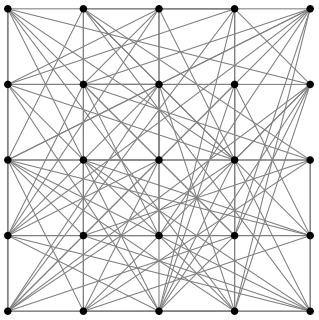
You have collected data about some individuals and their mutual friendships. This is the canonical example of a graph. Each person is a vertex, and edges represent friends. To begin making sense of this data it helps to visualize it first. There is an infinite number of ways to visualize a graph. If we line up the vertices in a grid this is what the data looks like:



What might we want to know about this graph? Societies tend to cluster into groups: Within each group there are many friendships, and among groups there are substantially fewer. Can we identify these groups from the data? They do not stand out in the grid. The reason is that grid embeddings are a poor representation for pairwise relationships like friendships. Here is a much better embedding:

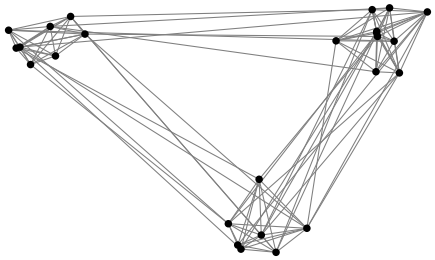
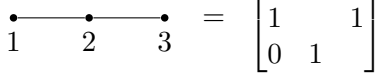


FIGURE 1: Clustering in a "friendship graph".

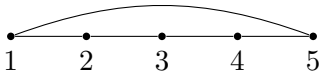
The three clusters now stand out visually. It is apparent from the picture that there are many more friendships within each group than there are between groups. The method that led to this representation is called *spectral decomposition*. Geometric representations of data are useful not only for human visualization but also for computer processing.

1 The geometry of data

A graph is nothing more and nothing less than a symmetric square matrix with zero-one entries (the adjacency matrix). The rows and columns of this matrix are indexed by vertices and the entries indicate edges. Here are the 2-path and the 5-cycle in matrix form:



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



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

The diagonal entries are left unspecified. We will fill them in later. Every symmetric matrix, including those of graphs, can be realized geometrically as a dot product of vectors. Namely, we can associate a

vector to every vertex so that entries in the matrix are the dot product of the corresponding vectors: edges are represented by pairs of vectors with dot product one (vectors that are "close") while non-edges are represented by *orthogonal* pairs (vectors that are "far"). The geometric realizations of the 2-path and the 5-cycle are shown in Figure 2.

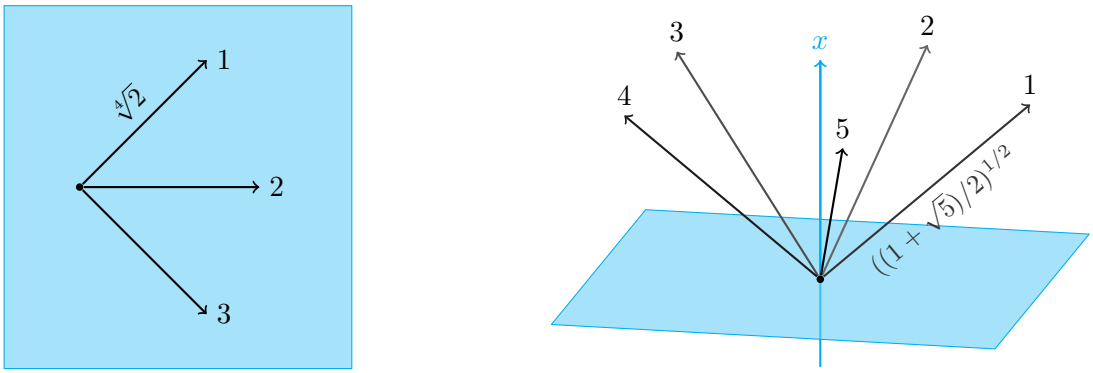


FIGURE 2: Geometric realizations of (a) the 2-path in the plane and (b) the 5-cycle in three dimensions. Non-edges are represented by orthogonal pairs of vectors. Pairs corresponding to edges have dot product one.

The 2-path has a two-dimensional geometric realization by dot products. In contrast, the 5-cycle doesn't. To obtain a faithful representation we have to enter the third dimension.

In general, any n-vertex graph has a geometric realization in at most n-dimensional space. For the purpose of visualization and data processing, lower-dimensional embeddings are preferable. Unfortunately, many interesting graphs and matrices require all or almost all n dimensions to embed. To what extent can these perfect embeddings be approximated in one, two, or a small number of dimensions?

To answer this question we need to formulate it in the language of algebra. Geometry is useful for visualizing data and algorithms, but algebra is their "programming language". What is the algebra behind geometric realizations of graphs?

2 Eigenvalues and eigenvectors

An *eigenvector* $\mathbf{v}$ of square matrix S is a nonzero vector that, when multiplied by S, becomes a constant multiple of itself: $S\mathbf{v} = \lambda\mathbf{v}$, with λ being its corresponding eigenvalue. This equation is invariant under scaling so we can insist that $\mathbf{v}$ should have unit length.

