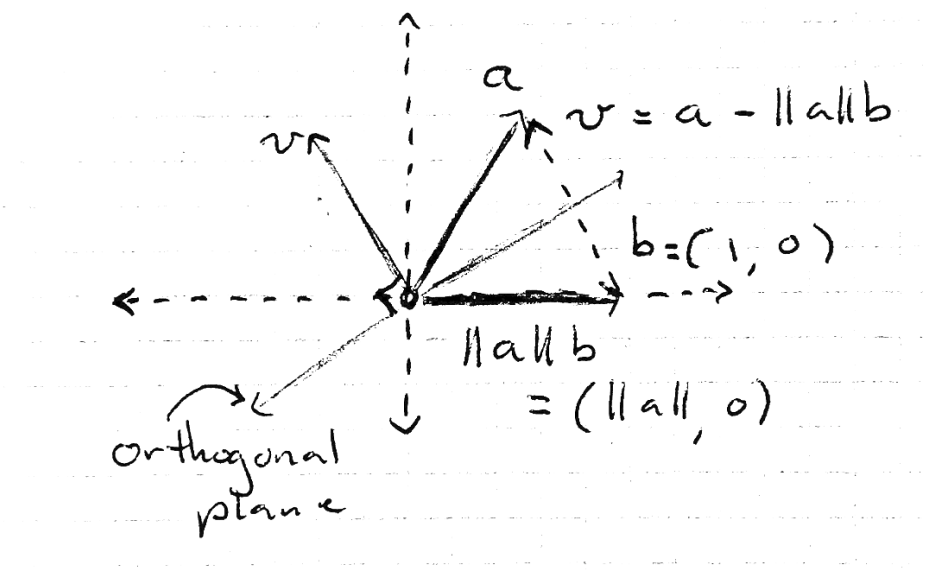

Figure 2: A 2D depiction of a Householder reflection reflecting vector $\boldsymbol{a}$ across the plane orthogonal to the difference vector $\boldsymbol{v}$ so that it aligns with the desired axis $\boldsymbol{b} = (1, 0)$.

where $\widehat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$. This matrix operates as

$$\boldsymbol{H}\boldsymbol{x} = \left(\boldsymbol{I} - 2\widehat{\boldsymbol{v}}\widehat{\boldsymbol{v}}^T\right)\boldsymbol{x} = \boldsymbol{x} - 2(\widehat{\boldsymbol{v}}^T\boldsymbol{x})\widehat{\boldsymbol{v}}.$$

In other words, the matrix subtracts twice the component of $\boldsymbol{x}$ along $\widetilde{\boldsymbol{v}}$. Figure 2 depicts this operation.

One of the remarkable properties of these Householder reflections is that they're both symmetric and orthogonal. In other words, it's its own inverse! Remembering that it's a reflection, though, makes that property clear. Reflecting twice puts the space back how we found it. For our purposes, the most important property is that this matrix is orthogonal—a perfect candidate for constructing an orthogonal matrix that reduces $\boldsymbol{A}$ to upper triangular form.

Starting from $k = 1$, for each k we want to construct a Householder rotation that zeros out all the components of $\boldsymbol{a}_k$ for dimensions $k + 1$ through m. We can do so using a matrix of the form

$$\boldsymbol{Q}_k = \begin{pmatrix} \boldsymbol{I}_{k-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} - 2\widehat{\boldsymbol{v}}_k\widehat{\boldsymbol{v}}_k^T \end{pmatrix}, \tag{10}$$

where, denoting the vector consisting of the last $m - k$ components of $\boldsymbol{a}_k$ as $\widetilde{\boldsymbol{a}}_k \in \mathbb{R}^{m-k}$, we have $\boldsymbol{v} = \widetilde{\boldsymbol{a}}_k - \|\widetilde{\boldsymbol{a}}_k\|\boldsymbol{b}_k$ with $\boldsymbol{b}_k = (1, 0, \ldots, 0)^T \in \mathbb{R}^{m-k}$. This matrix is constructed to leave the first k columns and rows of the matrix

untouched but to zero out entries $k+1$ through m of specifically the kth column. Thus, applying $\boldsymbol{Q}_1$ through $\boldsymbol{Q}_n$ in sequence to $\boldsymbol{A}$ successively zeros out the entries of the matrix below the diagonal one column at a time. Moreover, since each of these matrices are symmetric and orthogonal we can say

$$\boldsymbol{Q}_n \cdots \boldsymbol{Q}_1 \boldsymbol{A} = \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} \quad \Rightarrow \quad \boldsymbol{A} = \underbrace{\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_n}_{\boldsymbol{Q}} \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} = \boldsymbol{Q}_{m \times n} \boldsymbol{R}, \tag{11}$$

where $\boldsymbol{Q}$, which is a product of orthogonal matrices, is itself an orthogonal matrix, and $\boldsymbol{R}$ is, by construction an upper triangular matrix. Note that the zeros below the upper triangular matrix remove the $m - n$ right-most columns of $\boldsymbol{Q}$ giving $\boldsymbol{Q}_{m \times n} \in \mathbb{R}^{m \times n}$ which results a form more reminiscent of Gram-Schmidt orthogonalization.

For this algorithm, we haven't stipulated whether the diagonal elements of $\boldsymbol{R}$ are positive or negative. We could have ensured positivity by choosing the reflection more carefully, but that makes the description more complicated. So, instead, we just say that if the ith diagonal entry of $\boldsymbol{R}$ ends up being negative, we can simply flip the sign of the ith orthogonal vector and the corresponding ith row of $\boldsymbol{R}$. Doing so doesn't change the resulting matrix product (the signs cancel), and it ensures that this direct zeroing out procedure computes the unique (positive diagonal) QR-decomposition of the matrix $\boldsymbol{A}$.

This algorithm, known as the Householder-QR algorithm, is one of the state-of-the-art approaches to QR-decomposition. The baseline algorithm we discuss here isn't the most efficient variant (or even at all an efficient version)—more efficient variants never actually form the full Householder reflection matrices, and often preprocess the original matrix to an tri-diagonal form before proceeding—but the above construction lays out the basic principles of the algorithm. A good reference for understanding all the details and efficiency tricks of the Householder-QR method, and another variant based on what are called Givens rotations which is more amenable to parallelization, is Golub and Van Loan (1996).

3.3 A Note on Computational Complexity

Even a thin SVD calculation takes around $14mn^2 + 8n^3$ flops (using the Golub-Reinsch SVD algorithm), which is pretty hefty. On the other hand, a good implementation of the Householder-QR algorithm can compute the QR-decomposition in $2n^2(m - \frac{n}{3})$ flops, which is significantly faster in practice than the thin SVD calculation. The SVD cracks open a matrix and exposes the inner workings of the linear transform it represents. It's a very powerful representation—intuitive and easy to manipulate—but it's expensive. When you can get away with just computing the QR-decomposition, which is usually the case for most least squares problems, that results in a much more efficient solution strategy.

4 The Power Method

This section develops a remarkable algorithm for finding the primary Eigenvector (the one with the largest Eigenvalue) of a linear operator that's strikingly simple but very useful in practice, especially on large scale problems. Section 4.3 gives an example of an application of this algorithm to the web that's made a lot of people a lot of money.

We'll begin by reviewing some basic ideas around Eigenvalues and Eigenvectors that set the stage for this method.

4.1 A Note on Eigenvalues and Eigenvectors

Earlier we claimed that the fundamental structure of a rank K linear transform T takes the form $T = \sum_{i=1}^{K} \sigma_i |u_i\rangle\langle v_i|$ where the vectors $|u_i\rangle$ form an orthonormal basis of a fundamental space in the co-domain and the vectors $|v_i\rangle$ form an orthonormal basis for the corresponding fundamental space in the domain. From that structure, we talked briefly about Eigenvectors and Eigenvalues for "symmetric" linear operators, and how they appear in the operator's fundamental structure (see Ratliff (2014b)). There we focused on understanding the structure of the operator which is of significant practical use in solving linear systems. But the topic of Eigenvalues and Eigenvectors runs much deeper than that, and is indeed the theory used to *prove* the fundamental structure of a linear operator.

