Math 361S Lecture notes
Finding eigenvalues: The power method

Jeffrey Wong

April 12, 2019

Topics covered

- Finding eigenvalues
  - Power method, Rayleigh quotient
  - Benefit of symmetric matrices
  - Inverse power method
- General tricks
  - Deflation (and why it is dangerous)
  - Deflation for the power method (second largest λ)
  - Aitken extrapolation

1 Computing the dominant eigenvalues

Throughout, let A be an $n \times n$, non-singular, real-valued matrix with a basis of eigenvectors. Denote the eigenvalues by λ_j and eigenvectors by $\mathbf{v}_j$.

We assume here there is a single eigenvalue of largest magnitude (the 'dominant' eigenvalue). Label them as follows:

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n| > 0.$$

Note that if A has real-valued entries, it must be that λ_1 is real (why?).

The simplest approach to computing λ_1 and $\mathbf{v}_1$ is the power method. The idea is as follows. Let $\mathbf{x}$ be any vector. Then, since $\{\mathbf{v}_j\}$ is a basis,

$$\mathbf{x} = c_1\mathbf{v}_1 + \cdots + c_n\mathbf{v}_n.$$

Now suppose $c_1 \neq 0$ (i.e $\mathbf{x}$ has a non-zero $\mathbf{v}_1$ component). Then

$$A\mathbf{x} = c_1\lambda_1\mathbf{v}_1 + \cdots + c_n\lambda_n\mathbf{v}_n$$

and, applying A repeatedly,

$$A^k\mathbf{x} = \sum_{j=1}^{n} c_j\lambda_j^k\mathbf{v}_j.$$

Since the λ_1^k term is largest in magnitude, in the sequence

$$x, Ax, A^2x, A^3x, \cdots$$

we expect the $\lambda_1^k v_1$ term will dominate so

$$A^k x \approx c\lambda_1^k\mathbf{v}_1 + \text{smaller terms.}$$

Each iteration grows the largest term relative to the others, so after enough iterations only the first term (what we want) will be left.

1.1 Power method: the basic method

Let's formalize the observation and derive a practical method. The main trouble is that λ_1^k will either grow exponentially (bad) or decay to zero (less bad, but still bad). By taking the right ratio, the issue can be avoided.

Claim: Let $\mathbf{x}$ and $\mathbf{x}$ be vectors with $\mathbf{w}^T\mathbf{v}_1 \neq 0$ and such that $\mathbf{x}$ has a non-zero $\mathbf{v}_1$ component. Then

$$\frac{\mathbf{w}^T A^k\mathbf{x}}{\mathbf{w}^T A^{k-1}\mathbf{x}} = \lambda_1 + O((\lambda_2/\lambda_1)^k) \text{ as } k \to \infty \tag{1}$$

Proof. Since the eigenvectors form a basis, there are scalars $c_1, \cdots, c_n$ such that

$$\mathbf{x} = \sum_{j=1}^{n} c_j\mathbf{v}_j.$$

By assumption, $c_1 \neq 0$. Since $A^k\mathbf{v}_j = \lambda_j^k\mathbf{v}_j$ it follows that

$$A^k\mathbf{x} = \sum_{j=1}^{n} c_j\lambda_j^k\mathbf{v}_j = \lambda_1^k\left(c_1\mathbf{v}_1 + \sum_{j=2}^{n} c_j\left(\frac{\lambda_j}{\lambda_1}\right)^k\mathbf{v}_j\right).$$

All the terms in the parentheses except the first go to zero in magnitude as $k \to \infty$. Taking the dot product with $\mathbf{w}$ on the left and computing the ratio (1),

$$\begin{aligned}\frac{\mathbf{w}^T A^k\mathbf{x}}{\mathbf{w}^T A^{k-1}\mathbf{x}} &= \lambda_1\frac{d_1 + \sum_{j=2}^{n} d_j(\lambda_j/\lambda_1)^k}{d_1 + \sum_{j=2}^{n} d_j(\lambda_j/\lambda_1)^{k-1}} \\ &= \lambda_1\frac{1 + O((\lambda_2/\lambda_1)^k)}{1 + O((\lambda_2/\lambda_1)^k)} \\ &= \lambda_1 + O((\lambda_2/\lambda_1)^k)\end{aligned}$$

where $d_j = c_j \mathbf{w}^T \cdot \mathbf{v}_j$ (note that $d_1 \neq 0$ by assumption). Note that we have used that

$$\frac{1}{1+f} = 1 + O(f).$$

□

Thus the power method computes the **dominant eigenvalue** (largest in magnitude), and the convergence is linear. The rate depends on the size of λ_1 relative to the next largest eigenvalue λ_2.