If S is symmetric then left and right multiplication produce the same result: $S\mathbf{v}$ (as a column vector) equals $\mathbf{v}S$ (as a row vector). An important consequence is that eigenvectors associated to different eigenvalues are orthogonal: If $S\mathbf{v} = \lambda\mathbf{v}$ and $S\mathbf{w} = \mu\mathbf{w}$, then

$$\lambda\mathbf{v}\cdot\mathbf{w} = \mathbf{v}S\cdot\mathbf{w} = \mathbf{v}\cdot S\mathbf{w} = \mathbf{v}\cdot\mu\mathbf{w}$$

so unless $\lambda = \mu$, $\mathbf{v}\cdot\mathbf{w}$ must be zero: They are orthogonal. If λ does equal μ then every linear combination of $\mathbf{v}$ and $\mathbf{w}$ is also an eigenvector with the same eigenvalue:

$$S(a\mathbf{v} + b\mathbf{w}) = aS\mathbf{v} + bS\mathbf{w} = a\lambda\mathbf{v} + b\lambda\mathbf{w} = \lambda(a\mathbf{v} + b\mathbf{w}).$$

Taken together, this tells us that the eigenvectors of a symmetric matrix break up into mutually orthogonal *subspaces*, with each subspace labeled by a distinct eigenvalue.

This collection of subspaces with their corresponding eigenvalues is called the *spectrum* of S. When all the subspaces are one-dimensional, the spectrum can be described as a collection of eigenvector-eigenvalue pairs—uniquely up to the signs of the eigenvectors.

For example, the spectrum of the 2-path P_2 (assuming for now that there are zeros on the diagonal) is

$$\lambda_1 = \sqrt{2}, \mathbf{v}_1 = \begin{bmatrix} 1/2 \\ 1/\sqrt{2} \\ 1/2 \end{bmatrix} \qquad \lambda_2 = 0, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{bmatrix} \qquad \lambda_3 = -\sqrt{2}, \mathbf{v}_3 = \begin{bmatrix} 1/2 \\ -1/\sqrt{2} \\ 1/2 \end{bmatrix}. \tag{1}$$

In this example there are three orthogonal eigenvectors, exactly as many as there are dimensions. This is not a coincidence:

Theorem 1. *Every $n \times n$ symmetric matrix S has an orthonormal basis of eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$.*

This theorem is very important for the following reason. Every vector $\mathbf{x}$ can be written as a linear combination of $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$, for example

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \left(1 + \frac{1}{\sqrt{2}}\right)\mathbf{v}_1 + 0\mathbf{v}_2 + \left(1 - \frac{1}{\sqrt{2}}\right)\mathbf{v}_3.$$

The coefficients of $\mathbf{x} = (1, 1, 1)$ are nothing but the dot products $\mathbf{x}\cdot\mathbf{v}_1 = 1+1/\sqrt{2}, \mathbf{x}\cdot\mathbf{v}_2 = 0, \mathbf{x}\cdot\mathbf{v}_3 = 1-1/\sqrt{2}$. In general,

$$\mathbf{x} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \cdots + \alpha_n\mathbf{v}_n, \qquad \text{where } \alpha_1 = \mathbf{x}\cdot\mathbf{v}_1, \alpha_2 = \mathbf{x}\cdot\mathbf{v}_2, \ldots.$$

Applying S to $\mathbf{x}$ now amounts to scaling down each coefficient by the corresponding eigenvalue:

$$S\mathbf{x} = \alpha_1\lambda_1\mathbf{v}_1 + \alpha_2\lambda_2\mathbf{v}_2 + \cdots + \alpha_n\lambda_n\mathbf{v}_n. \tag{2}$$

For instance:

$$P_2 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = (1 + 1/\sqrt{2})\lambda_1\mathbf{v}_1 + 0 \cdot \lambda_2\mathbf{v}_2 + (1 - 1/\sqrt{2})\lambda_3\mathbf{v}_3,$$

which equals $(1, 2, 1)$ as it should. The effect of applying a matrix in the eigenvector basis is simple to describe: Each coefficient is scaled by the corresponding eigenvalue.

A geometric interpretation of (2) is that if we shift our "perspective" from the standard basis to the eigenvector basis, then S looks like a diagonal matrix: It affects all basis directions independently, and each is scaled by the corresponding eigenvalue. A shift between orthonormal bases is nothing but a high-dimensional "rotation." Geometric properties of vectors like norm, distance, and dot product are invariant under such transformations.

This change of perspective illuminates the effects of the matrix-vector multiplication $\mathbf{x} \to S\mathbf{x}$. In the eigenvector basis, this transformation scales the first coordinate by λ_1, the second one by λ_2, and so on. Therefore the maximum amount by which a vector $\mathbf{x}$ can grow is precisely equal to the largest eigenvalue—in absolute value. This is the spectral norm $\|S\| = \max_{\mathbf{x}\neq 0}\|S\mathbf{x}\|/\|\mathbf{x}\|$ we saw last week:

Theorem 2. *The spectral norm of a symmertic matrix S equals the largest eigenvalue in absolute value.*

For example, the spectral norm of P_2 (with zeros on the diagonal) is $\sqrt{2}$. It is attained by $\mathbf{x} = \mathbf{v}_1$, and also by $\mathbf{v}_3$.

Theorem 2 tells us that to calculate the spectral norm of a matrix all we need to do is figure out its eigenvalue of largest magnitude. How can we do that?

3 Power iteration

Algorithm *PI* (Power iteration)
Input: A symmetric matrix S.
1 Choose an initial vector $\mathbf{x}$ (of unit length).
2 Until two successive values of $\mathbf{x}$ are sufficiently close,
3 Replace $\mathbf{x}$ by $S\mathbf{x}$.
4 Normalize $\mathbf{x}$ to unit length.
5 Output the last $\mathbf{x}$ as the eigenvector and the last normalization factor as the spectral norm.