Here, for simplicity, we'll discuss these ideas specifically for matrices of linear operators in coordinates. But analogous arguments hold as well for the general abstract case. For those of you interested in a diving deeper into these ideas, Axler (1995) gives some very nice proofs of many of these results entirely without resorting to the non-intuitive and often confusing *determinant* of a matrix. He has since written an entire book developing the entire core of linear algebra using determinant free arguments (Axler, 2014).

An **Eigenvector** $\boldsymbol{e}$ of a matrix $\boldsymbol{A}$ is a vector direction in the domain that isn't rotated at all upon transformation. That vector direction is only scaled by a value λ, called the **Eigenvalue**. The Eigenvalue might be negative, which would indicate a reflection, but there's still no rotation. Algebraically, that condition can be written

$$\boldsymbol{A}\boldsymbol{e} = \lambda \boldsymbol{e}. \tag{12}$$

By convention we usually define the Eigenvectors $\boldsymbol{e}$ as normalized $\|\boldsymbol{e}\| = 1$, so that the Eigenvector represents only the direction and the corresponding Eigenvalue is unique and represents the entirety of the scaling. Technically, from the basic definition, each normalized Eigenvector represents an entire one-dimensional subspace of Eigenvectors all pointing in the same direction (or the opposite direction), since $\boldsymbol{A}(\alpha\boldsymbol{e}) = \lambda\alpha\boldsymbol{e}$. Normalizing the Eigenvectors allows us to canonically represent those one-dimensional spaces (up to a choice of sign on the Eigenvector).

These Eigenvectors expose the basic structure of a linear operator. For instance, if we know that $\boldsymbol{A} \in \mathbb{R}^{n^2}$ has n Eigenvectors $\{\boldsymbol{e}_i\}_{i=1}^n$ with corresponding distinct Eigenvalues $\{\lambda_i\}_{i=1}^n$ (specifically, $\lambda_i > \lambda_j$ for all i, j—ordering them from largest to smallest is a common convention), then we know a lot about the matrix itself.

4.1.1 Diagonalization via Eigen-bases

We start by noting that if the Eigenvalues are all distinct, then the Eigenvectors form a basis for the domain space. Appendix A provides a proof of this property; such a basis is known as an **Eigen-basis** for the space. Having distinct Eigenvalues is only one such condition required to show the Eigenvectors form a basis. For the next construction, we simply predicate on existence of an Eigen-basis.

Suppose a matrix $\boldsymbol{A} \in \mathbb{R}^{n^2}$ has a basis of Eigenvectors $\mathcal{B} = \{\boldsymbol{e}_i\}_{i=1}^n$ with corresponding Eigenvalues $\{\lambda_i\}_{i=1}^n$. Multiplying and Eigenvector by $\boldsymbol{A}$ produces a scaled version of that Eigenvector, so in combination, we can write

$$\boldsymbol{A} \begin{pmatrix} | & & | \\ \boldsymbol{e}_1 & & \boldsymbol{e}_n \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ \lambda_1 \boldsymbol{e}_1 & & \lambda_n \boldsymbol{e}_n \\ | & & | \end{pmatrix}. \tag{13}$$

Packing those Eigenvectors and Eigenvalues into matrices of the form

$$\boldsymbol{S} = \begin{pmatrix} | & & | \\ \boldsymbol{e}_1 & \cdots & \boldsymbol{e}_n \\ | & & | \end{pmatrix} \quad \text{and} \quad \boldsymbol{D} = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}, \tag{14}$$

That product becomes

$$\boldsymbol{AS} = \boldsymbol{SD}. \tag{15}$$

Since the Eigenvectors form a basis, the matrix $\boldsymbol{S}$ is invertible, so we can rewrite this relation as a matrix decomposition of the form

$$\boldsymbol{A} = \boldsymbol{SDS}^{-1}. \tag{16}$$

This decomposition is known as the **diagonalization** of $\boldsymbol{A}$ because we can rewrite it as $\boldsymbol{D} = \boldsymbol{S}^{-1}\boldsymbol{AS}$ demonstrating that $\boldsymbol{S}$ forms a similarity transform that turns $\boldsymbol{A}$ into a diagonal matrix.

Note that if we simply find any matrix $\boldsymbol{S}$ that allows us to write $\boldsymbol{S}^{-1}\boldsymbol{AS} = \boldsymbol{D}$, it must be that $\boldsymbol{AS} = \boldsymbol{SD}$. Writing it that way says that each column of $\boldsymbol{S}$, when multiplied by $\boldsymbol{A}$ just produces a scaled version of that column vector. In other words, the columns of $\boldsymbol{S}$ must, by definition, be Eigenvectors, and the corresponding diagonal entries of $\boldsymbol{D}$ are those column's Eigenvalues. So any decomposition of the form $\boldsymbol{A} = \boldsymbol{SDS}^{-1}$ must be a decomposition in terms of Eigenvectors and Eigenvalues.

Now consider what happens when $\boldsymbol{A}$ both has a full set of distinct Eigenvalues *and* is symmetric. In that case, $\boldsymbol{A}^T = \boldsymbol{A}$, and it's fairly straightforward to see that all Eigenvectors must be orthogonal to one another. We can see this simply through an examination of the matrix-scaled inner product. The product $\boldsymbol{e}_i^T \boldsymbol{A} \boldsymbol{e}_j$ can be grouped in two ways. If we group $\boldsymbol{A}$ with $\boldsymbol{e}_j$, we get $\boldsymbol{e}_i^T(\boldsymbol{A}\boldsymbol{e}_j) = \boldsymbol{e}_i^T(\lambda_j \boldsymbol{e}_j) = \lambda_j \boldsymbol{e}_i^T \boldsymbol{e}_j$. But since $\boldsymbol{A}$ is symmetric, we can also group it with $\boldsymbol{e}_i$ on the left to get $(\boldsymbol{e}_i^T \boldsymbol{A})\boldsymbol{e}_j = (\boldsymbol{A}\boldsymbol{e}_i)^T \boldsymbol{e}_j = \lambda_i \boldsymbol{e}_i^T \boldsymbol{e}_j$. This means

$$\begin{aligned} \boldsymbol{e}_i^T \boldsymbol{A} \boldsymbol{e}_j = \lambda_j \ \boldsymbol{e}_i^T \boldsymbol{e}_j = \lambda_i \ \boldsymbol{e}_i^T \boldsymbol{e}_j \\ \text{or} \quad (\lambda_j - \lambda_i) \boldsymbol{e}_i^T \boldsymbol{e}_j = 0. \end{aligned}$$

Since $\lambda_j \neq \lambda_i$, it must be that $\boldsymbol{e}_i^T \boldsymbol{e}_j = 0$ (i.e. they're orthogonal to one another). In short, a full set of distinct Eigenvalues implies that the Eigenvectors of a symmetric matrix form an orthonormal basis.

Interestingly, since the behavior of a linear operator is defined uniquely by it's behavior on a basis, we can write the matrix $\boldsymbol{A}$ as

$$\boldsymbol{A} = \sum_{i=1}^{n} \lambda_i \ \boldsymbol{e}_i \boldsymbol{e}_i^T \tag{17}$$

since apply that representation of $\boldsymbol{A}$ to any elements of its Eigenvector basis gives

$$\begin{aligned} \boldsymbol{A}\boldsymbol{e}_j = \left(\sum_{i=1}^{n} \lambda_i \ \boldsymbol{e}_i \boldsymbol{e}_i^T \right) \boldsymbol{e}_j &= \sum_{i=1}^{n} \lambda_i \ \boldsymbol{e}_i \underbrace{\left(\boldsymbol{e}_i^T \boldsymbol{e}_j \right)}_{\delta_{ij}} \\ &= \lambda_j \boldsymbol{e}_j, \end{aligned}$$

which is true by definition since $\boldsymbol{e}_j$ is an Eigenvector with Eigenvalue λ_j.

