Notes for 2016-10-17

1 Power iteration

In most introductory linear algebra classes, one computes eigenvalues as roots of a characteristic polynomial. For most problems, this is a *bad idea*: the roots of the characteristic polynomial are often very sensitive to changes in the polynomial coefficients even when they correspond to well-conditioned eigenvalues. Rather than starting from this point, we will start with another idea: the *power iteration*.

Suppose $A \in \mathbb{C}^{n\times n}$ is diagonalizable, with eigenvalues $\lambda_1, \ldots, \lambda_n$ ordered so that

$$|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|.$$

Then we have $A = V\Lambda V^{-1}$ where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. Now, note that

$$A^k = (V\Lambda V^{-1})(V\Lambda V^{-1})\ldots(V\Lambda V^{-1}) = V\Lambda^k V^{-1},$$

or, to put it differently,

$$A^k V = V\Lambda^k.$$

Now, suppose we have a randomly chosen vector $x = V\tilde{x} \in \mathbb{C}^n$, and consider

$$A^k x = A^k V\tilde{x} = V\Lambda^k\tilde{x} = \sum_{j=1}^{n} v_j\lambda_j^k\tilde{x}_j.$$

If we pull out a constant factor from this expression, we have

$$A^k x = \lambda_1^k\left(\sum_{j=1}^{n} v_j\left(\frac{\lambda_j}{\lambda_1}\right)^k\tilde{x}_j\right).$$

If $|\lambda_1| > |\lambda_2|$, then $(\lambda_j/\lambda_1)^k \to 0$ for each $j > 1$, and for large enough k, we expect $A^k x$ to be nearly parallel to v_1, assuming $\tilde{x}_1 \neq 0$. This is the idea behind the power iteration:

$$x^{(k+1)} = \frac{Ax^{(k)}}{\|Ax^{(k)}\|} = \frac{A^k x^{(0)}}{\|A^k x^{(0)}\|}.$$

Assuming that the first component of $V^{-1}x^{(0)}$ is nonzero and that $|\lambda_1| > |\lambda_2|$, the iterates $x^{(k)}$ converge linearly to the "dominant" eigenvector of A, with the error asymptotically decreasing by a factor of $|\lambda_1|/|\lambda_2|$ at each step.

There are three obvious potential problems with the power method:

1. What if the first component of $V^{-1}x^{(0)}$ is zero?
2. What λ_1/λ_2 is near one?
3. What if we want the eigenpair (λ_j, v_j) for $j \neq 1$?

The first point turns out to be a non-issue: if we choose $x^{(0)}$ at random, then the first component of $V^{-1}x^{(0)}$ will be nonzero with probability 1. Even if we were so extraordinarily unlucky as to choose a starting vector for which $V^{-1}x^{(0)}$ *did* have a zero leading coefficient, perturbations due to floating point arithmetic would generally bump us to the case in which we had a nonzero coefficient.

The second and third points turn out to be more interesting, and we address them now.

2 Spectral transformation and shift-invert

Suppose again that A is diagonalizable with $A = V\Lambda V^{-1}$. The power iteration relies on the identity

$$A^k = V\Lambda^k V^{-1}.$$

Now, suppose that $f(z)$ is any function that is defined locally by a convergent power series. Then as long as the eigenvalues are within the radius of convergence, we can define $f(A)$ via the same power series, and

$$f(A) = Vf(\Lambda)V^{-1}$$

where $f(\Lambda) = \mathrm{diag}(f(\lambda_1), f(\lambda_2), \ldots, f(\lambda_n))$. So the spectrum of $f(A)$ is the image of the spectrum of A under the mapping f, a fact known as the *spectral mapping theorem*.

As a particular instance, consider the function $f(z) = (z - \sigma)^{-1}$. This gives us

$$(A - \sigma I)^{-1} = V(\Lambda - \sigma I)^{-1}V^{-1},$$

and so if we run power iteration on $(A - \sigma I)^{-1}$, we will converge to the eigenvector corresponding to the eigenvalue λ_j for which $(\lambda_j - \sigma)^{-1}$ is maximal — that is, we find the eigenvalue closest to σ in the complex plane. Running the power method on $(A - \sigma I)^{-1}$ is sometimes called the shift-invert power method.

3 Changing shifts

If we know a shift σ that is close to a desired eigenvalue, the shift-invert power method may be a reasonable method. But even with a good choice of shift, this method converges at best linearly (i.e. the error goes down by a constant factor at each step). We can do better by choosing a shift *dynamically*, so that as we improve the eigenvector, we also get a more accurate shift.