Here is a sample run on the algorithm on input $\begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix}$ with initial vector $\mathbf{x} = (1, 0)$:

step	0	1	2	3	4	5	6
$\mathbf{x}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}.447\\.894\end{bmatrix}$	$\begin{bmatrix}.781\\.625\end{bmatrix}$	$\begin{bmatrix}.680\\.733\end{bmatrix}$	$\begin{bmatrix}.704\\.710\end{bmatrix}$	$\begin{bmatrix}.708\\.706\end{bmatrix}$	$\begin{bmatrix}.707\\.707\end{bmatrix}$
$\lvert\lambda\rvert$		2.236	2.864	2.984	2.998	3.000	3.000

The run quickly converges to $|\lambda| = 3, \mathbf{x} = (1/\sqrt{2}, 1/\sqrt{2})$. Indeed, $\lambda_1 = 3$ is the dominant eigenvalue, and $\mathbf{v}_1 = (1/\sqrt{2}, 1/\sqrt{2})$ is the corresponding eigenvector. The other pair is $\lambda_2 = -1$ and $\mathbf{v}_2 = (-1/\sqrt{2}, 1/\sqrt{2})$.

The reason power iteration works is formula (2). Initially, $\mathbf{x}$ has the form $\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2$. After one iteration it becomes $\alpha_1\lambda_1\mathbf{v}_1 + \alpha_2\lambda_2\mathbf{v}_2$. After two iterations direction $\mathbf{v}_i$ is scaled by λ_i again, so $\mathbf{x}$ becomes $\alpha_1\lambda_1^2\mathbf{v}_1 + \alpha_2\lambda_2^2\mathbf{v}_2$ normalized to unit length. After t iterations, $\mathbf{x}$ is equal to

$$\mathbf{x}_t = \alpha_1\lambda_1^t\mathbf{v}_1 + \alpha_2\lambda_2^t\mathbf{v}_2 \quad \text{normalized to unit length.}$$

The power of exponentials soon kicks in: As $\lambda_1^7 = 3^7 = 2187$ is much bigger than $\lambda_2^7 = 1^7 = 1$, the $\mathbf{v}_1$-part of $\mathbf{x}_7$ overwhelms its $\mathbf{v}_2$-part, provided that α_1 didn't happen to be zero, or vanishingly small relative to α_2. See Figure 3.

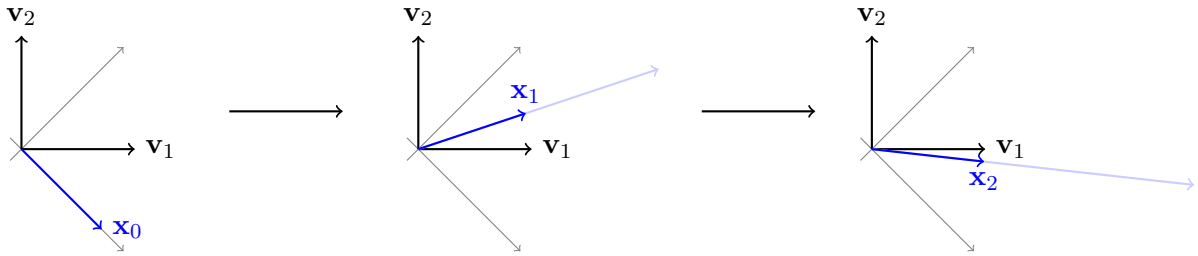


FIGURE 3: The first two power iterations on $\mathbf{x}$. In each step the components of $\mathbf{x}$ in the direction of eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$ are scaled by the respective eigenvalues $\lambda_1 = 3, \lambda_2 = -1$.

The validity of the last assumption depends on our initial choice of $\mathbf{x}$. If we were so unlucky that this initial $\mathbf{x}$ had completely aligned with $\mathbf{v}_2$, all the power iterations would have kept $\mathbf{x}$ orthogonal to $\mathbf{v}_1$ and prevented convergence to it. If there is some randomness involved in the choice of the initial $\mathbf{x}$ (for example if it was sampled as a standard multivariate Gaussian) this possibility becomes unlikely.

In general, power iteration converges rapidly to a dominant eigenvector provided the matrix S exhibits a gap in the magnitudes of its largest and its second largest eigenvalues. If $\lambda_1, \mathbf{v}_1$ is the dominant eigenvalue-eigenvector pair, after t steps the projection of $\mathbf{x}$ on $\mathbf{v}_1$ equals

$$\mathbf{x}_t \cdot \mathbf{v}_1 = \frac{1}{\sqrt{1 + (\alpha_2/\alpha_1)^2 \cdot (\lambda_2/\lambda_1)^{2t} + \cdots + (\alpha_n/\alpha_1)^2 \cdot (\lambda_n/\lambda_1)^{2t}}}, \tag{3}$$

and $\mathbf{x}$ approaches $\mathbf{v}_1$ at an exponential rate with base $\max_{i\neq 1}|\lambda_i|/|\lambda_1|$. This quantity is called the *spectral gap* of S. (More generally, if the highest eigenvalue λ_1 labels a multi-dimensional subspace of eigenvectors,

$\mathbf{x}$ will eventually align with some eigenvector in this subspace, with the choice of eigenvector depending on the initialization.)

The spectral norm estimate, on the other hand, does not depend on the spectral gap. It approaches the spectral norm at a rate of at least $\alpha_1^{1/t}$.

Theorem 3. *Assuming λ_1 is the largest eigenvalue in absolute value, the state $\mathbf{x}_t$ of Power Iteration at time t initialized with $\mathbf{x}$ satisfies $\|S\mathbf{x}_t\|/\|\mathbf{x}_t\| \geq |\alpha_1|^{1/t}|\lambda_1|$, where $\alpha_1 = \mathbf{x} \cdot \mathbf{v}_1$.*

The matrix P_2 (with zeros on the diagonal) fails to exhibit a spectral gap. Its eigenvalues λ_1 and λ_3 are opposite in sign but equal in magnitude. Here is what happens when we run power iteration on it with initialization $(0.6, 0.8, 0)$:

step	0	1	2	3	4
$\mathbf{x}$	$\begin{bmatrix}.600\\.800\\0\end{bmatrix}$	$\begin{bmatrix}.625\\.469\\.625\end{bmatrix}$	$\begin{bmatrix}.331\\.883\\.331\end{bmatrix}$	$\begin{bmatrix}.625\\.469\\.625\end{bmatrix}$	$\begin{bmatrix}.331\\.883\\.331\end{bmatrix}$
$\lvert\lambda\rvert$		1.281	1.414	1.414	1.414