We've seen the decomposition in Equation 17 before, but for abstract vector spaces. This decomposition is known as the Spectral Decomposition, Eigendecomposition, or diagonalization of the matrix depending on who you ask. We can even relate it back to the diagonalization we saw in Equation 16 by writing it

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T \tag{18}$$

where

$$\boldsymbol{D} = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \quad \text{and} \quad \boldsymbol{U} = \begin{pmatrix} | & | & & | \\ \boldsymbol{e}_1 & \boldsymbol{e}_2 & & \boldsymbol{e}_n \\ | & | & & | \end{pmatrix}.$$

Since $\boldsymbol{U}$ is orthogonal, $\boldsymbol{U}^{-1} = \boldsymbol{U}^T$. Knowing that the matrix has this particular Eigen-structure implies that it must take a diagonal form when transformed to the right space.

From this decomposition, we can immediately see also that a symmetric positive definite matrix must have positive Eigenvalues. In this case, we can always write $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$ in terms of $\widetilde{\boldsymbol{x}} = \boldsymbol{U}\boldsymbol{x}$

$$\widetilde{\boldsymbol{x}}^T\boldsymbol{D}\widetilde{\boldsymbol{x}} = \sum_{i=1}^{n} \lambda_i \widetilde{\boldsymbol{x}}_i^2.$$

Thus, if any λ_i is negative, we can choose a $\widetilde{\boldsymbol{x}}$ (which, in turn, implies a unique $\boldsymbol{x} = \boldsymbol{U}^T\widetilde{\boldsymbol{x}}$) so as to make this expression negative. Moreover, if any λ_i is zero, we can find an $\widetilde{\boldsymbol{x}}$ that's zero in that component and make a non-zero vector's matrix-scaled norm vanish. So, **symmetric positive definite matrices must have strictly positive Eigenvalues.**

Finally, now consider a general matrix $\boldsymbol{B} \in \mathbb{R}^{m\times n}$. The thin SVD of the matrix is $\boldsymbol{U}_{/\!/}\boldsymbol{S}\boldsymbol{V}_{/\!/}^T$ where $\boldsymbol{S}$ is diagonal, and $\boldsymbol{U}_{/\!/}$ and $\boldsymbol{V}_{/\!/}$ both have orthogonal columns. This decomposition gives

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{V}_{/\!/}\boldsymbol{S}^2\boldsymbol{V}_{/\!/}^T \quad \text{and} \quad \boldsymbol{B}\boldsymbol{B}^T = \boldsymbol{U}_{/\!/}\boldsymbol{S}^2\boldsymbol{U}_{/\!/}^T. \tag{19}$$

These expressions represent diagonalizations of the matrices $\boldsymbol{B}^T\boldsymbol{B}$ and $\boldsymbol{B}\boldsymbol{B}^T$, and it must be, therefore, that $\boldsymbol{V}_{/\!/}$ and $\boldsymbol{U}_{/\!/}$ contain the Eigenvectors of these respective matrices and $\boldsymbol{S}^2$ contains the (common) Eigenvalues. This simple observation draws a fundamental link between the singular vectors and values of a matrix and Eigenvectors and Eigenvalues. Indeed, this relationship is fundamental to the SVD's proof of existence.

4.2 Determinants and volume representations

Finally, we should talk about a number that has had significant historical impact on this subject: the **determinant**. The determinant of an operator is traditionally defined as a complicated calculation that can be made on a matrix that ends up having some interesting properties. It's those properties that we're most interested with here, so we'll just cut to the chase and define it a more intuitive way.[1]

Above we saw that any symmetric square matrix $\boldsymbol{A} \in \mathbb{R}^n$ with a full set of Eigenvalues could be written $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T = \sum_i \lambda_i \boldsymbol{e}_i\boldsymbol{e}_i^T$. Indeed, from that, we see that multiplication by $\boldsymbol{A}$ simply stretches the vector in along the Eigenvectors by factors given by the Eigenvalues (or reflects and stretches if the Eigenvalue is negative). Consider what happens to a volume under this operation. Any unit box, with sides defined by the Eigenvectors $\{\boldsymbol{e}_i\}_{i=1}^n$ (see Figure 3) is stretched by $\boldsymbol{A}$ as shown in the figure directly along those sides. The transformed box is now defined by sides $\{\lambda_i\boldsymbol{e}_i\}_{i=1}^n$. The original box was simply unit box with volume 1 since all $\boldsymbol{e}_i$ are normalized. But this new box is stretched along each dimension by λ_i, so **it's new volume is the product**

[1] This discussion is designed to emphasize intuition over completeness of the development. See (Strang, 2005; Hassani, 2013) (or any number of other linear algebra texts) for a more in-depth introduction to determinants.

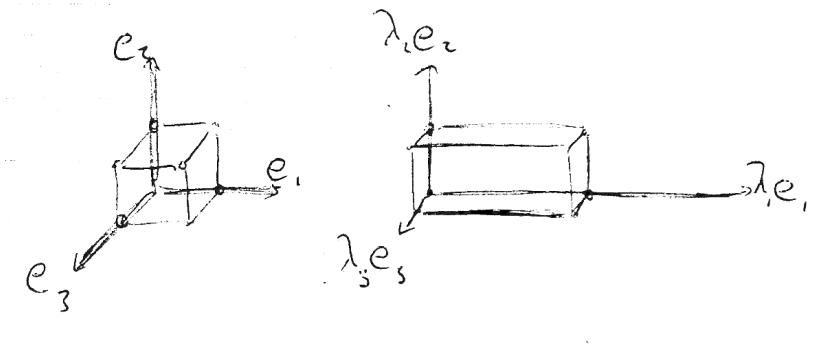

Figure 3: The sides of a unit cube aligned with the orthonormal Eigenvector axes $\boldsymbol{e}_i$ are stretched by values λ_i. **Left:** The cube in three-dimensions before transformation. **Right:** The same cube after transformation.

of all Eigenvalues. This product, which tells us how volumes are transformed by $\boldsymbol{A}$ is called the **determinant** of $\boldsymbol{A}$:

$$\textbf{det } \boldsymbol{A} = \prod_{i=1}^{n} \lambda_i. \tag{20}$$

Sometimes you'll see the determinant denoted by a vertical bars on either side of the matrix $\textbf{det } \boldsymbol{A} = |\boldsymbol{A}|$.

Note that for every negative λ_i, the determinant flips signs. Each negative λ_i denotes a reflection along axis $\boldsymbol{e}_i$ with a stretching by $|\lambda_i|$ in that direction. Negative determinants, therefore, represent an odd number of reflections. Additionally, if any λ_i is zero, the volume vanishes—the box is squished onto a reduced dimensional subspace which has no volume at all in the original space (in three dimensions, that amounts to squishing the box down to a plane ($\lambda_3 = 0$) or a line ($\lambda_2 = \lambda_3 = 0$)).

So that's what determinants are geometrically. You can see immediately that the determinant can act as an indicator for whether or not the matrix is invertible. If $\boldsymbol{A}$ squishes the box down to zero volume, we've lost information, we can't reinflate the box since whole continuums of points are mapped down to single points. On the other hand, if the box retains any volume at all, the identities of all points remain intact, so we can reverse the operation. In other words, if $\textbf{det } \boldsymbol{A} \neq 0$, the matrix is invertible, and if $\textbf{det } \boldsymbol{A} = 0$ it's not.

The magic of determinants is that all of this information can be calculated from the matrix itself without resorting to Eigenvalue calculations. We won't review the (somewhat complicated) calculation here, but we'll note a few properties of determinants which reveal why that true.

1. **Product rule.** The determinant of a matrix product is the product of determinants of the individual matrices: $\textbf{det } \boldsymbol{AB} = (\textbf{det } \boldsymbol{A})(\textbf{det } \boldsymbol{B})$.

2. **Determinant of orthogonal matrices has magnitude 1.** If $\boldsymbol{Q} \in \mathbb{R}^n$ is orthogonal, then $\det \boldsymbol{Q} = \pm 1$. Intuitively, such a matrix simply represents a change of basis (rotation, possibly with reflections), so the transformation is *rigid* (it isn't contorted in any way) and the volume is preserved.

