Course Notes: Week 1
Math 270C: Applied Numerical Linear Algebra

1 Lecture 1: Introduction (3/28/11)

We will focus on iterative methods for solving linear systems of equations (and some discussion of eigenvalues problems):

\[ Ax = b, \quad Ax = \lambda x \quad A \in \mathbb{R}^{n \times n} \]

where \( n \) is very large (e.g. \( n = 10^9 \) on your (powerful) desktop PC) and typically \( A \) is sparse. In other words, we want to obtain \( x \) from \( A \) and \( b \) (and \( \lambda \) for eigenvalue problems) but not necessarily \( A^{-1} \). We will also be focusing a great deal on the types of matrices that arise in numerical methods for partial differential equations (e.g. finite difference, finite volume methods etc.). However, I am assuming no familiarity with such techniques. Mainly they will provide examples for testing the algorithms we explore, but those algorithms will of course be applicable for all matrices (or at least a broad class such as symmetric matrices or symmetric positive definite matrices depending on the method we focus on).

Iterative methods are potentially much faster than direct methods (e.g. Gaussian elimination, QR factorization etc., see Math 270B for this class of methods). Notably, when the matrix is sparse we may be able to solve in linear time: \( O(n) \) (and quadratic in general: \( O(n^2) \)). This is a very appealing possibility considering the direct methods are always \( O(n^2) \) and \( O(n^3) \) for sparse and non-sparse. Furthermore, the direct methods typically incur \( O(n^2) \) storage in both the sparse and non-sparse cases whereas iterative methods will typically only require storage of \( A \) (i.e. \( O(n) \) storage for sparse and \( O(n^2) \) for dense). The analysis of iterative methods is in general much more involved. Also it can require great care to achieve linear performance in practice. Generally, the iterative method must be designed with some knowledge of the matrices it is designed for. This aspect is less appealing than the direct methods, but the potential for linear performance more than justifies the effort.

**Sparsity** means that each row in the matrix \( A \) has at most a \( c \) non-zero entries where \( c \ll n \). E.g. \( c = 3 \) for the matrix:

\[
A = \begin{pmatrix}
2 & -1 & 0 & 0 & \ldots & 0 \\
-1 & 2 & -1 & 0 & \ldots & 0 \\
0 & -1 & 2 & -1 & \ldots & 0 \\
0 & \ldots & 0 & -1 & 2
\end{pmatrix}
\]

Basically, the matrix is mostly zero so we can evaluate matrix vector multiplies in linear time (since each row can be dotted with an \( \mathbb{R}^n \) vector in \( (2c-1)n \) flops. The idea is that with (functional) iterative methods we should be able to get a very good approximation to \( x = A^{-1}b \) by some low number of matrix vector multiplies being the dominant cost in the algorithm. This may sound too good to be true (and sometimes it is). However, this is the promise of iterative methods for sparse matrices and it is very often possible to achieve this performance.

1.1 Geometric Convergence

Trefethen and Bau refer to “geometric convergence” as the optimal case for an iterative method (but exponential convergence is also often possible). In these scenarios, a constant number of iterations
is required in practice to reduce the residual to the system by a given factor (independent of the size of the problem \( n \)). In practice, my criterion for optimal convergence is that the residual is reduced by a constant factor each iteration. Let \( x^k \) be the \( k \)th iterate in our method. Therefore, the residual at this step is \( r^k = b - Ax^k \). In general, the residual will be our only means of estimating the accuracy of the \( k \)th iterate. Also, in general we hope that \(|r^k|_2 < |r^{k+1}|_2 < |r^{k+2}|_2\). When I am estimating the efficacy of a given iterative method I usually use the following formula:

\[
|r^{k+1}|_2 = \rho |r^k|_2
\]

and estimate \( \rho \) asymptotically. I.e. as \( k \to \infty \) I estimate \( \rho \) (numerically). In this case our asymptotic performance is \(|r^k|_2 = Ce^{\gamma k} \) with \( \rho = e^\gamma \) and \( \gamma < 0 \). In practice you can plot \( \log(|r^k|_2) \) vs. \( k \) and estimate the slope of the line.

### 1.2 Examples

The two most simplistic iterative methods are probably Jacobi and Gauss-Seidel. These methods are very simple to describe, but typically do not yield optimal performance. I.e. they usually require many many iterations and the the asymptotic \( \rho \) discussed above is 1. I.e. the method nearly stagnates as you run it longer and longer (typically without bringing you close enough to the solution). These methods are still very useful, just not typically on their own. For example, both of these techniques can be shown to “smooth” the error very efficiently for many matrices. That is, they reduce the error of the high frequency modes very quickly but reduce the error of the low frequency modes very slowly. This fact is leveraged in multigrid methods for discrete elliptic equations.