The value of $|\lambda|$ quickly converges to the spectral norm $\|S\| = \sqrt{2}$, but $\mathbf{x}$ keeps alternating between two possibilities, neither of which is an eigenvector of S. The reason is that the initial $\mathbf{x}$ contains a mixture of $\mathbf{v}_1$ and $\mathbf{v}_3$. As the corresponding eigenvalues have the same magnitude but different sign, $\mathbf{x}$ keeps alternating between the directions $\alpha_1\mathbf{v}_1 + \alpha_3\mathbf{v}_3$ and $\alpha_1\mathbf{v}_1 - \alpha_3\mathbf{v}_3$.

These alternations can be avoided by a simple but useful trick: Shift the eigenvalues of S so that they all become nonnegative. This can be accomplished by shifting all diagonal entries of S by the same amount. To prevent negativity, the shift should be at least as large as the smallest (negative) eigenvalue of S.

In the case of P_2, its smallest eigenvalue is $-\sqrt{2}$. Shifting all diagonal entries by $\sqrt{2}$ produces the matrix

$$P_2' = \begin{bmatrix}\sqrt{2} & 1 & 0\\ 1 & \sqrt{2} & 1\\ 0 & 1 & \sqrt{2}\end{bmatrix}$$

Its spectrum is derived from that of P_2 (1) with all its eigenvalues shifted by $\sqrt{2}$:

$$\lambda_1 = 2\sqrt{2}, \mathbf{v}_1 = \begin{bmatrix}1/2\\1/\sqrt{2}\\1/2\end{bmatrix} \qquad \lambda_2 = \sqrt{2}, \mathbf{v}_2 = \begin{bmatrix}1/\sqrt{2}\\0\\-1/\sqrt{2}\end{bmatrix} \qquad \lambda_3 = 0, \mathbf{v}_3 = \begin{bmatrix}1/2\\-1/\sqrt{2}\\1/2\end{bmatrix}.$$

Symmetric matrices with nonnegative eigenvalues like P_2' are very useful for data analysis and many other things. We already saw them in Lecture 2 under a different name.

4 Spectral decomposition

Formula (2) was incredibly useful for understanding how a symmetric matrix S acts on vectors $\mathbf{x}$. The same formula gives an alternative representation of S itself: its spectral decomposition.

A symmetric matrix is nothing more than a diagonal matrix when viewed in the basis of its eigenvectors. Here is an example decomposition of a diagonal matrix into very simple components:

$$\begin{bmatrix}3 & 0\\0 & -1\end{bmatrix} = 3 \cdot \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix} + (-1) \cdot \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix} = 3 \cdot \begin{bmatrix}1\\0\end{bmatrix} \cdot \begin{bmatrix}1 & 0\end{bmatrix} + (-1) \cdot \begin{bmatrix}0\\1\end{bmatrix} \cdot \begin{bmatrix}0 & 1\end{bmatrix}.$$

In general, a diagonal matrix D with entries $\lambda_1, \ldots, \lambda_n$ can be decomposed as

$$D = \lambda_1 \cdot \mathbf{e}_1 \otimes \mathbf{e}_1 + \cdots + \lambda_n \cdot \mathbf{e}_n \otimes \mathbf{e}_n, \tag{4}$$

where $\mathbf{e}_i$ is the i-th elementary unit vector—which is the eigenvector of D associated to λ_i, and $\mathbf{x} \otimes \mathbf{y}$ is *outer product*—the matrix obtained by multiplying column vector $\mathbf{x}$ by row vector $\mathbf{y}$.

Formula (4) is the spectral decomposition of a diagonal matrix. The same decomposition works for an arbitrary symmetric matrix when $\mathbf{e}_i$ is replaced by the corresponding eigenvector $\mathbf{v}_i$.

Spectral decomposition of a symmetric matrix:

$$S = \lambda_1 \cdot \mathbf{v}_1 \otimes \mathbf{v}_1 + \cdots + \lambda_n \cdot \mathbf{v}_n \otimes \mathbf{v}_n. \tag{5}$$

You can verify that this formula is valid on all the examples so far, for instance

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = 3 \cdot \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} + (-1) \cdot \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}.$$

Something miraculous happens when the negative eigenvalues disappear. For example, the spectral decomposition of P_2' is

$$\lambda_1 \cdot \mathbf{v}_1 \otimes \mathbf{v}_1 + \lambda_2 \cdot \mathbf{v}_2 \otimes \mathbf{v}_2 = 2\sqrt{2} \cdot \begin{bmatrix} 1/2 \\ 1/\sqrt{2} \\ 1/2 \end{bmatrix} \begin{bmatrix} 1/2 & 1/\sqrt{2} & 1/2 \end{bmatrix} + \sqrt{2} \cdot \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} \end{bmatrix}.$$