3. **Determinant of an inverse is the inverse of the determinant.** For any invertible matrix $\boldsymbol{A} \in \mathbb{R}^{n^2}$

$$\det \boldsymbol{A}^{-1} = \frac{1}{\det \boldsymbol{A}}. \tag{21}$$

4. **Determinant of a triangular matrix is the product of its diagonal entries.** For any triangular matrix $\boldsymbol{R} \in \mathbb{R}^{n^2}$, for instance an upper triangular matrix of the form

$$\boldsymbol{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1n} \\ 0 & r_{22} & r_{23} & \cdots & r_{2n} \\ 0 & 0 & r_{33} & \cdots & r_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & r_{nn} \end{pmatrix} \tag{22}$$

   the determinant is given by $\det \boldsymbol{R} = \prod_{i=1}^{n} r_{ii}$. In particular, **the determinant of a diagonal matrix is the product of its diagonal entries.**

With these properties we can see that whenever we can diagonalize a matrix $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{D}\boldsymbol{S}^{-1}$ the determinant of the matrix is

$$\begin{aligned} \det \boldsymbol{A} = \det\,(\boldsymbol{S}\boldsymbol{D}\boldsymbol{S}^{-1}) &= (\det \boldsymbol{S})(\det \boldsymbol{D})(\det \boldsymbol{S}^{-1}) \\ &= \Big(\det \boldsymbol{S}\Big)\left(\prod_{i=1}^{n} \lambda_i\right)\left(\frac{1}{\det \boldsymbol{S}}\right) \\ &= \prod_{i=1}^{n} \lambda_i. \end{aligned}$$

No matter how we calculate the determinant of the matrix, it must be equal to the product of Eigenvalues as we defined it to be above.

We won't dig any deeper into the enormous collection of results surrounding Eigenvectors and Eigenvalues. For our purposes in the discussion below, we'll primarily be using a couple of the basic properties outlined above: Matrices with a full set of Eigenvalues have an Eigen-basis, and if they additionally are symmetric, that basis is orthonormal.

4.3 PageRank: An application

Lets take a look at an interesting and intuitive example of how Eigenvectors have had significant impact on all of our lives. Around 1996, Larry Page and

Sergey Brin, who would later go on to found Google, were still graduate students at Stanford University. Frustrated with the irrelevance of the search results typically found by search engines of the day, they set out to see if they could crawl around the web and in some way calculate the influence each web page has on the interconnected web of linked pages as a whole. Can we learn something about page popularity from the underlying structure of which pages link to each other?

They built their ideas of a simple model of how someone might simply randomly click from link to link when wandering the web. Suppose Joe flipped on his computer and had a magic button to choose some random website to start from (he's an equal-opportunity surfer). And suppose he's entirely unbiased (everything's great to this guy) so that after consuming the contents of the current page, he simply clicks on a random link on that page to get to another page. Then, after consuming the contents of that page, clicks randomly to another page, and so on.

After doing that for a while, Larry and Sergey asked, where might Joe end up? Today, we can probably say that there's a good chance he'll get stuck in a giant mess of Wikipedia pages. Why? Because Wikipedia's darned useful! Everyone links to it because it's our Hitchhiker's Guide to the Galaxy. But generally, there are other pages where we're likely to find him. YouTube, BBC, Facebook, or any number of sites are good candidates. They figured, if that's the case—if randomly wandering the web wasn't really that random—perhaps the resulting likelihood of finding yourself at a given site is a good proxy for the influence that that site has on the world. If sites link to other sites, who link to other sites, who eventually all link to Wikipedia, then maybe Wikipedia's pretty influential.

So they set out devising an algorithm to simulate that process. To model this problem properly, suppose we have a collection of sites indexed $i = 1, \ldots, N$ where N in this case is in the millions (there are a lot of sites in the world). And suppose each site i links to a collection of other sites $j \in \mathcal{L}_i$. If you find yourself at site i, then you're equally likely to subsequently step to any of the $|\mathcal{L}_i|$ sites it links to.[2] Specifically, the probability of you clicking to a specific site $j \in \mathcal{L}_i$ is $\alpha_{ij} = \frac{1}{|\mathcal{L}_i|}$.

Now at time step t (indicating the number of clicks Joe has made, starting from 0), suppose the probably distribution over where he might be is given by the vector $\boldsymbol{p}^{(t)} \in \mathbb{R}^n$. Each element of that vector $p_i^{(t)}$ denotes the probability that Joe finds himself at site i after t clicks. $\boldsymbol{p}^{(0)}$ starts off as entirely uniform (equal probability $1/N$ for all sites). What happens to the vector of probabilities after one step?

After one step, a given site j has distributed its probability evenly among the sites it links to. Since it links to $|\mathcal{L}_j|$ other sites, there is a $1/|\mathcal{L}_j|$ chance that Joe gets to a particular site $i \in \mathcal{L}_j$. The probability that Joe was at site j in the first place, though, is $p_j^{(t)}$, so the total likelihood of getting to a specific site i there

[2]The notation $|\mathcal{L}_i|$ denotes the number of elements in the set, in this case the number of distinct links on a site.

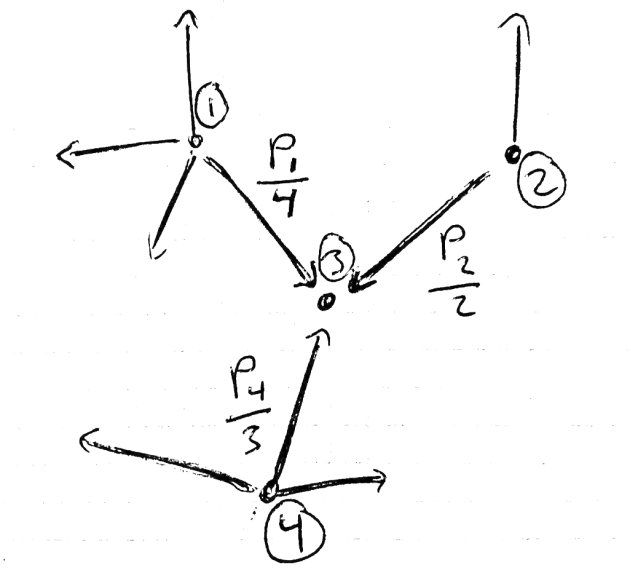

Figure 4: An example of the PageRank update for one node of the link-graph of websites. Each website linking to website 3 sends a fraction of their own influence (formally random walk probability or "page rank") to website 3. Website 1 sends $\frac{1}{4}$ of its page rank because it's distributing the value equally among the 4 pages it links to. On the other hand, website 2 only links to two other pages, so it sends half of its page rank to each.

along that route is the probability of the click weighed by the probability $p_j^{(t)}$ that Joe actually was at site j. Thus, for a given site i, the total probability of ending up at that site after this new click is sum of the probabilities of originally being at any of the sites linking to i and actually clicking on the specific link that gets to site i. Notationally, we can write this as

$$p_i^{(t+1)} = \sum_{j \mid i \in \mathcal{L}_j} \frac{p_j^{(t)}}{|\mathcal{L}_j|}. \tag{23}$$

The notation $j \mid i \in \mathcal{L}_j$ denotes the set of all sites j who link to the site in question i (i.e. for which i is in $\mathcal{L}_j$). Iterating this procedure constitutes (a basic form) of the famous algorithm we know as PageRank (Brin and Page, 1998). Figure 4 shows an example of the update for one node of the graph of web pages.

Equation 23 is very interesting. It says that the probability of getting to site i after $t+1$ steps is just a linear combination of the probabilities of where you

were one time step ago. If we write

$$\alpha_{ij} = \begin{cases} \frac{1}{|\mathcal{L}_j|} & \text{for } i \in \mathcal{L}_j \\ 0 & \text{otherwise} \end{cases}, \tag{24}$$

then we can write Equation 23 in matrix form

$$\boldsymbol{p}^{(t+1)} = \boldsymbol{A}\boldsymbol{p}^{(t)}, \tag{25}$$

where

$$\boldsymbol{A} = \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & \ddots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix}. \tag{26}$$

Equation 25 tells us that this algorithm for measuring the influence of sites based on where you might end up after randomly wandering around is just a series of matrix multiplications. We start with a uniform vector of probabilities $\boldsymbol{p}^{(0)}$ and at each iteration multiply by a matrix $\boldsymbol{A}$. Where do we get after k iterations? Simply $\boldsymbol{p}^{(k)} = \boldsymbol{A}^k\boldsymbol{p}^{(0)}$, where $\boldsymbol{A}^k$ denotes $\boldsymbol{A}$ multiplied together k times. So the question becomes, does this algorithm converge? Does the limit

$$\lim_{k\to\infty} \boldsymbol{A}^k\boldsymbol{p}^{(0)} \tag{27}$$

exist? If so, what is it?