Not that in both examples below, all operations can be made much more efficient in the case of sparse matrices. Matrix vector multiplies and residual updates can all be done with sparsity in mind to make each iteration of both methods linear in computational cost (linear in the size of the problem \( n \)).

#### 1.2.1 Jacobi

Jacobi iteration is a very simple algorithm for approximating \( x = A^{-1}b \). In practice, I would describe Jacobi iteration as:

```for k = 1 to max_it do
    // At this point, x is x^k
    r = b - Ax
    for i = 1 to n do
        δ ← r_i / A_{ii}
        x_i ← x_i + δ //Note that each x_i update can be done independently
    end for
    // At this point, x is now equal to x^{k+1}
end for```

This can also be described as:

\[
M^J x^{k+1} = N^J x^k + b
\]
where \( M^J \) is the diagonal part of \( A \) (\( D \)) and \( N^J = -(L+U) \) and \( L \) and \( U \) are the lower and upper triangular parts of the matrix \( A \). E.g. for

\[
A = \begin{pmatrix}
2 & -1 & 0 & 0 & \ldots & 0 \\
-1 & 2 & -1 & 0 & \ldots & 0 \\
0 & -1 & 2 & -1 & \ldots & 0 \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & 2 & -1 & 2
\end{pmatrix}
\]

\[
M^J = \begin{pmatrix}
2 & 0 & 0 & 0 & \ldots & 0 \\
0 & 2 & 0 & 0 & \ldots & 0 \\
0 & 0 & 2 & 0 & \ldots & 0 \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & 0 & 0 & 2
\end{pmatrix}, \quad L = \begin{pmatrix}
0 & 0 & 0 & 0 & \ldots & 0 \\
-1 & 0 & 0 & 0 & \ldots & 0 \\
0 & -1 & 0 & 0 & \ldots & 0 \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & -1 & 0 & 0
\end{pmatrix}, \quad U = \begin{pmatrix}
0 & -1 & 0 & 0 & \ldots & 0 \\
0 & 0 & -1 & 0 & \ldots & 0 \\
0 & 0 & 0 & -1 & 0 & \ldots & 0 \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & 0 & 0 & 0
\end{pmatrix}
\]

### 1.2.2 Gauss-Seidel

Gauss-Seidel usually performs better than Jacobi and is a little more expensive than Jacobi. Also, the order in which you apply a correction matters for Gauss-Seidel and individual updates are not independent. Jacobi is usually considered more parallel friendly than Gauss-Seidel for this reason.

The Gauss-Seidel iteration can be described as follows:

```plaintext
for k = 1 to max_it do
    // At this point, x is x^k
    for i = 1 to n do
        \( \delta \leftarrow \frac{r_i}{A_{ii}} \)
        \( x_i \leftarrow x_i + \delta \)
        \( r = b - Ax \) //Update the residual to reflect our change to \( x_i \)
    end for
    // At this point, x is now equal to x^{k+1}
end for
```

This can also be described as:

\[
M^G x^{k+1} = N^G x^k + b
\]

where \( M^G = D + L \) and \( N^G = -U \).

## 2 Lecture 2: Krylov Subspaces and Arnoldi Iteration (3/31/11)

Krylov subspace methods are very powerful techniques that often achieve near optimal performance. That is, these often have the potential of requiring just a few iterations independent of the size of the problem \( n \). Furthermore, they may be optimized for different classes of matrices. E.g. if the matrix is symmetric positive definite, the Krylov method is conjugate gradients (CG). If is just symmetric the method is MINRES. In general it is GMRES. CG is the most efficient in terms of memory and computation, MINRES second most so and GMRES least so. We will initially discuss the general case before we cover the specializations.

These methods are based on the Krylov subspaces \( K^k = \text{span}\{b, Ab, A^2b, ..., A^{k-1}b\} \). These
spaces have the property $\mathcal{K}^k \subset \mathcal{K}^{k'} \forall k' > k$ and $\exists k$ with $A^{-1}b \in \mathcal{K}^k$. The first property is trivial, the second property follows because (assuming $A$ is full rank) we can write:

$$A^{-1} = \sum_{i=0}^{k-1} c_i A^i.$$ 

To see this, we can construct a polynomial for which $A$ is a root. For example, consider the case of a diagonalizeable $A$. Let $A = Q\Lambda Q^{-1}$ where $\Lambda$ is a diagonal matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. Let $\lambda_1, \lambda_2, \ldots, \lambda_k$ be all of the eigenvalues in $\Lambda$ except for any repeats (i.e. just the distinct eigenvalues). Then $A$ is a root of the following $k$th degree polynomial:

$$p(\lambda) = (\lambda - \lambda_1)(\lambda - \lambda_2) \cdots (\lambda - \lambda_k) = \sum_{i=0}^{k-1} b_i \lambda^i + \lambda^k$$

where $k$ is then the number of distinct eigenvalues. Since $A$ is a root of this polynomial

$$A^{-1} = -\frac{1}{b_0} \left( \sum_{i=1}^{k-1} b_i A^{i-1} + A^{k-1} \right).$$

Here $b_0$ is the product of the distinct eigenvalues and thus will not be zero in the full rank case. Therefore, when $A$ is full rank, the solution $x = A^{-1}b$ will be in the $k$th Krylov subspace $\mathcal{K}^k$ where $k$ is the number of distinct eigenvalues.

In the general case, we can create a polynomial

$$p(\lambda) = (\lambda - \lambda_1)^{d_1} (\lambda - \lambda_2)^{d_2} \cdots (\lambda - \lambda_k)^{d_k} = \sum_{i=0}^{\hat{k}-1} b_i \lambda^i + \lambda^\hat{k}$$

where $d_k$ is the size of the largest Jordan block that eigenvalue $\lambda_k$ appears in and $\hat{k} = \sum_{i=1}^{k} d_i$. In that case, the solution $x = A^{-1}b$ will be in the $\hat{k}$th Krylov subspace $\mathcal{K}^{\hat{k}}$.

### 2.1 Krylov Subspace Iterative Methods

Krylov methods generate the $k$th iterate in our method ($x^k$) by choosing the best (in the least squares sense) vector in $\mathcal{K}^k$ at solving $Ax = b$. Letting $Q^k = [q_1, q_2, \ldots, q_k] \in \mathbb{R}^{n \times k}$ be an orthonormal basis for $\mathcal{K}^k$, we can write the $k$th iterate in the Krylov method as:

$$w^k \in \mathbb{R}^k = \arg\min ||b - AQ^k w^k||_2, \quad x^k = Q^k w^k.$$ 

Given that we know the solution will be in some Krylov space, we know that this iterative process will terminate. Thus, you might not want to call this an iterative method at all. However, we will see that in general we will not solve this least squares problem exactly and furthermore we hope that the error decreases as we move from each Krylov space to the next rapidly enough that we do not need to continue until we have reached a number of iterations equal to the degree of the polynomials just discussed. Note that in general we will need to generate an orthogonal basis $Q^k$ for $\mathcal{K}^k$ because the vectors $A^k b$ are very ill-conditioned in the sense that they are very nearly linearly dependent as $k$ becomes large (namely, they all start to resemble the dominant eigenvectors). The process that we use to generate this basis is called Arnoldi iteration. It is effectively Graham-Schmidt run on an informed choice of vectors and can even be used to estimate the eigenvalues of the matrix $A$. 
2.2 Arnoldi Iteration