Power method (naive version):

1) Choose vectors $\mathbf{x}$ and $\mathbf{w}$ 'at random'.[1]
2) For $k = 1, 2, \cdots$ compute

$$\mathbf{z}_k = A\mathbf{z}_{k-1}, \qquad \lambda^{(k)} = \frac{\mathbf{w}^T \mathbf{z}_k}{\mathbf{w}^T \mathbf{z}_{k-1}}$$

3) Stop when $\lambda^{(k)}$ is close to converged.

The result is that $\lambda^{(k)}$ converges linearly to the dominant eigenvalue λ_1. The issue of when to stop is addressed in section subsection 1.4.

1.2 A better version

The 'naive version' provides the eigenvalue, and the vector $\mathbf{z}_k = A^k \mathbf{x}$ becomes more and more parallel to the desired eigenvector $\mathbf{v}_1$. However, the magnitude can also grow exponentially (or decay), which is unacceptable.

In practice, there are two differences:

i) Normalize x at each step to have $\|x\|_2 = 1$, which keeps elements from growing exponentially in size. This also gives an eigenvector of unit 2-norm.

ii) Use a 'left' vector that depends on k instead of the fixed w in (1). A good choice is the normalized $A^{k)}x$ (which we have anyway).

Power method (improved version):

- Pick $q^{(0)}$ such that $\|q^{(0)}\|_2 = 1$
- For $k = 1, 2, \cdots$:
  - $x^{(k)} = Aq^{(k-1)}$

- $q^{(k)} = x^{(k)}/\|x^{(k)}\|_2$
- $\lambda^{(k)} = (q^{(k)})^T A q^{(k)}$.

In practice, very little needs to be stored, and the algorithm is quite simple:

- Pick q such that $\|q\|_2 = 1$ and set $x = Aq$
- For $k = 1, 2, \cdots$:
  - $q = x/\|x\|_2$
  - $\lambda = x^T q$
  - $x = Aq$

Claim: The result is that

$$\lambda^{(k)} = \lambda_1 + O((\lambda_2/\lambda_1)^k),$$
$$\mathbf{q}_k = \mathbf{v}_1 + O((\lambda_2/\lambda_1)^k)$$

where $\mathbf{v}_1$ is an eigenvector with $\|\mathbf{v}_1\|_2 = 1$ and the Big-O term means that the error $\|\mathbf{q}^{(k)} - \mathbf{v}_1\|$ has that Big-O.

Sketch of proof: First, define

$$\mathbf{z}_k = A^k \mathbf{q}_0,$$

the 'unscaled' iterates. Observe that since $\|\mathbf{q}\|_2 = 1$ we may write $\lambda^{(k)}$ as a ratio:

$$\lambda^{(k)} = \mathbf{q}^T A \mathbf{q} = \frac{\mathbf{q}^T A \mathbf{q}}{\mathbf{q}^T \mathbf{q}}.$$

It is not too hard (but tedious) to show that the normalization factors all cancel when replacing $\mathbf{q}_k$ with $\mathbf{z}_k$, leading to

$$\lambda^{(k)} = \frac{\mathbf{q}_k^T A \mathbf{q}_k}{\mathbf{q}_k^T \mathbf{q}_k}. = \frac{\mathbf{z}_k^T A \mathbf{z}_k}{\mathbf{z}_k^T \mathbf{z}_k}. \tag{2}$$

Now, as before, write the initial vector in terms of the eigenvector basis:

$$\mathbf{q}_0 = \sum_{j=1}^{n} c_j \mathbf{v}_j, \qquad c_1 \neq 0.$$

It follows that

$$\mathbf{z}_k = A^k \mathbf{q}_0 = \sum_{j=1}^{n} c_j \lambda_j^k \mathbf{v}_j. \tag{3}$$

Then plug (3) into (2) and sort out the mess, identifying the largest terms (left as an exercise; but see below for a special case).

The convergence of the eigenvector takes more work and is omitted here.

1.3 Symmetric power method

The method above has a nice benefit: if A is a real symmetric matrix, then the convergence rate is actually better. If A is (real) symmetric then its eigenvectors are **orthogonal**:

$$\mathbf{v}_i \cdot \mathbf{v}_j = 0 \text{ for } i \neq j.$$

We may also take them to be orthonormal, i.e. $\|\mathbf{v}_i\|_2 = 1$.

