# Notes for 2016-11-16

## 1 Arnoldi

Krylov subspaces are good spaces for approximation schemes. But the power basis (i.e. the basis $A^j b$ for $j = 0, \ldots, k-1$) is not good for numerical work. The vectors in the power basis tend to converge toward the dominant eigenvector, and so the power basis quickly becomes ill-conditioned. We would much rather have orthonormal bases for the same spaces. This is where the Arnoldi iteration and its kin come in.

Each step of the Arnoldi iteration consists of two pieces:

- Compute $Aq_k$ to get a vector in the space of dimension $k+1$
- Orthogonalize $Aq_k$ against $q_1, \ldots, q_k$ using Gram-Schmidt. Scale the remainder to unit length.

If we keep track of the coefficients in the Gram-Schmidt process in a matrix $H$, we have

$$h_{k+1,k} q_{k+1} = Aq_k - \sum_{j=1}^{k} q_j h_{jk}$$

where $h_{jk} = q_j^T A q_k$ and $h_{k+1,k}$ is the normalization constant. Rearranging slightly, we have

$$Aq_k = \sum_{j=1}^{k+1} q_j h_{jk}.$$

Defining $Q_k = \begin{bmatrix} q_1 & q_2 & \ldots & q_k \end{bmatrix}$, we have the *Arnoldi decomposition*

$$AQ_k = Q_{k+1} \bar{H}_k, \quad \bar{H}_k = \begin{bmatrix} h_{11} & h_{12} & \ldots & h_{1k} \\ h_{21} & h_{22} & \ldots & h_{2k} \\ & h_{32} & \ldots & h_{3k} \\ & & \ddots & \vdots \\ & & & h_{k+1,k} \end{bmatrix} \in \mathbb{R}^{(k+1)\times k}.$$

The Arnoldi *decomposition* is simply the leading $k$ columns of an upper Hessenberg reduction of the original matrix $A$. The Arnoldi *algorithm* is the

```
% [Q,H] = arnoldi(A,b)
%
% Compute an Arnoldi decomposition
%
%  A*Q(:,1:end-1) = Q*H
%
% where H is a k+1-by-k upper Hessenberg matrix and Q has
% orthonormal columns.
%
function [Q,H] = arnoldi(A,b,k)

  n = length(A);
  Q = zeros(n,k+1); % Orthonormal basis
  H = zeros(k+1,k); % Upper Hessenberg matrix

  Q(:,1) = b/norm(b);
  for j = 1:k

    % Get a vector in the next subspace (and its norm)
    Q(:,j+1) = A*Q(:,j);
    norma = norm(Q(:,j+1));

    % Modified Gram-Schmidt (standard Arnoldi)
    for l = 1:j
      H(l,j) = Q(:,l)'*Q(:,j+1);
      Q(:,j+1) = Q(:,j+1)-Q(:,l)*H(l,j);
    end
    H(j+1,j) = norm(Q(:,j+1));

    % Normalize final result
    Q(:,j+1) = Q(:,j+1)/H(j+1,j);

  end

end
```

Figure 1: The standard Arnoldi algorithm.

```matlab
% [Q,H] = arnoldi2(A,b)
%
% Compute an Arnoldi decomposition
%
%  A*Q(:,1:end-1) = Q*H
%
% where H is a k+1-by-k upper Hessenberg matrix and Q has
% orthonormal columns. We use MGS, and make a second
% re-orthogonalization pass if there is enough cancellation
% in the first pass.
%
function [Q,H] = arnoldi2(A,b,k)

  n = length(A);
  Q = zeros(n,k+1); % Orthonormal basis
  H = zeros(k+1,k); % Upper Hessenberg matrix
  alpha = 0.1;   % The "twice is enough" threshold

  Q(:,1) = b/norm(b);
  for j = 1:k

    % Get a vector in the next subspace (and its norm)
    Q(:,j+1) = A*Q(:,j);
    norma = norm(Q(:,j+1));

    % Modified Gram-Schmidt (standard Arnoldi)
    for l = 1:j
      H(l,j) = Q(:,l)'*Q(:,j+1);
      Q(:,j+1) = Q(:,j+1)-Q(:,l)*H(l,j);
    end
    H(j+1,j) = norm(Q(:,j+1));

    % The "twice is enough" second pass, if the residual is small
    if H(j+1,j) < alpha*norma
      for l = 1:j
        mu = Q(:,l)'*Q(:,j+1);
        Q(:,j+1) = Q(:,j+1)-Q(:,l)*mu;
        H(j,l) = H(j,l) + mu;
      end
      H(j+1,j) = norm(Q(:,j+1));
    end

    % Normalize final result
    Q(:,j+1) = Q(:,j+1)/H(j+1,j);

  end

end
```

