1 Lecture 6: Conjugate Gradients (Lanczos Version)

The Arnoldi iteration is called Lanczos iteration in the case of a symmetric matrix $A$. We will go a step further in the next few lectures and assume that $A$ is symmetric positive definite:

$$A = A^T, \quad x^T A x > 0 \quad \forall x \neq 0$$

In this case, the Krylov method for solving $Ax = b$ is called Conjugate Gradients. This method is much more simplistic than the GMRES algorithm and does not require the excessive storage of all previous Arnoldi (here Lanczos) vectors $q_i$. Also, the method does not require the increasing number of inner products necessary for constructing $H^k$. We can derive this method using what we have covered thus far for GMRES.

The first thing to note is that if we define $T^k$ as $(Q^k)^T A Q^k$ where $Q^k \in \mathbb{R}^{n \times k}$ are the Lanczos (formerly Arnoldi) vectors, then $T^k = (T^k)^T$ since $A$ is symmetric and since this is essentially the upper Hessenberg matrix from the GMRES discussion, we can see that $T^k$ is symmetric tridiagonal:

$$T^k = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \ldots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ldots & 0 \\ 0 & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & 0 & \beta_{k-1} & \alpha_k \end{pmatrix}$$

The addition of zeros above the diagonal that results from the symmetric upper Hessenberg nature will save us the inner product calculations needed in the Graham-Schmidt stage of the GMRES process. However, we will see that we can go even further and avoid storage of $Q^k$ altogether. Again, this is noteworthy because this increasingly excessive storage (and the excessive cost of the inner products) that arises when the number of iterations is large are the primary drawbacks of the GMRES algorithm.

In general for the Krylov methods with Lanczos/Arnoldi bases we write $x^k = Q^k \lambda^k$. It would be nice if we could write this as $x^k = x^{k-1} + \lambda_k^k q_k$ and then only require the generation/storage of the newest Lanczos vector. Unfortunately, for both CG and GMRES, this will not be possible. However, for CG we will be able to write $x^k = x^{k-1} + \alpha_k p_k$ in terms of a different basis for the Krylov space. This modification to the algorithm will be what allows us to avoid the storage of the increasingly costly $Q^k$. We can derive this new basis by modifying the functional that we minimize at each iteration (e.g. for GMRES we minimized the $L^2$ norm of the residual over the $k^{th}$ Krylov space) to yield a symmetric positive definite tridiagonal solve at each iteration that we can show leads to the basis vectors $p_k$. I will call this the Lanczos derivation of CG. We will also do steepest descent inspired derivation in the following lectures.

1.1 Minimizing the $A$ Norm of the Error

When $A$ is symmetric positive definite, it defines a norm:

$$\|x\|_A = \sqrt{x^T A x}.$$
Most importantly for us, it will be easy to evaluate the $A$-norm of the error. Specifically, we will minimize the $A$-norm over the Krylov space at each iteration. This will be more useful than simply minimizing the $L^2$ norm of the residual (which is equivalent to minimizing the $A^T A$ norm of the error).

The error at step $k$ is $e^k = x - x^k$. The residual is related to the error as $r^k = Ae^k$. We can see the useful equality:

$$||e^k||_A^2 = (e^k)^T A e^k = x^T b - 2x^k b + x^k A x^k = x^T b + 2 \phi(x^k)$$

where $\phi(x) = x^T b + \frac{1}{2} x^T A x$. So technically in order to know the $A$-norm of the error we would need to know $x^T b$ which we cannot know. However, the above expression shows that the $A$-norm of the error has the same minimizer as $\phi$. Therefore, we can choose our $k^{th}$ iterate to minimize $\phi$ rather than the $L^2$ norm of the residual. We will see that this leads to more favorable properties of the iterative scheme (namely that we do not have to store the $Q_k$).

In order to find $x^k$ as the minimizer of $\phi$ over $K^k$, we can use the Lanczos basis and write $x^k = Q_k \lambda^k$ where $\lambda^k$ solves $\nabla_\lambda f^k(\lambda) = 0$ with $f^k(\lambda) = \phi(Q_k \lambda)$. Therefore, $\lambda^k$ is the solution of

$$\nabla_\lambda f^k(\lambda) = (Q_k)^T A Q_k \lambda^k - (Q_k)^T b = T^k - \hat{b}^k.$$  

This is particularly useful because it is relatively easy to solve (via $LDL^T$ factorization) a symmetric positive definite tridiagonal system. Notably, it is very difficult to solve this system (via factorization) if the matrix is symmetric but indefinite (i.e. if $A$ has both negative and positive eigenvalues). In other words, this is really only going to be possible for the symmetric positive definite case (even though $T^k$ is tridiagonal even in the case of symmetric indefinite).