Now return to the convergence proof. Observe that

$$\mathbf{z}_k^T \mathbf{z}_k = \left(\sum_{i=1}^n c_i \lambda_i^k \mathbf{v}_i\right) \cdot \left(\sum_{j=1}^n c_j \lambda_j^k \mathbf{v}_j\right) = \sum_{j=1}^n c_j^2 \lambda_j^{2k}$$

since $\mathbf{v}_i \cdot \mathbf{v}_j = 0$ for $i \neq j$ and 1 for $i = j$. Similarly,

$$\mathbf{z}_k^T A \mathbf{z}_k = \left(\sum_{i=1}^n c_i \lambda_i^k \mathbf{v}_i\right) \cdot \left(\sum_{j=1}^n c_j \lambda_j^{k+1} \mathbf{v}_j\right) = \sum_{j=1}^n c_j^2 \lambda_j^{2k+1}.$$

Now plugging (3) into (2) is nice:

$$\begin{aligned}
\lambda^{(k)} &= \frac{\mathbf{z}_k^T A \mathbf{z}_k}{\mathbf{z}_k^T \mathbf{z}_k} \\
&= \frac{\sum_{j=1}^n c_j \lambda_j^{2k+1}}{\sum_{j=1}^n c_j^2 \lambda_j^{2k}} \\
&= \lambda_1 \frac{c_1 + \sum_{j=2}^n c_j (\lambda_j/\lambda_1)^{2k+1}}{c_1 + \sum_{j=2}^n c_j (\lambda_j//\lambda_1)^{2k}} \\
&= \lambda_1 \frac{1 + O((\lambda_2/\lambda_1)^{2k})}{1 + O((\lambda_2/\lambda_1)^{2k})}
\end{aligned}$$

and so

$$\lambda^{(k)} = \lambda_1 + O((\lambda_2/\lambda_1)^{2k})$$

i.e. the rate of convergence is squared compared to the non-symmetric case. The larger error term from the non-symmetric case is a cross term (from $\mathbf{v}_1 \cdot \mathbf{v}_2$) which vanishes here.

1.4 Aitken extrapolation

We saw that the eigenvalue estimate converges linearly to the true value:

$$\lambda^{(n)} \sim \lambda + cr^n$$

where $r = (\lambda_2/\lambda_1)$ (non-symmetric) or $(\lambda_2/\lambda_1)^2$ (symmetric). Both c and r are, of course, unknown. However, just as with Richardson extrapolation, we can 'cheat' here and 'solve' for c and r to improve the estimate.

To generalize, suppose we have a scalar sequence

$$x_n \sim x + cr^n \tag{4}$$

with a limit x, the desired quantity. Observe that

$$\lim_{n\to\infty} \frac{x_{n+1} - x_n}{x_n - x_{n-1}} = \lim_{n\to\infty} \frac{cr^{n+1} - cr^n}{cr^n - cr^{n-1}} = r.$$

Similarly

$$(x_{n+1} - x_n)^2 \sim c^2(r^n)^2(r-1)^2$$

and

$$x_{n+2} - 2x_{n+1} + x_n \sim cr^{n+2} - 2r^{n+1} + cr^n = cr^n(r-1)^2.$$

Thus

$$\frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n} \sim cr^n \sim x_n - x.$$

Let us define the new sequence

$$y_n := Ax_n = x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}. \tag{5}$$

If (4) holds exactly then $y_n = x$. Otherwise, $\{Ax_n\}$ tends to converge to x faster (linear, with a better rate) than $\{x_n\}$. This process is called **Aitken extrapolation**, which can be used to accelerate the convergence of a sequence.

In practice, it is used when a sequence is known to converge linearly (from theory), and one wants a bit of extra accuracy (or efficiency) for cheap.

Example: Consider the sequence

$$x_n = 1 + 2^{-n} + 4^{-n}.$$

Then $x_n \to 1$ linearly with rate $1/2$. However,

$$Ax_n = 1 + O(4^{-n})$$

(proof: exercise), so Ax_n converges with a rate of $1/4$ instead. By doing the more or less trivial computation (5), we have a new approximating sequence $\{Ax_n\}$ that is much more accurate (twice the number of digits!).

An example for eigenvalues was shown in subsection 1.5.

Remark: There are many other acceleration techniques of this flavor, that recycle existing data to construct better approximations. For iterative methods, a more sophisticated technique called **Chebyshev acceleration** can be used to obtain an 'optimized' approximation using the first k iterates.