Suppose $\hat{v}$ is an approximate eigenvector for A, i.e. we can find some $\hat{\lambda}$ so that

$$A\hat{v} - \hat{v}\hat{\lambda} \approx 0. \tag{1}$$

The choice of corresponding approximate eigenvalues is not so clear, but a reasonable choice (which is always well-defined when $\hat{v}$ is nonzero) comes from multiplying (1) by $\hat{v}^*$ and changing the $\approx$ to an equal sign:

$$\hat{v}^* A\hat{v} - \hat{v}^*\hat{v}\hat{\lambda} = 0.$$

The resulting eigenvalue approximation $\hat{\lambda}$ is the *Rayleigh quotient*:

$$\hat{\lambda} = \frac{\hat{v}^* A\hat{v}}{\hat{v}^*\hat{v}}.$$

If we dynamically choose shifts for shift-invert steps using Rayleigh quotients, we get the *Rayleigh quotient iteration*:

$$\begin{aligned}
\lambda_{k+1} &= \frac{v^{(k)*} A v^{(k)}}{v^{(k)*} v^{(k)}} \\
v^{(k+1)} &= \frac{(A - \lambda_{k+1})^{-1} v^{(k)}}{\|(A - \lambda_{k+1})^{-1} v^{(k)}\|_2}
\end{aligned}$$

Unlike the power method, the Rayleigh quotient iteration has locally quadratic convergence — so once convergence sets in, the number of correct digits roughly doubles from step to step. We will return to this method later when we discuss symmetric matrices, for which the Rayleigh quotient iteration has locally *cubic* convergence.

4 Subspaces and orthogonal iteration

So far, we have still not really addressed the issue of dealing with clustered eigenvalues. For example, in power iteration, what should we do if λ_1 and

λ_2 are very close? If the ratio between the two eigenvalues is nearly one, we don't expect the power method to converge quickly; and we are likely to not have at hand a shift which is much closer to λ_1 than to λ_2, so shift-invert power iteration might not help much. In this case, we might want to relax our question, and look for the invariant subspace associated with λ_1 and λ_2 (and maybe more eigenvalues if there are more of them clustered together with λ_1) rather than looking for the eigenvector associated with λ_1. This is the idea behind *subspace iteration*.

In subspace iteration, rather than looking at $A^k x_0$ for some initial vector x_0, we look at $\mathcal{V}_k = A^k \mathcal{V}_0$, where $\mathcal{V}_0$ is some initial subspace. If $\mathcal{V}_0$ is a p-dimensional space, then under some mild assumptions the space $\mathcal{V}_k$ will asymptotically converge to the p-dimensional invariant subspace of A associated with the p eigenvalues of A with largest modulus. The analysis is basically the same as the analysis for the power method. In order to actually *compute*, though, we need bases for the subspaces $\mathcal{V}_k$. Let us define these bases by the recurrence

$$Q_{k+1} R_{k+1} = A Q_k$$

where Q_0 is a matrix with p orthonormal columns and $Q_{k+1} R_{k+1}$ represents an economy QR decomposition. This recurrence is called *orthogonal iteration*, since the columns of Q_{k+1} are an orthonormal basis for the range space of AQ_k, and the span of Q_k is the span of $A^k Q_0$.

Assuming there is a gap between $|\lambda_p|$ and $|\lambda_{p+1}|$, orthogonal iteration will usually converge to an orthonormal basis for the invariant subspace spanned by the first p eigenvectors of A. But it is interesting to look not only at the behavior of the subspace, but also at the span of the individual eigenvectors. For example, notice that the first column $q_{k,1}$ of Q_k satisfies the recurrence

$$q_{k+1,1} r_{k+1,11} = A q_{k,1},$$

which means that the vectors $q_{k,1}$ evolve according to the power method! So over time, we expect the first columns of the Q_k to converge to the dominant eigenvector. Similarly, we expect the first two columns of Q_k to converge to a basis for the dominant two-dimensional invariant subspace, the first three columns to converge to the dominant three-dimensional invariant subspace, and so on. This observation suggests that we might be able to get a complete list of nested invariant subspaces by letting the initial Q_0 be some square matrix. This is the basis for the workhorse of nonsymmetric eigenvalue algorithms, the *QR method*, to which we shall turn next time.