The $\lambda_3 \cdot \mathbf{v}_3 \otimes \mathbf{v}_3$ part is omitted because λ_3 equals zero. The eigenvalues can now be split equally among the row and column vector. The first term can be refactored as

$$2\sqrt{2} \cdot \begin{bmatrix} 1/2 \\ 1/\sqrt{2} \\ 1/2 \end{bmatrix} \begin{bmatrix} 1/2 & 1/\sqrt{2} & 1/2 \end{bmatrix} = 2^{3/4} \begin{bmatrix} 1/2 \\ 1/\sqrt{2} \\ 1/2 \end{bmatrix} \cdot 2^{3/4} \begin{bmatrix} 1/2 & 1/\sqrt{2} & 1/2 \end{bmatrix} = \begin{bmatrix} 2^{-1/4} \\ 2^{1/4} \\ 2^{-1/4} \end{bmatrix} \cdot \begin{bmatrix} 2^{-1/4} & 2^{1/4} & 2^{-1/4} \end{bmatrix}.$$

and similarly, the second one as the outer product of $(2^{-1/4}, 0, -2^{-1/4})$ with itself. As a whole, P_2' is the sum of these two terms, which can be grouped as a product $A^T A$ of a matrix and its transpose:

$$\begin{aligned} P_2' &= \begin{bmatrix} 2^{-1/4} \\ 2^{1/4} \\ 2^{-1/4} \end{bmatrix} \cdot \begin{bmatrix} 2^{-1/4} & 2^{1/4} & 2^{-1/4} \end{bmatrix} + \begin{bmatrix} 2^{-1/4} \\ 0 \\ -2^{-1/4} \end{bmatrix} \cdot \begin{bmatrix} 2^{-1/4} & 0 & -2^{-1/4} \end{bmatrix} \\ &= \begin{bmatrix} 2^{-1/4} & 2^{-1/4} \\ 2^{1/4} & 0 \\ 2^{-1/4} & -2^{-1/4} \end{bmatrix} \cdot \begin{bmatrix} 2^{-1/4} & 2^{1/4} & 2^{-1/4} \\ 2^{-1/4} & 0 & -2^{-1/4} \end{bmatrix}. \end{aligned}$$

The columns of A are precisely the vectors in the geometric realization P_2 from Figure 2. Here is an important general consequence of these manipulations:

Theorem 4. *S is symmetric with nonnegative eigenvalues if and only if S equals $A^T A$ for some A.*

Matrices that have one of these equivalent properties are called positive semidefinite (PSD).

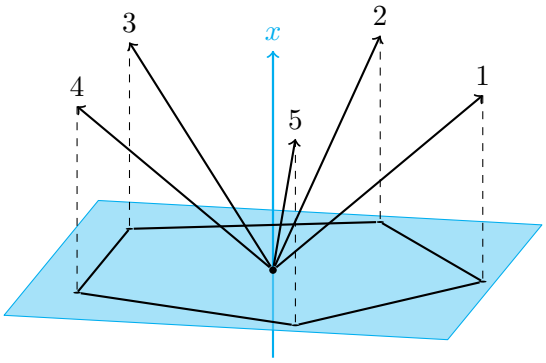
Proof. In the only if direction, starting with the spectral decomposition of S, we define A to be the matrix with *rows* $\sqrt{\lambda_1}\mathbf{v}_1, \sqrt{\lambda_2}\mathbf{v}_2, \cdots, \sqrt{\lambda_n}\mathbf{v}_n$. Then $A^T A$ is the sum of outer products of the rows of S, which is the spectral decomposition of S.

In the if direction, every matrix S of the form $A^T A$ is symmetric because its (i, j)-th entry is the dot product of the ith and jth columns of A, which doesn't change if we swap i and j. If λ is an eigenvalue $A^T A$ with eigenvector $\mathbf{v}$, then $\mathbf{v}^T A^T A \mathbf{v} = \mathbf{v}^T \lambda \mathbf{v} = \lambda \|\mathbf{v}\|^2$. On the other hand, $\mathbf{v}^T A^T A \mathbf{v}$ is the dot product of $A\mathbf{v}$ with itself, so $\mathbf{v}^T A^T A \mathbf{v} = \|A\mathbf{v}\|^2$. So $\lambda = \|A\mathbf{v}\|^2 / \|\mathbf{v}\|^2$, which cannot be negative as it is a square. □

The geometric realization of S can be read off from the *columns* of the matrix A. The entries of S are precisely the dot products of the columns of A. Thus geometric realizations can be directly derived from the spectral decomposition.

Just like we did for P_2, we can derive the geometric realization of the cycle C_5 in Figure 2 (b) in the same manner. The eigenvalues of C_5 with zeros on the diagonal happen to be 2 once, $(\sqrt{5}-1)/2 \approx 0.618$ twice, and $-(\sqrt{5}+1)/2 \approx -1.618$ twice. The eigenvector corresponding to 2 is (a scaling of the) all-ones vector. Setting the diagonal entries to $(\sqrt{5}+1)/2$ zeroes out the two copies of the smallest eigenvalue, giving a representation $C_5' = A^T A$ for a matrix A with three rows and five columns. The geometric realization given by the columns of A is thus a three-dimensional embedding.

The top row of A is a scaling of the all-ones vector. All its entries are equal. Thus all vectors in the embedding have the same x-component. This happens whenever all vertices have the same degree, namely the graph is *regular*. As this part of the representation is not informative it is sensible to project it out. What remains is the natural embedding of the 5-cycle in the plane:



5 Approximate embeddings

In the spectral decomposition of a generic graph G we wouldn't expect significant coincidences among its eigenvalues. The resulting geometric realization would therefore live in close to n dimensions. High-dimensional representations are not very useful for data analysis. It is more convenient to pick a low-dimensional one even if we have to settle for some imperfections. How should we go about this?

Picking the "best possible" low-dimensional representation L of PSD matrix S is an optimization problem. Once we fix the dimension d, two natural quantities we could try to minimize are the spectral norm $\|S - L\|$, or the sum of the squares of the entries of $S - L$. For both of these losses, the truncated spectral representation is the answer!