4.4 Analyzing the Power Method

Section 4.3 gives an example of a very general class of algorithm. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{n^2}$, what do we get when we multiply a vector $\boldsymbol{v}$ by $\boldsymbol{A}$ over and over? Does it even converge?

The answers to these questions lie in the Eigenvalues. Suppose our matrix $\boldsymbol{A}$ has a full set of real-valued Eigenvalues $\{\lambda_i\}_{i=1}^n$ with corresponding Eigenvectors $\{\boldsymbol{e}_i\}_{i=1}^n$, and suppose $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n|$, i.e. the first Eigenvector strictly dominates in absolute value.

As we discussed above, those Eigenvectors form a basis for the domain, so for any vector $\boldsymbol{v}$ that we'd like to transform by the matrix, we can analyze the behavior of the transformation by expanding the vector in terms of its Eigenvector basis. Let $c_1, \ldots, c_n$ be real numbers such that $\boldsymbol{v} = c_1\boldsymbol{e}_1 + c_2\boldsymbol{e}_2 + \ldots + c_n\boldsymbol{e}_n$. Since the basis consists of Eigenvectors, transforming $\boldsymbol{v}$ gives

$$\begin{aligned} \boldsymbol{A}\boldsymbol{v} &= c_1\boldsymbol{A}\boldsymbol{e}_1 + c_2\boldsymbol{A}\boldsymbol{e}_2 + \ldots + c_n\boldsymbol{A}\boldsymbol{e}_n \\ &= c_1\lambda_1\boldsymbol{e}_1 + c_2\lambda_2\boldsymbol{e}_2 + \ldots + c_n\lambda_n\boldsymbol{e}_n. \end{aligned}$$

Multiplying by $\boldsymbol{A}$ scales each coefficient c_i by λ_i since, by definition, $\boldsymbol{A}\boldsymbol{e}_i = \lambda_i\boldsymbol{e}_i$. Thus, iterating this process gives

$$\boldsymbol{A}^k\boldsymbol{v} = c_1\lambda_1^k\boldsymbol{e}_1 + c_2\lambda_2^k\boldsymbol{e}_2 + \ldots + c_n\lambda_n^k\boldsymbol{e}_n. \tag{28}$$

Every time we multiply by $\boldsymbol{A}$ we multiply each coefficient by another factor of λ_i. Since λ_1 is the largest Eigenvalue in absolute value, the ratios $r_i = \frac{\lambda_i}{\lambda_1}$ are all smaller than 1 in absolute value except the first, which is exactly 1. So as k gets larger, $r_i^k = \left(\frac{\lambda_i}{\lambda_1}\right)^k$ all get smaller and smaller, each approaching zero except the first. That first ratio is always 1. This means that as we keep multiplying by $\boldsymbol{A}$ the influence of each Eigenvector $\boldsymbol{e}_i$, relative to the influence of the first Eigenvector $\boldsymbol{e}_1$, becomes diminishingly small. That first Eigenvalue dominates exponentially fast as we continue to multiply by $\boldsymbol{A}$.

We've been describing the behavior of this method as though the dominant Eigenvalue were positive, but it might actually be negative. In that case, the factors λ_1^k flip sign depending on whether we've multiplied an even or odd number of times, but the basic convergence behavior in terms of the dominant Eigen-direction (if not the specific vector, which continues to flip signs) is still consistent with the above argument.

Thus, we're now equipped to answer the questions posed above. And the answer is... sometimes. As long as we have a full set of Eigenvalues, then we have a full set of linearly independent Eigenvectors and we can expand the initial vector in the Eigen-basis. We see that multiplying repeatedly by $\boldsymbol{A}$ makes the primary Eigenvector dominate. Unfortunately, if that primary Eigenvalue is greater than 1, the vector explodes and grows to infinity. And if it's smaller than 1, the process shrinks the vector to 0. It's only that unique case where λ_1 is exactly 1 that converges in the limit to the primary Eigenvector. That issue can be fixed pretty easily by normalizing the vectors after each iteration (see Section 4.6), but even before that we're currently ready to return to the question of PageRank. Does that algorithm converge? And if so, to what?

4.5 Does PageRank Converge?

What do we need to prove convergence? The PageRank algorithm, as we established in Section 4.3, is literally a sequence of matrix multiplications. So the inner workings of those multiplications, as exposed in the analysis above, are directly applicable. We need the maximal Eigenvalue of this PageRank matrix $\boldsymbol{A}$, whose elements are defined by Equation 24, to be 1. And we need that maximal Eigenvalue to be strictly larger than all others in absolute value.

For that analysis, we need to turn to the Perron-Frobenius theorem (Wikipedia (2002-2014) links to some comprehensive references). That theorem states that **any matrix with strictly positive entries (no zero entries) has a unique real-valued maximal Eigenvalue and a corresponding Eigenvector with strictly positive entries.** In this case, though, our matrix doesn't have strictly positive entries! It has a lot of zeros. Since we've restricted Joe to wander only among web pages linked to each other, there is zero probability that he jumps to a site not linked from the site he's currently at. The zeros can be seen explicitly in Equation 24.

And unfortunately, that Perron-Frobenius theorem, when applied to **stochastic matrices** such as the PageRank random walk matrix we're considering here,

isn't just being overly conservative. We can give a simple counter example showing that the algorithm may, indeed, just not converge. Consider a simple graph consisting of only two websites, website 1 and website 2. And suppose website 1 links to website 2 and website 2 links to website 1. In this case, no matter what probability distribution $\boldsymbol{p}^{(0)} = (p_1^{(0)},\ p_2^{(0)})$ you start with, after one iteration your probabilities will just swap $\boldsymbol{p}^{(1)} = (p_2^{(0)},\ p_1^{(0)})$ since all of the probability mass of either website is shoveled off to the other website. And after two iterations, they'll just swap back to $\boldsymbol{p}^{(2)} = (p_1^{(0)},\ p_2^{(0)})$. The process will simply oscillate that way indefinitely with no variation, so convergence is out of the question.

We can see this issue arise in the Eigenvalues of this transition matrix. For this simple example we have

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{29}$$

This matrix has two Eigenvalues, 1 and -1, and neither dominates the other in absolute value. Thus, the Power Method doesn't converge to anything.

On the other hand, if there was any non-zero probability $b = .01$ of returning back to the current website, then the matrix becomes

$$\boldsymbol{A} = \begin{pmatrix} .01 & .99 \\ .99 & .01 \end{pmatrix}, \tag{30}$$

which has two Eigenvalues 1 and $-.98$, the first of which strictly dominates in absolute value. Over time, intuitively, the probabilities of being at a specific website diffuse into each other until they've fully converged to a uniform distribution. Indeed, the Eigenvector corresponding to that primary Eigenvalue of 1 has equal entries.

So, somehow, we need to ensure that, in practice, there are no truly zero entries. Fortunately, to do this we just need to make the model a little more realistic. In reality, we don't wander around the web indefinitely clicking from page to page without variation. Instead, we might start at a random page, link from page to page for a while, but then get bored and jump to another random page. With some probability at any time step, we end up at any random page.

Suppose there's a probability $0 < b < 1$ that at any moment in time we get bored and jump to a random page. If we do get bored (probability b), the probability of ending up at any of the N web pages is uniform $\frac{1}{N}$. On the other hand, if we haven't gotten bored yet (probability $1 - b$), we simply continue clicking links as usual. So, in combination, the new probability of jumping from page j to page i is given by the matrix elements

$$\widetilde{\alpha}_{ij} = \begin{cases} \frac{1-b}{|\mathcal{L}_j|} & \text{for } i \in \mathcal{L}_j \\ \\ \frac{b}{N} & \text{otherwise} \end{cases} \tag{31}$$

Now we form a new matrix $\widetilde{\boldsymbol{A}}$ from these elements. Multiplying by that matrix implements what is known as the *damped* PageRank algorithm. This new matrix

still has the property that each column j represents a probability distribution of where you might end up next after having been at website j, so formally it's still a *stochastic* matrix, but now all of the elements of the matrix are strictly positive. We've gotten rid of the zeros by actually making the model more realistic.