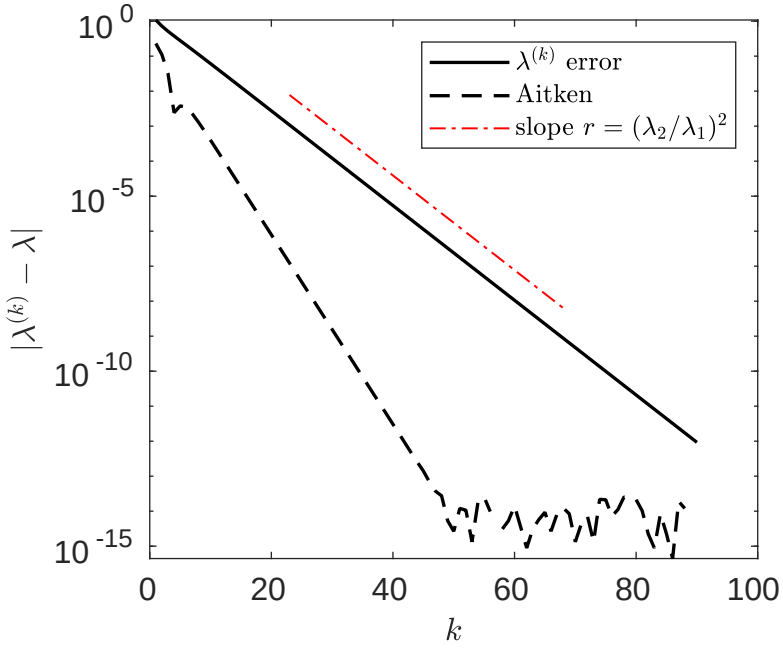


Figure 1: Error in the eigenvalue approximation for the symmetric power method with and without Aitken extrapolation.

1.5 Example

We find the dominant eigenvalue of the matrix

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & -1 & 1 \\ 2 & -1 & -1 \end{bmatrix},$$

The result of using the power method with $\mathbf{q}_0 = (1, 0, 0)^T$ is shown in Figure 1. The eigenvalues are

$$\lambda_1 \approx 2.74, \quad \lambda_2 = -2.35, \quad \lambda_3 = -1.4.$$

The error in $\lambda^{(k)}$ (the approximation to λ_1) is shown along with an accelerated estimate using Aitken extrapolation (subsection 1.4). Since A is symmetric, the convergence is linear with rate

$$r = (\lambda_2/\lambda_1)^2 \approx 0.73$$

which is not bad. The accelerated version (which does not require any more work), however, does much better. The noise for $k > 50$ is due to rounding; of course we cannot expect the error to get much better then around machine precision. There is some additional error in the Aitken sequence due to cancellation (ratio of two differences of nearly equal numbers), but it is not of much concern here.

2 Inverse power method

A simple change allows us to compute the **smallest** eigenvalue (in magnitude). Let us assume now that A has eigenvalues

$$|\lambda_1| \geq |\lambda_2| \cdots > |\lambda_n|.$$

Then A^{-1} has eigenvalues λ_j^{-1} satisfying

$$|\lambda_n^{-1}| > |\lambda_2^{-1}| \geq \cdots \geq |\lambda_n^{-1}|.$$

Thus if we apply the power method to A^{-1}, the algorithm will give $1/\lambda_n$, yielding the smallest eigenvalue of A (after taking the reciprocal at the end).

Note that in practice, instead of computing A^{-1}, we first compute an LU factorization of A, and then solve

$$Ax^{(k+1)} = x^{(k)}$$

at each step, which only takes $O(n^2)$ operations after the initial work.

Now suppose instead we want to find the eigenvalue closest to a number μ. Notice that the matrix $(A - \mu I)^{-1}$ has eigenvalues

$$\frac{1}{\lambda_j - \mu}, \qquad j = 1, \cdots n.$$

The eigenvalue of largest magnitude will be $1/(\lambda_{j_0} - \mu)$ where λ_{j_0} is the closest eigenvalue to μ (assuming there is only one). This leads to the **inverse power method** (sometimes called **inverse iteration**):

Inverse power method: To find the eigenvalue of A closest to μ,

1) Apply the power method to $(A - \mu I)^{-1}$, solving

$$(A - \mu I)\mathbf{x}_k = \mathbf{x}_{k-1}$$

at each step using some linear system solver (e.g. LU factorization).

2) Compute λ from the output $1/(\lambda - \mu)$.

Note that if μ is fixed, the LU factorization only needs to be computed once.!