5 Codes

5.1 Basic power iteration

```
% [v,lambda] = power(A, v, maxiter, rtol)
%
% Run power iteration to compute the dominant eigenvalue of A and
% an associated eigenvector. This will fail in general if there are
% multiple dominant eigenvalues (e.g. from a complex conjugate pair).
%
% Inputs:
%  A: Matrix to be analyzed
%  v: Start vector (default random)
%  maxiter: Maximum number of iterations allowed (default 1000)
%  rtol: Rel residual tolerance for convergence (default 1e-3)
%
function [v,lambda] = power(A, v, maxiter, rtol)

  % Fill in default parameters
  if nargin < 2, v = []; end
  if nargin < 3, maxiter = 1000; end
  if nargin < 4, rtol = 1e-3; end

  % Start with a random vector by default
  if isempty(v)
    v = randn(length(A),1);
  end
  v = v / norm(v);

  % Get estimate of 2-norm of A for convergence test
  normAest = sqrt(norm(A,1) * norm(A,inf));

  % Run the iteration
  Av = A*v;
  for k = 1:maxiter

    % Take a power method step and compute Rayleigh quotient
    v = Av/norm(Av);
    Av = A*v;
    lambda = v'*Av;

    % Compute the residual and check vs tolerance
    r = Av-v*lambda;
    normr = norm(r);
    if normr < rtol*normAest,
```

```
    return;
  end

 end

 % If we get here, give a warning
 warning(...
  sprintf('Power did not converge (rel resid %e after %d steps)', ...
        normr/normAest, maxiter));

end
```

5.2 Rayleigh quotient iteration

```
% [v,lambda] = rqi(A, sigma, v, maxiter, rtol)
%
% Run Rayleigh quotient iteration to compute an eigenpair of A.
%
% Inputs:
%  A: Matrix to be analyzed
%  sigma: Initial shift
%  v: Start vector (default random)
%  maxiter: Maximum number of iterations allowed (default 1000)
%  rtol: Rel residual tolerance for convergence (default 1e-8)
%
function [v,lambda] = rqi(A, lambda, v, maxiter, rtol)

 % Fill in default parameters
 if nargin < 2, lambda = 0; end
 if nargin < 3, v = []; end
 if nargin < 4, maxiter = 1000; end
 if nargin < 5, rtol = 1e-8; end

 % Start with a random vector by default
 if isempty(v)
  v = randn(length(A),1);
 end
 v = v / norm(v);

 % Get estimate of 2-norm of A for convergence test
 normAest = sqrt(norm(A,1) * norm(A,inf));
 I = eye(length(v));

 % Run the iteration
 for k = 1:maxiter
```

```
32
33    % Take a power method step and compute Rayleigh quotient
34    v = (A-lambda*I)\v;
35    v = v/norm(v);
36    Av = A*v;
37    lambda = v'*Av;
38
39    % Compute the residual and check vs tolerance
40    r = Av-v*lambda;
41    normr = norm(r);
42    if normr < rtol*normAest,
43      return;
44    end
45
46  end
47
48  % If we get here, give a warning
49  warning(...
50    sprintf('RQI did not converge (rel resid %e after %d steps)', ...
51          normr/normAest, maxiter));
52
53 end
```

5.3 Subspace iteration

```
1  % [v,lambda] = subspace(A, k, V, maxiter, rtol)
2  %
3  % Orthogonal iteration to compute the dominant invariant subspace of A.
4  %
5  % Inputs:
6  %  A: Matrix to be analyzed
7  %  k: Dimension of subspace
8  %  v: Start vector (default random)
9  %  maxiter: Maximum number of iterations allowed (default 1000)
10 %  rtol: Rel residual tolerance for convergence (default 1e-4)
11 %
12 function [V,L] = subspace(A, k, V, maxiter, rtol)
13
14  % Fill in default parameters
15  if nargin < 2, k = 1;  end
16  if nargin < 3, V = []; end
17  if nargin < 4, maxiter = 1000; end
18  if nargin < 5, rtol = 1e-4; end
19
20  % Start with a random vector by default
```

```
21   if isempty(V)
22     V = randn(length(A),k);
23   end
24   V = V / norm(V);
25
26   % Get estimate of 2-norm of A for convergence test
27   normAest = sqrt(norm(A,1) * norm(A,inf));
28
29   % Run the iteration
30   AV = A*V;
31   for k = 1:maxiter
32
33     % Take a power method step and compute Rayleigh quotient
34     [V,R] = qr(AV,0);
35     AV = A*V;
36     L = V'*AV;
37
38     % Compute the residual and check vs tolerance
39     R = AV-V*L;
40     normR = norm(R, 'fro');
41     if normR < rtol*normAest,
42       return;
43     end
44
45   end
46
47   % If we get here, give a warning
48   warning(...
49     sprintf('Power did not converge (rel resid %e after %d steps)', ...
50           normR/normAest, maxiter));
51
52 end
```