Now the Perron-Frobenius theorem is directly applicable, and it shows that this matrix has a unique real Eigenvalue and a corresponding Eigenvector with strictly positive entries. But we can say more. We know that after each iteration, the probability distribution over websites remains a probability distribution. It doesn't explode to infinity, shrink to zero, or flip signs repeatedly. The only way that can happen is if that unique maximal Eigenvalue is exactly 1. Thus, here we see explicitly that this *damped* PageRank algorithm does actually converge, and it converges to the primary Eigenvector[3] of the damped PageRank matrix $\widetilde{\boldsymbol{A}}$.

For a nice introduction to stochastic matrices and the Perron-Frobenius theorem, see Knill (2011a,b). Our analysis has show that the naïve algorithm introduced in Section 4.3 doesn't actually work as advertised. We need to augment it with a damping factor that models the (perhaps small) likelihood of starting over and jumping randomly to any page an any moment in time.

4.6 Normalized Power Method

In general, though, we don't want to rely on $\lambda_1 = 1$ for ensured convergence. We often can't say anything about whether the largest Eigenvalue is greater than 1 or less than 1 without a lot of tough analysis. And as alluded above, that causes practical problems for the Power Method since the limit diverges $\lambda_1^k \to \infty$ for $\lambda_1 > 0$ and drives to zero $\lambda_1^k \to \mathbf{0}$ for $\lambda_1 < 0$. We can't have our algorithm diverging or converging to zero, so how do we fix this?

Looking back at Equation 28, we see that the dominance of the primary Eigenvector is agnostic to whether or not we scale the entire vector by some constant value at each iteration. The behavior of the algorithm is still governed by the relative ratios $r_i = \frac{\lambda_i}{\lambda_1}$. This suggests that we can normalize the vector after every iteration and what results will be a normalized vector parallel to $\boldsymbol{e}_1$. In other words, what results is $\boldsymbol{e}_1$, itself (or its negative).

With this modification, the Normalized Power Method suggests the following, more robust and reliable updates:

$$\widetilde{\boldsymbol{v}}_{t+1} = \boldsymbol{A}\boldsymbol{v}_t \quad \text{followed by} \quad \boldsymbol{v}_{t+1} = \frac{\widetilde{\boldsymbol{v}}_{t+1}}{\|\widetilde{\boldsymbol{v}}_{t+1}\|}. \tag{32}$$

[3]Note that the primary Eigenvector, by convention, is normalized to unit length, which is different from the condition of summing to 1 required by probability distribution. It must be, then, that $c_1 = \frac{1}{\mathbf{1}^T \boldsymbol{e}_i}$, where $\mathbf{1}$ is the vector of ones. The algorithm then converges to $c_1 \boldsymbol{e}_1 = \frac{\boldsymbol{e}_1}{\mathbf{1}^T \boldsymbol{e}_i}$, the probability distribution aligned with the primary Eigenvector.

4.7 An Extension to Multiple Eigenvectors through Orthogonalization

What if we want more than just the single largest Eigenvector? The answer is kind of what you would expect: You simply run the method for multiple linearly independent vectors, and then orthogonalize the result. There's a caveat that we'll explain in a minute, but for now, let's analyze this simplified version.

Consider a collection of initial vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l$. Each has a decomposition in terms of the Eigen-basis of $\boldsymbol{A}$, which we can denote as

$$\begin{aligned}
\boldsymbol{v}_1 &= c_{11}\boldsymbol{e}_1 + c_{12}\boldsymbol{e}_2 + \ldots + c_{1n}\boldsymbol{e}_n \\
\boldsymbol{v}_2 &= c_{21}\boldsymbol{e}_1 + c_{22}\boldsymbol{e}_2 + \ldots + c_{2n}\boldsymbol{e}_n \\
&\vdots \\
\boldsymbol{v}_l &= c_{l1}\boldsymbol{e}_1 + c_{l2}\boldsymbol{e}_2 + \ldots + c_{ln}\boldsymbol{e}_n.
\end{aligned}$$

After k iterations of the Power Method, the collection of vectors has the form

$$\begin{aligned}
\boldsymbol{A}^k\boldsymbol{v}_1 &= c_{11}\lambda_1^k\boldsymbol{e}_1 + c_{12}\lambda_2^k\boldsymbol{e}_2 + \ldots + c_{1n}\lambda_n^k\boldsymbol{e}_n \\
\boldsymbol{A}^k\boldsymbol{v}_2 &= c_{21}\lambda_1^k\boldsymbol{e}_1 + c_{22}\lambda_2^k\boldsymbol{e}_2 + \ldots + c_{2n}\lambda_n^k\boldsymbol{e}_n \\
&\vdots \\
\boldsymbol{A}^k\boldsymbol{v}_l &= c_{l1}\lambda_1^k\boldsymbol{e}_1 + c_{l2}\lambda_2^k\boldsymbol{e}_2 + \ldots + c_{ln}\lambda_n^k\boldsymbol{e}_n.
\end{aligned}$$

Now we apply Gram-Schmidt to these vectors to orthogonalize them. To sketch the idea. After iterating for a while, the first $\boldsymbol{A}^k\boldsymbol{v}_1$ is pretty well aligned with $\boldsymbol{e}_1$. If we think of it as effectively fully aligned with $\boldsymbol{e}_1$, subtracting the component of $\boldsymbol{A}^k\boldsymbol{v}_2$ along $\boldsymbol{A}^k\boldsymbol{v}_1$ effectively removes the $\boldsymbol{e}_1$ term from $\boldsymbol{A}^k\boldsymbol{v}_2$. Doing that for all vectors $\boldsymbol{A}^k\boldsymbol{v}_2$ through $\boldsymbol{A}^k\boldsymbol{v}_l$ removes the $\boldsymbol{e}_1$ term from them all, leaving us with the same situation as before for these remaining vectors, except with the $\boldsymbol{e}_2$ term dominating. Completing the Gram-Schmidt operation recursively removes the leading Eigenvector terms as we continue down the line.

There are some details regarding the fact that after a finite number of iterations the first vector isn't actually fully aligned with $\boldsymbol{e}_1$. So in slightly more detail, to gain some intuition for the behavior of the orthogonalization in this case, lets take a look at the first two vectors. Gram-Schmidt starts by subtracting off the component of $\boldsymbol{A}^k\boldsymbol{v}_2$ along the direction $\boldsymbol{A}^k\boldsymbol{v}_1$. We can denote the resulting vector as

$$\begin{aligned}
\boldsymbol{c}_2 &= \boldsymbol{A}^k\boldsymbol{v}_2 - \alpha\boldsymbol{A}^k\boldsymbol{v}_1 \\
&= (c_{21} - \alpha c_{11})\lambda_1^k\boldsymbol{e}_1 + (c_{22} - \alpha c_{12})\lambda_2^k\boldsymbol{e}_2 + \ldots + (c_{2n} - \alpha c_{1n})\lambda_n^k\boldsymbol{e}_n.
\end{aligned}$$

where α is the requisite scaling factor for the projection. Since λ_1^k dominates significantly, the projection coefficient used in α is scaled primarily to make $c_{21} - \alpha c_{11}$ vanish. What's left is a collection of terms corresponding to the remaining Eigenvectors with new coefficients $\widetilde{c}_{2i} = c_{2i} - \alpha c_{1i}$. Since the calculation of α

was dominated by that first term, these remaining coefficients are essentially just a new collection of random coefficients, as though the original vector itself was just constructed from these coefficients. And for this remaining portion, we recursively have the situation where now λ_2^k dominates the remaining terms.

Repeating that argument recursively across all vectors and Eigen-components in turn, we see that Gram-Schmidt orthogonalization incrementally removes the leading Eigenvector terms leaving vectors dominated by the first remaining Eigenvector term.