The method is a cheap, often effective way of computing one eigenvalue, which is often all that matters. Moreover, a good choice of μ helps convergence. It should be a 'guess' of the eigenvalue. Suppose the eigenvalues satisfy

$$\frac{1}{|\lambda_1 - \mu|} > \frac{1}{|\lambda_2 - \mu|} > \cdots$$

and we seek λ_1. Then inverse iteration will yield λ_1, and from the power method,

$$\text{error in } \lambda_1 = O\left(\frac{|\lambda_1 - \mu|^k}{|\lambda_2 - \mu|^k}\right)$$

with k replaced by $2k$ for a symmetric A. In either case, the (linear) rate improves as μ gets closer to λ_1. With a good guess, the convergence rate will be quite good (close to zero).

An improvement for symmetric matrices: The advantage of a fixed value of μ is that $A - \mu I$ only needs to be factored once. But the convergence is only linear.

For a **symmetric matrix**, the convergence can be accelerated even more by choosing μ at each step as the **Rayleigh quotient**

$$\mu_k = \frac{x^{(k)} \cdot Ax^{(k)}}{x^{(k)} \cdot x^{(k)}}.$$

The iteration ('Rayleigh quotient iteration') is then

$$(A - \mu_k I)x^{(k+1)} = x^{(k)}.$$

For a typical symmetric matrix, the convergence becomes **cubic**, which is much faster than linear! Thus the disadvantage of factoring $A - \mu_k I$ at each step is balanced out by the dramatic boost in convergence rate.

2.1 More on eigenvalues

The methods above give an extreme eigenvalue, but all the other eigenvalues disappear as the iteration proceeds. Finding the other eigenvalues is more difficult, and the power/inverse power methods are not particularly well suited to doing so. There are some ways to reduce the problem after finding one eigenvalue (to find the next largest using the power method, and so on), but they can be problematic (see next section).

For computing all the eigenvalues of A, there is a powerful class of iterative methods that can be used, such as the QR algorithm, which will not be covered here.

3 Deflation

Here we introduce a practical trick that is occasionally useful if used carefully. For robust computation, other more sophisticated methods are used instead.

3.1 Deflation: the idea

To illustrate the point, consider the problem of finding **all the roots** of a degree n polynomial $p(x)$ with n real roots. Suppose we have access to a solver that finds one root like Newton. Then we can proceed by using **deflation**:

1) Use Newton's method to find a root x_0 of $p(x)$
2) Define
$$q_0(x) = p(x)/(x - x_0)$$
and find a zero x_1 of q_0.
3) Define $q_1 = q_0/(x - x_1)$, find a root x_2 and so on up through q_{n-1}.

Definition: Given a method that finds one solution to a system, the process of 'dividing out a solution' one by one to get all of them is called **deflation**.

In theory, this will 'divide out' the found roots one by one, e.g.

$$p = (x-1)(x-2)(x-3) \implies q_0 = (x-1)(x-3) \implies q_1 = (x-1).$$

Note that each q_k is smooth; the division does not add any singularities.

However, if x_j is not computed exactly, e.g. $\tilde{x}_0 \approx x_0$, there is trouble, since then

$$q_0(x) = \frac{x - x_0}{x - \tilde{x}_0}(\cdots)$$

which introduces a 'singularity' at $\tilde{x}_0$ that can be disastrous unless $\tilde{x}_0$ is computed to very high accuracy. It could be that Newton's method on q_0 will still converge to x_0, or just blow up. The method relies on a **theoretical guarantee** that must hold exactly for the method to be correct. Otherwise, it may work in practice, or it may not.

This is not to say the method is worthless; just that it should be used cautiously, and without high expectations.

3.2 Deflation for the power method

Since the eigenvalues are the roots of the characteristic polynomial

$$p(\lambda) = \det(A - \lambda I),$$

we might expect that deflation could be used in conjunction with the power method. Indeed, it can be used, if one is willing to accept the numerical issues that arise.

Reminder (outer product): The **outer product** $\mathbf{v}\mathbf{w}^T$ of two column vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ is the matrix

$$C = \mathbf{v}\mathbf{w}^T \text{ with } c_{ij} = \mathbf{v}_i \mathbf{w}_j.$$

That is, the (i, j) component of the outer product is the i-th component of $\mathbf{v}$ times the j-th of $\mathbf{w}_j$. Observe that the rank of an outer product is always one; for this reason a matrix

$$B = A + \mathbf{v}\mathbf{w}^T$$

is called a 'rank one perturbation' (it is A plus a rank-one matrix).