## 2 Lecture 7: Tridiagonal Factorization Properties

### 2.1 Tridiagonal Symmetric Positive Definite $LDL^T$

The $LDL^T$ factorization of $T^k$ can be determined as

$$L^k = \begin{pmatrix} 1 & & \\ \mu_1 & 1 & \\ & \ddots & \ddots \\ & & \mu_{k-2} & 1 \end{pmatrix}, \quad D^k = \begin{pmatrix} d_1 & & \\ & d_2 & \\ & & \ddots \\ & & & d_k \end{pmatrix}, \quad T^k = L^k D^k (L^k)^T.$$  

If you just multiply these out you can see that the $\mu_i$ and $d_i$ can be determined as

\[
\begin{align*}
d_1 &\leftarrow \alpha_1 \\
\mu_1 &\leftarrow \frac{\beta_1}{d_1} \\
\text{for } i = 2 \text{ to } k &\text{ do} \\
d_i &\leftarrow \alpha_i - \mu_{i-1} \beta_{i-1} \\
\mu_i &\leftarrow \frac{\beta_i}{d_i} \\
\text{end for}
\end{align*}
\]

In general, the sensitivity of this procedure to roundoff errors will be proportionate to the condition number of $A$. Also, this process can fail catastrophically if $A$ is symmetric indefinite. Of course, $\beta_i$ and $\alpha_i$ do not change when $k > i$ so we will only need to compute $d_k$ and $\mu_{k-1}$ at the $k^{th}$ iteration.
2.2 Forward and Backward Substitution

The \( LDL^T \) factorization is useful because it facilitates forward and backward substitution based solution for \( \lambda^k \). To first do the forward substitution, we use \( p^k = (L^k)^T \lambda^k \) and solve \( L^k D^k p^k = b^k \):

\[
\begin{align*}
s &\leftarrow \|b\|_2; \\
p_1^k &\leftarrow \frac{s}{d_1}; \\
\text{for } i = 2 \text{ to } k \text{ do} &\quad s \leftarrow s \mu_{i-1}; \\
&\quad p_i^k \leftarrow (-1)^{i+1} \frac{s}{d_i}; \\
\text{end for}
\end{align*}
\]

Of course, at iteration \( k \) all the \( p_i^k \) for \( i = 1, \ldots, k - 1 \) are the same as from the previous iteration so the only new data is \( p_k^k \). So to sum up, at the \( k \)th iteration you need to calculate:

\[
d_k = \alpha_k - \mu_{k-1} \beta_{k-1}, \quad \mu_k = \frac{\beta_k}{d_k}, \quad s = s \mu_{k-1}, \quad p_k^k = (-1)^{k+1} \frac{s}{d_k}
\]

Unfortunately, although the \( p^k \) only change in the last component at each iteration, the \( \lambda^k \) will be completely different at each iteration. We can see this from the expression

\[
L^k \lambda^k = \begin{pmatrix} 1 & \mu_1 \\ & 1 & \mu_2 \\ & & \ddots & \ddots \\ & & & 1 & \mu_{k-1} \\ & & & & 1 \end{pmatrix} \begin{pmatrix} \lambda_1^k \\ \lambda_2^k \\ \vdots \\ \vdots \\ \lambda_k^k \end{pmatrix} = \begin{pmatrix} p_1^k \\ p_2^k \\ \vdots \\ \vdots \\ p_k^k \end{pmatrix}
\]

that although the \( p_k^k \) is the only new piece of information in the right hand side of the equation it will unfortunately affect each \( \lambda_i^k \) in the back substitution process. Unlike the forward substitution for the \( p_i^k \) where information propagates from small \( i \) to large, the back substitution process for the \( \lambda_i^k \) will propagate information from large \( i \) to small. And since the change in the right hand side is at \( p_k^k \), every \( \lambda^k \) will unfortunately be affected. This means that expressing the \( k \)th iterate in terms of the Lanczos vectors \( (x^k = Q^k \lambda^k) \) requires that we store all previous vectors which we will see is suboptimal for symmetric positive definite matrices.