Figure 2: The Arnoldi algorithm with re-orthogonalization.

interlaced multiply-and-orthogonalize process used to obtain the decomposition (Figure 1). Unfortunately, the modified Gram-Schmidt algorithm — though more stable than classical Gram-Schmidt! — is nonetheless unstable in general. The sore point occurs when we start with a matrix that lies too near the span of the vectors we orthogonalize against; in this case, by the time we finish orthogonalizing against previous vectors, we have cancelled away so much of the original vector that what is left may be substantially contaminated by roundoff. For this reason, the "twice is enough" reorthogonalization process of Kahan and Parlett is useful: this says that if the remainder after orthogonalization is too small compared to what we started with, then we should perhaps refine our computation by orthogonalizing the remainder again. We show this process in Figure 2.

## 2 Lanczos

Now suppose that $A$ is a symmetric matrix. In this case, the Arnoldi decomposition takes a special form: the upper Hessenberg matrix is now a symmetric upper Hessenber matrix (aka a tridiagonal), and we dub the resulting decomposition the *Lanczos* decomposition:

$$AQ_k = Q_{k+1}\bar{T}_k, \quad T_k = \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \beta_{k-2} & \alpha_{k-1} & \beta_{k-1} \\ & & & & \beta_{k-1} & \alpha_k \\ & & & & & \beta_k \end{bmatrix}$$

The Lanczos algorithm is the specialization of the Arnoldi algorithm to the symmetric case. In exact arithmetic, the tridiagonal form of the coefficient matrix allows us to do only a constant amount of orthogonalization work at each step (Figure 3).

Sadly, the Lanczos algorithm in floating point behaves rather differently from the algorithm in exact arithmetic. In particular, the iteration tends to "restart" periodically as the space starts to get very good approximations of eigenvectors. One can deal with this via full reorthogonalization, as with the Arnoldi iteration; but then the method loses the luster of low cost, as

```
% [Q,alpha,beta] = lanczos(A,b)
%
% Compute an Lanczos decomposition
%
%  A*Q(:,1:end-1) = Q*T
%
% where T is a k+1-by-k tridiagonal matrix with diagonal
% entries alpha and super/subdiagonals beta, and Q has
% orthonormal columns.
%
function [Q,H] = lanczos(A,b,k)

  n = length(A);
  Q = zeros(n,k+1); % Orthonormal basis
  alpha = zeros(k,1);
  beta = zeros(k,1);

  Q(:,1) = b/norm(b);
  for j = 1:k
    Q(:,j+1) = A*Q(:,j);
    alpha(j) = Q(:,j)'*Q(:,j+1);
    Q(:,j+1) = Q(:,j+1)-alpha(j)*Q(:,j);
    if j > 1
      Q(:,j+1) = Q(:,j+1)-beta(j-1)*Q(:,j-1);
    end
    beta(j) = norm(Q(:,j+1));
    Q(:,j+1) = Q(:,j+1)/beta(j);
  end

end
```

Figure 3: Lanczos iteration. Note that in floating point, the columns of $Q$ will lose orthogonality.

we have to orthogonalize against several vetors periodically. An alternate *selective orthogonalization* strategy proposed by Parlett and Scott lets us orthogonalize only against a few previous vectors, which are associated with converged eigenvector approximations (Ritz vectors). But, as we shall see, such orthogonalization is mostly useful when we want to solve eigenvalue problems. For linear systems, it tends not to be necessary.