So this procedure of running k Power Method iterations on l linearly independent vectors and then orthogonalizing the resulting vectors in theory converges, as k approaches infinity, to the first l Eigenvectors. In practice, though, since any vector with a component in the direction of the primary Eigenvector converges to that primary Eigenvector, the linearly independent vectors start aligning with one another over time. If we had infinite precision, that would be fine. But we don't. So we need to worry about numerical stability of the problem. What's the trick? Simply orthogonalize after each iteration. The resulting algorithm works well in practice and is called the *Block* Power Method.

5 The Almighty QR-Algorithm

We're now ready to describe a surprisingly simple and effective algorithm for estimating the complete set of Eigenvalues of a matrix, called the QR-Algorithm, named for its repeated use of orthogonalizations. The analysis uses similar arguments to those developed above for analyzing the behavior of the Block Power Method described in Section 4.7, but it ends up being much more efficient in practice. For simplicity, we'll restrict ourselves to the case where $\boldsymbol{A}$ is symmetric in order to build more intuitive arguments, but for a more complete development of these ideas see (Olver, 2010).

The algorithm, itself, is very easy to describe. But at face value it looks absolutely ridiculous:

The QR-Algorithm:

1. Start with $\boldsymbol{B}_0 = \boldsymbol{A}$.
2. Iterate
   (a) Take the QR-decomposition of the current matrix $\boldsymbol{B}_t = \boldsymbol{Q}_t \boldsymbol{R}_t$.
   (b) Form the new matrix by flipping the order of $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$ to form $\boldsymbol{B}_{t+1} = \boldsymbol{R}_t \boldsymbol{Q}_t$.

And that's it. **When $\boldsymbol{A}$ is symmetric,[4] $\boldsymbol{B}_t$ converges to a diagonal matrix whose diagonal elements are the original matrix's Eigenvalues.**

[4] In the more general case, we need to modify the statement to say $\boldsymbol{B}_t$ converges to a *upper triangular* matrix whose diagonal elements are the Eigenvalues of the original matrix. See (Olver, 2010).

Why is that? First, let's expand these definitions out a bit to see what each $\boldsymbol{B}_t$ really is. Since each matrix is constructed as $\boldsymbol{B}_{t+1} = \boldsymbol{R}_t\boldsymbol{Q}_t$ where both of those factors comes from the QR-decomposition of the previous $\boldsymbol{B}_t$, we can write

$$\begin{aligned}\boldsymbol{Q}_t\boldsymbol{B}_{t+1} &= \boldsymbol{Q}_t\,\boldsymbol{R}_t\boldsymbol{Q}_t = \boldsymbol{B}_t\boldsymbol{Q}_t \\ \Rightarrow \boldsymbol{B}_{t+1} &= \boldsymbol{Q}_t^T\boldsymbol{B}_t\boldsymbol{Q}_t.\end{aligned}$$

In other words, each new $\boldsymbol{B}_{t+1}$ is just a similarity transform away from $\boldsymbol{B}_t$. Repeating this argument recursively all the way down to $\boldsymbol{B}_0 = \boldsymbol{A}$, we see that

$$\boldsymbol{B}_{t+1} = \boldsymbol{Q}_t^T\boldsymbol{Q}_{t-1}^T\cdots\boldsymbol{Q}_0^T\boldsymbol{A}\boldsymbol{Q}_0\boldsymbol{Q}_1\cdots\boldsymbol{Q}_t.$$

So in truth, $\boldsymbol{B}_{t+1}$ actually just a similarity transform away from $\boldsymbol{A}$ itself for every t. Since $\boldsymbol{A}$ is symmetric, we can write it in terms of its Eigen-decomposition, as discussed in Section 4.1, as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$ where $\boldsymbol{U}$ is an orthogonal matrix of Eigenvectors and, importantly, $\boldsymbol{D}$ is the diagonal matrix of Eigenvalues. Wouldn't it be great if $\widetilde{\boldsymbol{Q}}_t = \boldsymbol{Q}_0\boldsymbol{Q}_1\cdots\boldsymbol{Q}_t$ limited to $\lim_{t\to\infty}\widetilde{\boldsymbol{Q}}_t = \boldsymbol{U}$? If it did, then in the limit we'd have $\boldsymbol{B}_t \to \boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U} = (\boldsymbol{U}^T\boldsymbol{U})\boldsymbol{D}(\boldsymbol{U}^T\boldsymbol{U}) = \boldsymbol{D}$. In other words, if this orthogonal matrix $\widetilde{\boldsymbol{Q}}_t$, constructed as the product of the orthogonal factors of the QR-decompositions of each $\boldsymbol{B}_t$, were to approach the matrix $\boldsymbol{U}$ consisting of $\boldsymbol{A}$'s Eigenvectors, then the $\boldsymbol{B}_t$'s themselves must be approaching the diagonal matrix of Eigenvalues $\boldsymbol{D}$.

Okay, now lets look more closely at what the definition of $\boldsymbol{B}_t$ mean for the Power Method product $\boldsymbol{A}^k$. First, naïvely, $\boldsymbol{A}^k = (\boldsymbol{Q}_0\boldsymbol{R}_0)(\boldsymbol{Q}_0\boldsymbol{R}_0)\cdots(\boldsymbol{Q}_0\boldsymbol{R}_0)$ multiplied k times. But aside from the first and last factors, we can also regroup those $\boldsymbol{Q}_0$ and $\boldsymbol{R}_0$ factors in reverse to construct $\boldsymbol{B}_1$ factors:

$$\begin{aligned}\boldsymbol{A}^k &= \boldsymbol{Q}_0\,(\boldsymbol{R}_0\boldsymbol{Q}_0)\,(\boldsymbol{R}_0\boldsymbol{Q}_0)\,\cdots\,(\boldsymbol{R}_0\boldsymbol{Q}_0)\,\boldsymbol{R}_0 \\ &= \boldsymbol{Q}_0\,\boldsymbol{B}_1\boldsymbol{B}_1\cdots\boldsymbol{B}_1\,\boldsymbol{R}_0.\end{aligned}$$

But now we also take the QR-decomposition of each new matrix $\boldsymbol{B}_1 = \boldsymbol{Q}_1\boldsymbol{R}_1$, so we can repeat the argument.

$$\begin{aligned}\boldsymbol{A}^k &= \boldsymbol{Q}_0\,\boldsymbol{B}_1\boldsymbol{B}_1\cdots\boldsymbol{B}_1\,\boldsymbol{R}_0 \\ &= \boldsymbol{Q}_0\,(\boldsymbol{Q}_1\boldsymbol{R}_1)\,(\boldsymbol{Q}_1\boldsymbol{R}_1)\,\cdots\,(\boldsymbol{Q}_1\boldsymbol{R}_1)\,\boldsymbol{R}_0 \\ &= \boldsymbol{Q}_0\boldsymbol{Q}_1\,(\boldsymbol{R}_1\boldsymbol{Q}_1)\,(\boldsymbol{R}_1\boldsymbol{Q}_1)\,\cdots\,(\boldsymbol{R}_1\boldsymbol{Q}_1)\,\boldsymbol{R}_1\boldsymbol{R}_0 \\ &\;\;\vdots \\ &= \boldsymbol{Q}_0\boldsymbol{Q}_1\cdots\boldsymbol{Q}_k\,\boldsymbol{R}_k\boldsymbol{R}_{k-1}\cdots\boldsymbol{R}_0.\end{aligned}$$

Finally, we note that the product of orthogonal matrices is also orthogonal, and the product of upper triangular matrices is also upper triangular. So what we've written here is $\boldsymbol{A}^k = \widetilde{\boldsymbol{Q}}_k\widetilde{\boldsymbol{R}}_k$, the unique QR-decomposition of the kth power of $\boldsymbol{A}$. Since we can consider $\boldsymbol{A}^k$ itself as the kth transformation of the identity matrix $\boldsymbol{I}$, whose columns consist of each canonical basis vector $(0,\ldots,0,1,0,\ldots,0)$,

the matrix $\widetilde{\boldsymbol{Q}}_k$ of $\boldsymbol{A}^k$'s QR-decomposition gives the orthogonalization of the transformed basis vectors. From the argument proposed in Section 4.7, this orthogonalization gives back the Eigenvectors of $\boldsymbol{A}$ as $k \to \infty$. Therefore, $\widetilde{\boldsymbol{Q}}_k \to \boldsymbol{U}$, which is precisely what we wanted!