Note that the transpose notation makes some manipulations convenient, e.g.

$$(\mathbf{v}\mathbf{w}^T)\mathbf{x} = \mathbf{v}(\mathbf{w}^T\mathbf{x}) = (\mathbf{w}^T\mathbf{x})\mathbf{v}$$

noting that $\mathbf{v}\mathbf{w}^T$ is a matrix and $\mathbf{w}^T\mathbf{x}$ is a scalar (inner product).

Let A be an $n \times n$ real symmetric matrix with (distinct) non-zero eigenvalues

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| > 0.$$

and eigenvector $\mathbf{v}_1, \cdots, \mathbf{v}_n$. The goal of **deflation** is to build a modified matrix that has only $\lambda_2, \cdots, \lambda_n$ as eigenvalues.[2] Suppose we have obtained λ_1 and $\mathbf{v}_1$.

This goal is achieved by defining the 'deflated' matrix

$$B = A - \lambda_1 \mathbf{v}_1 \mathbf{x}^T$$

where $\mathbf{x}$ is any vector such that

$$\mathbf{x}^T \mathbf{v}_1 = 1.$$

Claim: The eigenvalues of B are 0 and $\lambda_2, \cdots, \lambda_n$ with eigenvectors $\mathbf{v}_1, \mathbf{w}_2, \cdots \mathbf{w}_n$. Only $\mathbf{v}_1$ is the same, but $\mathbf{v}_2$ can be obtained easily from $\mathbf{w}_2$.

Proof. First, 0 is an eigenvalue of B with eigenvector $\mathbf{v}_1$ since

$$B\mathbf{v}_1 = A\mathbf{v}_1 - \lambda_1 \mathbf{v}_1 \mathbf{x}^T \mathbf{v}_1 = \lambda_1 \mathbf{v}_1 - \lambda_1 \mathbf{v}_1 = 0$$

by the choice of $\mathbf{x}$. If $j \neq 1$ then $\mathbf{v}_j$ is not an eigenvector. Instead, look for an eigenvector

$$\mathbf{w}_j = c\mathbf{v}_1 + \mathbf{v}_j.$$

We need to find a c such that $(B - \lambda_j I)\mathbf{w}_j = 0$. Compute

$$\begin{aligned}(B - \lambda_j I)\mathbf{w}_j &= c(A - \lambda_j I)\mathbf{v}_1 - \lambda_1 \mathbf{v}_1 \mathbf{x}^T \mathbf{w}_j \\ &= c(\lambda_1 - \lambda_j)\mathbf{v}_1 - \lambda_1 \mathbf{v}_1 \mathbf{x}^T \mathbf{w}_j \\ &= (c(\lambda_1 - \lambda_j) - \lambda_1 \mathbf{x}^T \mathbf{w}_j)\mathbf{v}_1.\end{aligned}$$

[2] Discussed also in some textbooks, e.g. Burden and Faires, *Numerical analysis, p587-588*, from which this presentation is adapted.

Since $\mathbf{x}^T\mathbf{w}_j = c + \mathbf{x}^T\mathbf{v}_j$, the right value of c is

$$c = -\lambda_1 \mathbf{x}^T \mathbf{v}_j / \lambda_j$$

so λ_j is an eigenvalue with eigenvector

$$\mathbf{w}_j = \mathbf{v}_j - \left(\frac{\lambda_1}{\lambda_j} \mathbf{x}^T \mathbf{v}_j \right) \mathbf{v}_1. \tag{6}$$

□

This gives a way to compute the second-largest eigenvalue in magnitude.

Deflation for the second-largest eigenvalue: To compute λ_2 and $\mathbf{v}_2$ (where $|\lambda_1| > |\lambda_2| > \cdots$),

- Use the power method to obtain λ_1 and $\mathbf{v}_1$,
- Construct the deflated matrix

  $$B = A - \lambda_1 \mathbf{v}_1 \mathbf{x}^T$$

  where $\mathbf{x}$ is chosen so that $\mathbf{x}^T\mathbf{v}_1 = 1$.
- Use the power method on B to obtain λ_2 and $\mathbf{w}_2$, then recover the eigenvector $\mathbf{v}_2$ for A from (6).

It is not hard to find such a vector $\mathbf{x}$, but the numerical stability depends on this choice. One option is **Weilandt deflation**, which chooses

$$\mathbf{x} = \frac{1}{\lambda_1 (\mathbf{v}_1)_1} R_1$$

where $R_1 = (a_{11}, \cdots a_{1n})^T$ is the first row of A and $(\mathbf{v}_1)_1$ is the first component of $\mathbf{v}_1$.