Theorem 5. *Assume S is PSD with spectral decomposition (5) and $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$. Among all d by n matrices A, the spectral norm $\|S - A^T A\|$ and the sum of the squares of the entries of $S - A^T A$ are both minimized when A is the matrix with rows $\sqrt{\lambda_1}\mathbf{v}_1, \ldots, \sqrt{\lambda_d}\mathbf{v}_d$.*

Theorem 5 tells us that each successive term in the expansion (5) provides a finer approximation of S. The first term $\lambda_1 \mathbf{v}_1 \otimes \mathbf{v}_1$ gives crude information about S. For example, when S is a graph, $\mathbf{v}_1$ measures how "important" each vertex is, but doesn't say much about how different vertices relate to one another. When all vertices have the same degree as in a regular graph, we need to look at the next term $\lambda_2 \mathbf{v}_2 \otimes \mathbf{v}_2$ to obtain information about their relative proximity. If we are not satisfied with this approximation, we can then add in $\lambda_3 \mathbf{v}_3 \otimes \mathbf{v}_3$ and so on. The price we have to pay for higher-quality approximations is that the matrix becomes harder to visualize and process algorithmically.

In particular, the "best" three-dimensional embedding of a graph G is given by the columns of the matrix A with rows $\sqrt{\lambda_1}\mathbf{v}_1, \sqrt{\lambda_2}\mathbf{v}_2, \sqrt{\lambda_3}\mathbf{v}_3$. If G is (almost) regular, $\mathbf{v}_1$ won't be informative as all of its entries are (almost) the same. As we did for the 5-cycle, it is sensible to drop the first row and visualize the remaining two-dimensional embedding. This is how the clustering in Figure 1 was produced.

For automated data processing, even one-dimensional embeddings can be useful. If we drop the y-coordinate in Figure 1, the position of each vertex is determined by its value in the eigenvector $\mathbf{v}_2$ associated to the second largest eigenvalue:

$$\text{position of vertex } i \text{ on the line} = i\text{-th entry in } \mathbf{v}_2.$$

The three clusters no longer stand out but it is still easy to visually separate the one on the left from the two on the right. One algorithmic approach for finding the division is to try all possible *cuts* that split the vertices into *L*eft and *R*ight parts and choose the one that minimizes some loss.

A loss measure that stands out is *cut density*. This is the number of edges crossing between L and R divided by the number of left vertices times the number of right vertices. As there are only $n+1$ cuts to consider this can be algorithmically tractable if n is not too large. The resulting cut may not be the best possible among all 2^{n-1} possibilities, but Cheeger's inequality guarantees that it cannot deviate too far from it.[1]

6 Calculating the spectral decomposition

If you took a linear algebra class, chances are that you were taught to calculate spectral decompositions like this: First, set up the characteristic equation $\det(S - \lambda I) = 0$, where det is the determinant and I is the identity matrix. This is a polynomial equation of degree n in λ, so it has n solutions counting multiplicities. Solve for λ to find the eigenvalues $\lambda_1, \ldots, \lambda_n$. Then solve

$$S\mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{6}$$

by Gaussian elimination for every i to recover all corresponding eigenvectors.

This textbook algorithm works well on 2×2 matrices and some special classes of larger matrices. The main difficulty for it is that eigenvalues of general matrices are not algebraic. They cannot be calculated using elementary arithmetic operations. We must resort to numerical approximations. If we do not know λ_i exactly, however, system (6) has no nonzero solution and cannot be handled by Gaussian elimination. We could try to look for approximate solutions, but overall this method is slow and sensitive to the buildup of errors in the process.

Power iteration provides an alternative. It finds an excellent approximation of the largest in magnitude eigenvalue λ and its associated eigenvector $\mathbf{v}$. We know that the other eigenvectors are orthogonal to $\mathbf{v}$, and that applying S to them preserves orthogonality to $\mathbf{v}$. Once we know $\mathbf{v}$, we can initialize power iteration with any $\mathbf{x}$ that is orthogonal to $\mathbf{v}$. We might expect that the output of power iteration should then converge to a new eigenvalue-eigenvector pair $\lambda', \mathbf{v}'$. Does it?

I tested this strategy on matrix

$$\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}. \tag{7}$$

[1] Cheeger's inequality applies to the Laplacian matrix representation of graphs. The Laplacian representation coincides with our adjacency matrix representation when the graph is regular.

The first power iteration, initialized with $(1, 0)$, produced the pair $\lambda = 3.618$, $\mathbf{v} \approx (.526, .851)$. I initialized the second power iteration with $\mathbf{x} = (-.851, .526)$ which is orthogonal to $\mathbf{v}$. This one took longer to settle, but after 20 steps it stabilized around 3.618 and $(.526, .851)$ again. What happened?

The strategy would have worked flawlessly if $\mathbf{x}$ was perfectly orthogonal to $\mathbf{v}$. Perfection, however is impossible to enforce because $\mathbf{v}$ is unavailable to us in infinite precision. It is therefore quite likely that $\mathbf{x}$ retains some small component in the direction of $\mathbf{v}$. This component will be amplified by the power iteration eventually resulting in convergence to $\mathbf{v}$ again! It appears that we have not made any progress.

There is an effective solution to this problem: We should enforce orthogonality between $\mathbf{x}$ and $\mathbf{v}$ at every step of power iteration, not only at initialization.