This iterative approach to defining the basis $Q^k$ for $K^k$ leverages the fact that we have done most of the work generating a basis in the previous iteration. That is, $K^k \subset K^{k+1} \subset K^{k+2}$ and so on, thus $Q^{k+1} = [Q^k, q_{k+1}]$. I.e. at each iteration, we only need to generate one new vector ($q_{k+1}$) when we are constructing the next basis $Q^{k+1}$. Since the most obvious vector in $K^{k+1}$ that is not in $K^k$ is $A^k b$ it would seem that adding in $A^k b$ and then “Graham-Schmidting” the components away that are not orthogonal to the previous $q_i$s would be a good idea. However, we want to avoid computing $A^k b$ since it is numerically not very stable (i.e. roundoff will make it hard to distinguish between $A^k b$ and $A^{k+1} b$ for large $k$). Instead, Arnoldi will use $A q_k$. That is, Arnoldi iteration starts with

$$Q^0 = \left[ \frac{b}{\|b\|_2} \right]$$

then, in general

$$q_{k+1} = \frac{v_{k+1}}{\|v_{k+1}\|_2}, \quad v_{k+1} = A q_k - h_{1k} q_1 - h_{2k} q_2 - \ldots - h_{kk} q_k$$

where the $h_{ik}$ are chosen to guarantee that $q_{k+1}$ is orthogonal to $q_i \forall i < k + 1$ (this is the Graham-Schmidt part). In other words, we have

$$h_{ik} = q_i \cdot (A q_k).$$

Basically, we take the previous vectors $q_i$, we throw in the new vector $A q_k$ and then we make a new basis for the span of those $k + 1$ vectors by adding in $q_{k+1}$.

A few things are not completely clear here. First, why does it suffice to add $A q_k$ to yield a basis $Q^{k+1}$ that will span $K^{k+1}$? Second, how do we know $v_{k+1}$ is never zero in practice.

3 Lecture 3: Properties of Arnoldi Iteration (4/1/11)

The Arnoldi iteration is convenient in that it provides a basis for $K^{k+1}$ given a basis for $K^k$ with only the addition of one new search direction $q^{k+1}$ and it avoids the need to use the extremely ill-conditioned $A^k b$. We will see that this iteration also reduces the least squares problem to one that can be naturally treated with Givens rotations updates to the solution at the previous iteration. In other words, the Arnoldi iteration lets us avoid solving a least squares problem “from scratch” just like it let us avoid running Graham-Schmidt “from scratch”. First, I will discuss the ability of $Q^{k+1}$ to span $K^{k+1}$ and second I will discuss why $v_{k+1}$ used to generate $Q^{k+1}$ is never zero in practice.

3.1 Why $v_{k+1} \neq 0$ in Practice

Assume $Q^k$ spans $K^k$ and that $x = A^{-1} b$ is not in $K^k$ (otherwise the Krylov iteration would have terminated in the last iteration. It is trivial to see that $Q^1$ spans $K^1$, so it will be enough to show that $v_{k+1} \neq 0$ assuming $Q^k$ spans $K^k$ and that $x = A^{-1} b$ is not in $K^k$.

The formula for $v_{k+1}$ is

$$v_{k+1} = A q_k - h_{1k} q_1 - h_{2k} q_2 - \ldots - h_{kk} q_k$$
if \( v_{k+1} \) were zero then \( Aq_k \in \mathcal{K}^k \). Now,

\[
A^k b = A \left( A^{k-1} b \right) = A \left( \sum_{i=1}^{k} \alpha_i q_i \right)
\]

since we are assuming here that \( Q^k \) spans \( \mathcal{K}^k \). Also, \( Aq_i = \sum_{j=1}^{k} \beta_{ij} q_j \) for \( i = 1, 2, \ldots, k-1 \) since \( \mathcal{K}^{k-1} \subset \mathcal{K}^k \). Therefore,

\[
A^k b = \alpha_k Aq_k + \sum_{i=1}^{k} \gamma_i q_i.
\]

Therefore, if \( Aq_k \in \mathcal{K}^k \) then \( A^k b \in \mathcal{K}^k \) which would then imply that \( \mathcal{K}^k = \mathcal{K}^k \forall \hat{k} > k \) and this can’t happen because we’ve assumed \( x = A^{-1} b \) is not in \( \mathcal{K}^k \) and we know it is in some Krylov space. Therefore, \( v_{k+1} \neq 0 \) in practice because we would have stopped the iteration at the previous time step if it were.

### 3.2 \( Q^k \) Spans \( \mathcal{K}^k \)

We’ll look at this with a quick inductive argument as well. It is trivially true for \( k = 1 \). Again, assume \( Q^k \) spans \( \mathcal{K}^k \) and that \( x = A^{-1} b \) is not in \( \mathcal{K}^k \) (because again otherwise we would have stopped). As pointed out above

\[
A^k b = \alpha_k Aq_k + \sum_{i=1}^{k} \gamma_i q_i.
\]

Now, if \( \alpha_k \) were zero then we would have stopped at the previous iteration because that would have implied that \( \mathcal{K}^k = \mathcal{K}^k \forall \hat{k} > k \) as above. Therefore,

\[
Aq_k = \frac{1}{\alpha_k} \left( A^k b - \sum_{i=1}^{k} \gamma_i q_i \right)
\]

and therefore \( q_{k+1} \) is a linear combination of vectors in \( \mathcal{K}^k \) one other vector: \( A^k b \). Therefore, \( Q^{k+1} \) spans \( \mathcal{K}^{k+1} \).

### 3.3 Upper Hessenberg Least Squares

The Arnoldi iterations yields the following recursive relationship between the search directions

\[
Aq_k = \sum_{i=1}^{k+1} h_{ik} q_i, \quad k = 1, 2, \ldots, n - 1
\]

In other words, \( AQ^k = Q^{k+1} H^k \) for the upper Hessenberg \( H^k \in \mathbb{R}^{k+1 \times k} \)

\[
A = \begin{pmatrix}
  h_{11} & h_{12} & \cdots & h_{1k} \\
  h_{21} & h_{22} & \cdots & \vdots \\
  0 & h_{32} & \cdots & \ddots \\
  \vdots & 0 & \ddots & \ddots \\
  0 & \ldots & 0 & h_{k+1k}
\end{pmatrix}
\]