6 Parting words

These theoretical ideas and algorithms only lightly graze the surface of what is a very deep and thoroughly studied field at the intersection of linear algebra and computation. If you work with these methods practically, ten years from now you'll still be discovering neat little tricks some mathematician developed 20 years prior, and it'll be useful, and it'll make your day.

Fortunately, many of these results are conveniently summarized for us, the practitioners, as ready-made tools in any number of linear algebra libraries. The popular platform Matlab (or Octave, its open source competitor), is entirely built around linear algebra; its core data structure is the matrix, and it has numerous built in methods for computing SVDs, Eigen-decompositions, QR-factorizations, and the likes. Similarly, in C++, the library `Eigen` is quickly becoming the overwhelming toolkit of choice since it's fast, easy to use, and comprehensive. Python also has a standard library `numpy` that rivals many of Matlab's tools. And beyond that, there's significant interest and effort in developing general purpose tools for leveraging parallel computation both across clusters and for GPUs.

If you're thinking of implementing your own versions of these methods and tools for practical use, consider this: Five years from now, you'll have significantly more experience, and you, yourself, will look back at your own tool, realize how naïve the implementation is (there are always clever tricks with memory allocation, cache hits, avoiding temporaries, etc. you can do to make things faster), and you'll tweak and rework this and that until you have something you're happy with again. But then five years after that... and so on and so forth. Many of these tools are open source. Entire communities of world-class developers and mathematicians with years of practical experience have been iterating that process for, in some cases, decades now. There's very little chance of coming up with something that outperforms those tools unless you're building it on a unique premise (parallelization, FPGAs, quantum computers, whatever (Eigen did it leveraging expression templates)). So embrace linear algebra toolkits.[5] They'll be your friends for years to come.

[5] Although, don't let that stop you from implementing your own if you're just curious and want to have a few hours of fun. Implementing these algorithms forces you to acknowledge and address all the little nuances you might have missed from the abstract theoretical description, and it can be an excellent way to learn.

Appendix

A Distinct Eigenvalues Imply an Eigen-basis

This section explores in more detail why distinct Eigenvalues in a matrix implies the corresponding Eigenvectors form a basis for the domain.

Suppose a square matrix $\boldsymbol{A} \in \mathbb{R}^{n^2}$ has n distinct Eigenvalues $\lambda_1 > \lambda_2 > \cdots > \lambda_n$ with corresponding Eigenvectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$. Then, starting with just the first two Eigen-pairs, suppose we claim that there's some non-zero linear combination of the two that construct the zero vector

$$c_1\boldsymbol{e}_1 + c_2\boldsymbol{e}_2 = \boldsymbol{0} \tag{33}$$

for some $c_1, c_2 \neq 0$. Multiplying that equation by $\boldsymbol{A}$ gives a new equation which is now written in terms of Eigenvalues

$$c_1\boldsymbol{A}\boldsymbol{e}_1 + c_2\boldsymbol{A}\boldsymbol{e}_2 = c_1\lambda_1\boldsymbol{e}_1 + c_2\lambda_2\boldsymbol{e}_2 = \boldsymbol{0}. \tag{34}$$

Multiplying Equation 33 by λ_2 and subtracting it from Equation 34 gives

$$\Big(c_1\lambda_1\boldsymbol{e}_1 + c_2\lambda_2\boldsymbol{e}_2 \Big) - \lambda_2 \Big(c_1\boldsymbol{e}_1 + c_2\boldsymbol{e}_2 \Big) = c_1(\lambda_1 - \lambda_2)\boldsymbol{e}_1 = \boldsymbol{0}.$$

But that means that $\lambda_1 = \lambda_2$, which is wrong. So we know that $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ must be linearly independently.

Now we can apply induction. Suppose that we know that the first k Eigenvectors are linearly independent when we have a full set of distinct Eigenvalues. Consider the next vector $\boldsymbol{e}_{k+1}$. If this new set of vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k, \boldsymbol{e}_{k+1}$ is *not* linearly independent, then it must be that $\boldsymbol{e}_{k+1}$ lies in the linear span of the first k vectors. But if that's the case, we can perform a similar argument as we did above.

Suppose $c_1, \ldots, c_k, c_{k+1}$ are non-zero coefficients such that

$$c_1\boldsymbol{e}_1 + \ldots + c_k\boldsymbol{e}_k + c_{k+1}\boldsymbol{e}_{k+1} = \boldsymbol{0}. \tag{35}$$

Then multiplying this equation by $\boldsymbol{A}$ again gives a similar equation, but now in with Eigenvalues present:

$$\begin{aligned} &\boldsymbol{A}\Big(c_1\boldsymbol{e}_1 + \ldots + c_k\boldsymbol{e}_k + c_{k+1}\boldsymbol{e}_{k+1} \Big) \\ &\quad = c_1\lambda_1\boldsymbol{e}_1 + \ldots + c_k\lambda_k\boldsymbol{e}_k + c_{k+1}\lambda_{k+1}\boldsymbol{e}_{k+1} \\ &\quad = \boldsymbol{0}. \end{aligned} \tag{36}$$

Subtracting λ_{k+1} times Equation 35 from Equation 36 gives

$$\begin{aligned} &\Big(c_1\lambda_1\boldsymbol{e}_1 + \ldots + c_k\lambda_k\boldsymbol{e}_k + c_{k+1}\lambda_{k+1}\boldsymbol{e}_{k+1} \Big) \\ &\qquad - \lambda_{k+1}\Big(c_1\boldsymbol{e}_1 + \ldots + c_k\boldsymbol{e}_k + c_{k+1}\boldsymbol{e}_{k+1} \Big) \\ &= c_1(\lambda_1 - \lambda_{k+1})\boldsymbol{e}_1 + c_2(\lambda_2 - \lambda_{k+1})\boldsymbol{e}_1 + \ldots + c_k(\lambda_k - \lambda_{k+1})\boldsymbol{e}_k \\ &= \boldsymbol{0}. \end{aligned}$$

The $k + 1^{\text{st}}$ term vanishes leaving us with a statement that a non-trivial linear combination of the first k Eigenvectors produces the zero vector. But we've already established that the first k Eigenvectors are linearly independent, so that can't be right. It must be that all $k + 1$ vectors are fully linearly independent.

And by induction, we can conclude that **all n Eigenvectors of a matrix with distinct Eigenvalues must be linearly independent.** Since there are n of them, we can also say that they form a basis for the domain.

References

Sheldon Axler. Down with determinants! *American Mathematical Monthly*, pages 139 – 154, 1995.

Sheldon Axler. *Linear Algebra Done Right*. Springer, 3rd edition, November 2014.

S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:107 – 117, 1998.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

Sadri Hassani. *Mathematical Physics: A Modern Introduction to its Foundations*. Springer, 2nd edition, 2013.

Oliver Knill. Math 19b lecture notes: Linear algebra with probability, markov matrices, 2011a. URL `http://www.math.harvard.edu/~knill/teaching/math19b_2011/handouts/lecture33.pdf`.

Oliver Knill. Math 19b lecture notes: Linear algebra with probability, perron frobenius theorem, 2011b. URL `http://www.math.harvard.edu/~knill/teaching/math19b_2011/handouts/lecture34.pdf`.

Peter J. Olver. Orthogonal bases and the QR algorithm, 2010. URL `http://www.math.umn.edu/~olver/aims_/qr.pdf`. Lecture notes.

Nathan Ratliff. Linear algebra I: Matrix representations of linear transforms, 2014a. Lecture notes.

Nathan Ratliff. Linear algebra II: The fundamental structure of linear transforms, 2014b. Lecture notes.

Gilbert Strang. *Linear Algebra and Its Applications (4rd Ed.)*. Cengage Learning, 4th edition, 2005.

Wikipedia. Perron-Frobenius theorem, 2002-2014. URL `http://en.wikipedia.org/wiki/Perron-Frobenius_theorem`.