Orthogonalization

Let's abstract the question. We are given a vector $\mathbf{v}$ of unit norm, and another vector $\mathbf{x}$ that is almost orthogonal to it. How can we make it exactly orthogonal? This is high school geometry: We need to project $\mathbf{x}$ onto $\mathbf{v}$ and subtract the projection from $\mathbf{x}$:

$$\mathbf{x}' = \mathbf{x} - (\mathbf{x} \cdot \mathbf{v})\mathbf{v}.$$

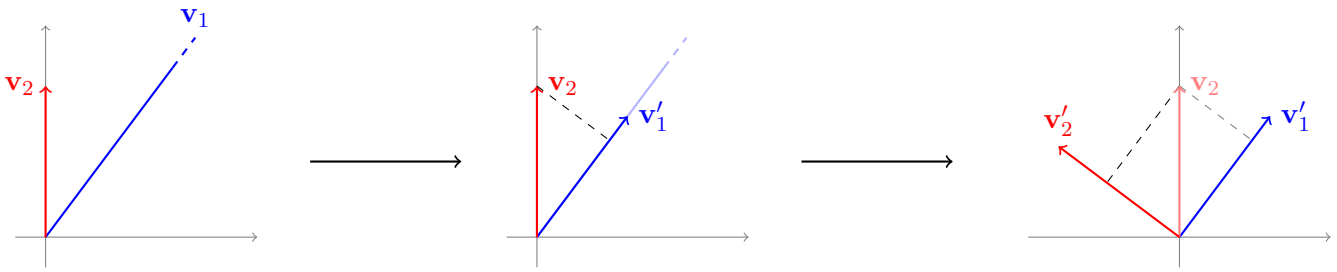
This $\mathbf{x}'$ is indeed orthogonal to $\mathbf{v}$: If we take the dot product of both sides by $\mathbf{v}$ we get

$$\mathbf{x}' \cdot \mathbf{v} = \mathbf{x} \cdot \mathbf{v} - (\mathbf{x} \cdot \mathbf{v})(\mathbf{v} \cdot \mathbf{v}) = \mathbf{x} \cdot \mathbf{v} - (\mathbf{x} \cdot \mathbf{v})1 = 0.$$

Gram-Schmidt orthogonalization is a generalization of this procedure to many dimensions. Its input is a collection of vectors, say $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$. It transforms this collection into orthonormal $\mathbf{v}'_1, \mathbf{v}'_2, \mathbf{v}'_3$ such that $\mathbf{v}'_1$ is parallel to $\mathbf{v}_1$, $\mathbf{v}'_2$ is spanned by $\mathbf{v}_1$ and $\mathbf{v}_2$, and $\mathbf{v}'_3$ is spanned by $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$.

Algorithm *GS* (Gram-Schmidt Orthogonalization)
Input: Vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$.
1 For i ranging from 1 to n:
2 Replace $\mathbf{v}_i$ with $\mathbf{v}_i - (\mathbf{v}_i \cdot \mathbf{v}_1)\mathbf{v}_1 - (\mathbf{v}_i \cdot \mathbf{v}_2)\mathbf{v}_2 - \cdots - (\mathbf{v}_i \cdot \mathbf{v}_{i-1})\mathbf{v}_{i-1}$.
3 Normalize $\mathbf{v}_i$ to unit length.

For example, running *GS* on input $\mathbf{v}_1 = (3, 4), \mathbf{v}_2 = (0, 1)$ first normalizes $\mathbf{v}_1$ to $\mathbf{v}'_1 = (.6, .8)$, then projects $\mathbf{v}_2$ onto $\mathbf{v}'_1$ to obtain $(0, 1) - .8 \cdot (.6, .8) = (-.48, .36)$, and then normalizes this vector by $\sqrt{.48^2 + .36^2} = .6$ to $\mathbf{v}'_2 = (-.8, .6)$.



The steps can be reversed to obtain a formula for $\mathbf{v}_1$ and $\mathbf{v}_2$ in terms of $\mathbf{v}'_1$ and $\mathbf{v}'_2$: $\mathbf{v}_1 = 5\mathbf{v}'_1$ and $\mathbf{v}_2 = .8\mathbf{v}'_1 + .6\mathbf{v}'_2$. In matrix form this says:

$$\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix} = \begin{bmatrix}\mathbf{v}'_1 & \mathbf{v}'_2\end{bmatrix} \cdot \begin{bmatrix}5 & .8 \\ 0 & .6\end{bmatrix} \qquad \text{or} \qquad \begin{bmatrix}3 & 0 \\ 4 & 1\end{bmatrix} = \begin{bmatrix}.6 & -.8 \\ .8 & -.6\end{bmatrix} \cdot \begin{bmatrix}5 & .8 \\ 0 & .6\end{bmatrix}.$$

This is the *QR decomposition* of the matrix with columns $\mathbf{v}_1$ up to $\mathbf{v}_n$: It is the product of the orthogonal matrix Q with columns $\mathbf{v}'_1$ up to $\mathbf{v}'_n$ times an upper triangular matrix R that describes the conversion between the two sets of vectors.

Subspace iteration and the QR algorithm

Armed with orthogonalization we can now find all the eigenvalues of a symmetric matrix by interleaving power iteration and orthogonalization steps.

Algorithm *SI* (Subspace iteration)
Input: A $n \times n$ symmetric matrix S.
1 Choose an initial orthonormal basis $\mathbf{x}_1, \cdots, \mathbf{x}_n$.
2 Until $\mathbf{x}_1, \cdots, \mathbf{x}_n$ stabilize,
3 Replace $\mathbf{x}_i$ by $S\mathbf{x}_i$ for all i.
4 Orthogonalize $\mathbf{x}_1, \cdots, \mathbf{x}_n$ using the GS algorithm.

The initial basis can be the standard one $\mathbf{e}_1, \ldots, \mathbf{e}_n$. Here is how the column vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ evolve throughout the execution on input (7):

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \xrightarrow{1} \begin{bmatrix} -.894 & -.447 \\ -.447 & .894 \end{bmatrix} \xrightarrow{2} \begin{bmatrix} -.707 & -.707 \\ -.707 & .707 \end{bmatrix} \xrightarrow{3} \begin{bmatrix} -.6 & -.8 \\ -.8 & -.6 \end{bmatrix} \xrightarrow{4} \begin{bmatrix} -.555 & -.832 \\ -.832 & .555 \end{bmatrix} \cdots \xrightarrow{7} \begin{bmatrix} -.526 & -.851 \\ -.851 & .526 \end{bmatrix}.$$

The iteration approximates the true eigenvectors up to three decimals in step 7. The eigenvalues can be recovered by calculating the ratios $\|S\mathbf{v}_i\|/\|\mathbf{v}_i\|$. In the example these become $\lambda_1 \approx 3.618$ and $\lambda_2 \approx 1.382$ after 10 steps.