The peril is that deflation is numerically unstable, and repeated applications can lead to disaster. Using it to get λ_2 is usually fine except for ill-behaved eigen-problems, but it is **not advisable to use it to find all the eigenvalues**.

Here is an extreme example where there is trouble. Let

$$A = \lambda_1 \mathbf{v}_1 \mathbf{v}_1^T + \lambda_2 \mathbf{v}_2 \mathbf{v}_2^T$$

where $\mathbf{v}_1^T\mathbf{v}_1 = \mathbf{v}_2^T\mathbf{v}_2 = 1$. Then A has eigenvalues λ_1, λ_2 and eigenvectors $\mathbf{v}_1, \mathbf{v}_2$. Now suppose $\tilde{\lambda}_1 \approx \lambda_1$ is computed. We then deflate:

$$B = A - \tilde{\lambda}_1 \mathbf{v}_1 \mathbf{v}_1^T$$

choosing $\mathbf{x} = \mathbf{v}_1^T$ for simplicity (of the example). Then

$$B = (\lambda_1 - \tilde{\lambda}_1)\mathbf{v}_1\mathbf{v}_1^T + \lambda_2\mathbf{v}_2\mathbf{v}_2^T.$$

So B has an eigenvalue $\lambda_1 - \tilde{\lambda}_1$. If, say

$$\lambda_1 = 10^8, \quad \lambda_2 = 10^{-8}$$

and $\tilde{\lambda}_1$ is computed to machine precision (relative error 10^{-16}) then

$$|\lambda_1 - \tilde{\lambda}_1| \approx 10^{-8}$$

which is the same size as λ_2. Thus the spurious 'leftover' from the deflation is actually the dominant part, and the power method cannot see λ_2.

4 An example: PageRank

The notes here expand on the brief discussion in the textbook. For an introduction, see https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/moler/exm/chapters/pagerank.pdf. For a detailed exposition, the best source is Langville & Myer, *Google's PageRank and Beyond*.

A Matlab demonstration can be found at https://www.mathworks.com/help/matlab/math/use-page-rank-algorithm-to-rank-websites.html. The summary here is adapted from this source.

4.1 The setup

We regard a set of web pages as a graph whose vertices are pages $\{W_i\}$ connected by a directed edge $i \to j$ if W_i links to page W_j. The goal is to determine how a metric for the popularity of each page.

Imagine a hypothetical user (a 'surfer') meandering through the web by following links. The surfer will visit pages that are more connected more often. We define

$$x_i = \text{ proportion of time spent at page } W_i.$$

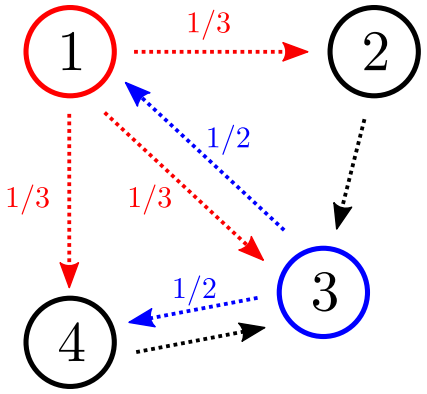
Given pages W_i and W_j, also define the **transition probability**

$$p_{ij} = \text{ probability of going from } W_i \text{ to } W_j. \tag{7}$$

and the transition matrix P whose entries are p_{ij}. A small example and its transition matrix are shown in Figure 2.

If the surfer is at page W_i, they have come from some other site, which occurs with probability p_{ji}. It should follow that

$$x_i = \sum_j p_{ji}x_j.$$



$$\begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1 & 0 \\ 1/2 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 2: Directed graph of four linked pages to be ranked and the transition matrix.

In probabilistic terms,[3]

$$P(\text{at } W_i) = \sum_j P(\text{goes from } W_j \text{ to } W_i \,\big|\, \text{at } W_j) \cdot P(\text{at } W_j).$$

Now we need to know how likely it is that the surfer goes from i to j. We need one last definition; the **adjacency matrix** A is the matrix whose (i, j) entry is 1 if there is a link from W_i to W_j:

$$a_{ij} = \begin{cases} 1 & i \text{ points to } j \\ 0 & \text{otherwise} \end{cases}.$$

The PageRank algorithm (in its simplest form) makes the assumption that the surfer clicks a link uniformly at random to leave a page:

$$p_{ji} = \frac{a_{ji}}{N_j}, \qquad \text{where } N_j = \#\text{ of outgoing links from } W_j.$$

4.2 The eigenvalue problem

Now let $\mathbf{x}$ be the vector of x_i's. Then the steady-state equation (7) is

$$\mathbf{x} = P^T\mathbf{x}$$

or, setting $M = P^T$,

$$\mathbf{x} = M\mathbf{x}. \tag{8}$$

Thus $\mathbf{x}$ is an eigenvector of M with eigenvalue 1. It can be shown that if the system is 'irreducible' (all states can be reached from all others), then:[4]

- $\lambda = 1$ is an eigenvalue with multiplicity 1

[3]This is a specific example of a **steady state** of a **Markov chain**.

[4]This is a consequence of the **Perron-Frobenius theorem**. The key facts here are that the entries of M are non-negative and the entries in each column sum to 1.

- All other eigenvalues are less than one in magnitude

Thus there is a unique eigenvector $\mathbf{x}$ (the 'ranking') such that

$$\sum_i x_i = 1.$$

The popularity of the website is then measured by this ranking. To get around the issue of pages that do not link back and for numerical reasons, there is one more ingredient in the basic model. The surfer is assumed to jump to a random page (regardless of links) with a small probability $1 - \alpha$. That is,

$$m_{ij} = \alpha \frac{a_{ji}}{N_j} + \frac{1-\alpha}{n}.$$

The value of α is typically chosen to be near 1, e.g. $\alpha \approx 0.85$. In matrix terms,

$$M = \alpha P^T + \frac{1-\alpha}{n} E \tag{9}$$

where E is a matrix of all ones.

4.3 Computation

The ranking $\mathbf{x}$ is the eigenvector for the dominant eigenvalue $\lambda = 1$. Thus the power method can be employed. Moreover, we do not need to normalize at each step (why not?). The matrix P^T has a simple structure, so the multiplication $M\mathbf{x}$ can be implemented quite efficiently.

It is not hard to show that

$$\sum x_i = 1 \implies \sum (Mx)_i = 1$$

and then that

$$(Mx)_i = \alpha \sum_{j=1}^{n} (a_{ji} x_j / N_j) + (1-\alpha)/n.$$

In Matlab, this is simply coded as

```
Mx = alpha*A'*x./N + (1-alpha)/n
```

where N is the vector of N_j's. The adjacency matrix A is typically sparse (most pages link only to a few others), so $A^T x$ is fast to compute. Note that the sparseness is essential for the enormous data-sets produced by crawling the web.

4.4 Example

Let us compute the ranking for the small system in Figure 2. The transition probabilities and transition matrix P are also shown. The adjacency matrix and N values are

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad N = \begin{bmatrix} 3 \\ 1 \\ 2 \\ 1 \end{bmatrix}.$$

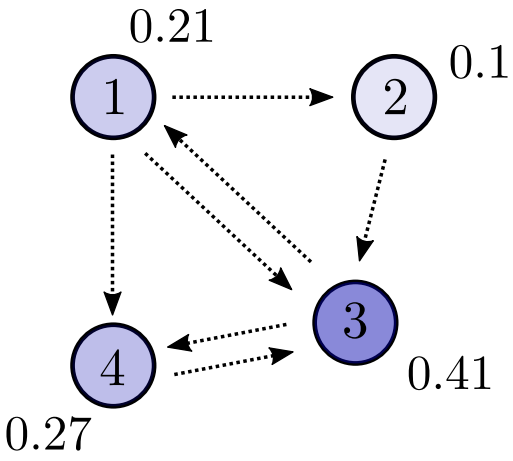


Figure 3: Ranking (darker shade $\implies$ higher rank) and computed $\mathbf{x}$ for the four-page example.

The matrix to be used for the power method is (with $n = 4$)

$$M = \alpha P^T + \frac{(1-\alpha)}{n} E$$

where E is a matrix of all ones and we seek an eigenvector $\mathbf{x}$ such that

$$\mathbf{x} = M\mathbf{x}.$$

Applying the power method (no normalization is required because the eigenvalue is 1) with $\alpha = 0.85$, we obtain the solution

$$\mathbf{x} = (0.214,\ 0.098,\ 0.414,\ 0.274)^T$$

which indicates that the highest ranked page should be #3 (see Figure 3). Since 3 is in the middle of the network and has the most incoming links, the result makes sense. Similarly, #2 is last, as it is only connected from #1.

For a larger example, see the Matlab demonstration linked at the start of the section.