The SI algorithm can be sped up if we are only interested in the top k eigenvectors: the variables $\mathbf{x}_{k+1}$ up to $\mathbf{x}_n$ can simply be dropped, saving memory, the complexity of each iteration, and convergence time.

If we are interested in the eigenvalues only, there is a slick variant of subspace iteration called QR iteration. QR iteration does not track $\mathbf{x}_1, \cdots, \mathbf{x}_n$ explicitly, but only the representation of S in this basis. By an algebraic miracle that I don't completely understand (project: compare the SI and QR algorithms), every update of Subspace Iteration has the following equivalent effect on S:

Algorithm *QR* (QR iteration)
Input: A $n \times n$ symmetric matrix S.
1 Until the diagonal entries of S stabilize,
2 Calculate the QR decomposition of S.
3 Replace S with RQ.

Here is a sample run of QR iteration on the same input:

$$\begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix} \xrightarrow{1} \begin{bmatrix} 3.5 & .5 \\ .5 & 1.5 \end{bmatrix} \xrightarrow{2} \begin{bmatrix} 3.6 & -.2 \\ -.2 & 1.4 \end{bmatrix} \xrightarrow{3} \begin{bmatrix} 3.615 & .077 \\ .077 & 1.385 \end{bmatrix} \xrightarrow{4} \begin{bmatrix} 3.618 & -.029 \\ -.029 & 1.382 \end{bmatrix} \xrightarrow{5} \begin{bmatrix} 3.618 & .011 \\ .011 & 1.382 \end{bmatrix}$$

The diagonal closes in on the eigenvalues to three digits of precision in iteration 4. The off-diagonal entries approach zero, as they should when S is viewed from the basis of its eigenvectors.

Theorem 6. *Let $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and S be the states of SI and QR after t iterations, respectively. If both are run on input S_0 and SI is initialized with the standard basis, then $S = X^T S_0 X$.*

This theorem explains why if Subspace Iteration is correct, so must be QR iteration. After many iterations of SI, X approaches the matrix of eigenvectors $(\mathbf{v}_1, \ldots, \mathbf{v}_n)$. The effect S_0 on X is to scale each column by its corresponding eigenvalue, so $S_0 X$ approaches $(\lambda_1 \mathbf{v}_1, \ldots, \lambda_n \mathbf{v}_n)$. Left multiplication by X^T, i.e., the matrix with *rows* $(\mathbf{v}_1, \ldots, \mathbf{v}_n)$, kills all non-diagonal entries, because distinct $\mathbf{v}_i$ are orthogonal. The remaining diagonal entries are $\lambda_i \mathbf{v}_i \cdot \mathbf{v}_i = \lambda_i$: the eigenvalues of S_0. In summary, the output S of QR iteration approaches a diagonal matrix containing the eigenvalues of its input S_0.

Optional Proof. We will rely on two facts from matrix algebra: (1) Square matrices with orthonormal columns, called *orthogonal matrices*, form a group: Their products are orthogonal and they are invertible

by their transpose, which is also an orthogonal matrix. (2) The QR decomposition is unique up to the sign of the columns.

Armed with these properties we show that $S = X^T S_0 X$ is an invariant for the simultaneous execution of the two algorithms. Initially, X is the identity, S is S_0, so S equals $X^T S_0 X$. Assuming $S = X^T S_0 X$ before a transition, SI updates its state to X', where $S_0 X$ has QR factorization $X'R$. Then the QR decomposition of $S = X^T S_0 X$ must equal $(X^T X')R$. The next state of QR iteration is

$$S' = R(X^T X') = (RX^T)X' = (X'^T S_0)X' = X'^T S_0 X'. \qquad \square$$

7 Spectral decomposition beyond graphs

Spectral decomposition as described so far applies not only to graphs but to any symmetric data set. One important example are word embeddings. These are representations of words by vectors that capture semantic relationships by the geometry of the embedding. When the words are close in meaning their embeddings align. When they are unrelated, the embeddings are roughly orthogonal. A famous example that was found in the word2vec embedding is that the difference between the embeddings of "man" and "woman" is very close to the difference between the embeddings of "king" and "queen". Modern natural processing systems ignore the words themselves and operate directly on their embeddings. (In large language models these embeddings are contextual: The vector representation of a word depends on the presence of other nearby words.) Spectral decomposition reveals such dependencies. It detects that the word embeddings of "man", "woman", "king", and "queen" are almost co-planar.

A more general form of spectral decomposition called *singular value decomposition* applies to asymmetric and even non-square matrices A. A document or a textual prompt can be represented as a matrix whose columns are the embeddings of its words. The singular value decomposition of this matrix gives information about the topic. For example, the top singular vector of a news article about safaris in Kenya might be close to the subspace spanned by (the word embeddings of) "giraffe", "zebra", and "rhinoceros". This could help an automated news feed identify "wild animals" as a topic, or deduce similarity to another article by comparing their respective top singular vectors.

Approximate embeddings are sometimes useful for faster data processing. In topic modelling rows are labeled by words, columns are labeled by documents, and matrix entries represent the number of occurrences of a given word in the document. If the documents are a mixture arising from a small number of sources, we would expect that documents coming from the same source should cluster together. In such applications the matrices are enormous. There could be hundreds of thousands of documents and tens of thousands of words. The number of sources d could be smaller by several orders of magnitude. Projecting to a subspace of dimension d or smaller often retains the data structure while allowing for much faster processing.