3 Lecture 8: Memory Efficient Basis for the Krylov Space

The insight needed to yield an algorithm that does not require the storage of all prior search directions is basically related to the derivation of a new basis for the Krylov space \( K^k \). We will not see this completely until we go through the steepest descent derivation of CG, but we will derive the basis vectors now and prove their properties later. The hope for avoiding the storage of the search directions is hinted at in the tridiagonal structure of conjugation of the matrix with the Lanczos basis for \( K^k \). Unfortunately, the behavior of the \( \lambda^k \) that we just discussed suggests that we need some other basis for representing \( x^k \) over \( K^k \). These basis vectors \( (C^k = [c_1^k, c_2^k, \ldots, c_k^k] \in R^{n \times k}) \) are given as

\[
C^k = Q^k (L^k)^{-T}
\]

We can then see that \( x^k = Q^k \lambda^k = C^k p^k = [c_1^k, c_2^k, \ldots, c_{k-1}^k] p^{k-1} + p_k^k c_k^k \). We have already shown that \( p^k \) only changes in the last component (i.e. that \( p^{k-1} \) is from the previous iteration). Therefore, if \( [c_1^k, c_2^k, \ldots, c_{k-1}^k] = C^{k-1} \) then we can see that \( x^k = x^{k-1} + p_k^k c_k^k \). We will show this here.
Later we will show that \( \text{span} \{c_1, \ldots, c_k\} = K^k \) and so we can think of the vectors \( \{c_1, \ldots, c_k\} \) as a more useful basis for the Krylov space in that they allow us to avoid excessive storage.

The following equality suggests that only the last column of \( C^k \) changes at the \( k^{\text{th}} \) iteration:

\[
C^k(L^k)^T = \begin{bmatrix}
c_1^k, \mu_1 c_1^k + c_2^k, \mu_2 c_2^k + c_3^k, \ldots, \mu_{k-1} c_{k-1}^k + c_k^k
\end{bmatrix} = Q^k
\]

In other words, \( c_i^k = q_1, c_i^k = q_i - \mu_{i-1} c_{i-1}^k \) for \( i = 2, \ldots, k \).

Therefore at the \( k^{\text{th}} \) iteration, the \( c_i^k \) for \( i = 1, \ldots, k = 1 \) are all the same as the previous iteration. Thus we can drop the super script \( k \) and just call these \( c_i \) and most importantly we can say that

\[
x^k = x^{k-1} + p_k^k c_k
\]

and that \( x^k \) is the minimizer of \( \phi \) over \( K^k \) without requiring storage of all previous Lanczos vectors.

### 3.1 Wrapping Up: Lanczos CG Pseudocode

Here I will outline a version of the CG algorithm suggested by the results of the previous three lectures.

\[
\begin{align*}
v & \leftarrow b; \\
\beta_{k-1} & \leftarrow 0; \\
\beta_k & \leftarrow ||v||_2; \\
q_k & \leftarrow 0; \\
\text{for } k = 1 \text{ to } k_{\text{max}} \text{ do} \\
& \quad q_{k-1} \leftarrow q_k; \\
& \quad q_k \leftarrow \frac{1}{\beta_k} v; \\
& \quad v \leftarrow Aq_k; \\
& \quad \alpha_k \leftarrow q_k^T v; \\
& \quad v \leftarrow v - \alpha_k q_k - \beta_{k-1} q_{k-1}; \\
& \quad \beta_{k-1} \leftarrow \beta_k; \\
& \quad \beta_k \leftarrow ||v||_2; \\
& \quad \text{if } k\text{==1 then} \\
& \quad \quad d_k \leftarrow \alpha_k; \\
& \quad \quad c_k \leftarrow q_k; \\
& \quad \quad \rho_k \leftarrow \frac{\beta_{k-1}}{\alpha_k}; \\
& \quad \quad x \leftarrow \rho_k q_k; \\
& \quad \text{else} \\
& \quad \quad d_{k-1} \leftarrow d_k; \\
& \quad \quad \rho_{k-1} \leftarrow \rho_k \\
& \quad \quad c_{k-1} \leftarrow c_k \\
& \quad \quad \mu_{k-1} \leftarrow \frac{\beta_{k-1}}{d_{k-1}^2}; \\
& \quad \quad d_k \leftarrow \alpha_k - \beta_{k-1} \mu_{k-1}; \\
& \quad \quad c_k \leftarrow q_k - \mu_{k-1} c_{k-1} \\
& \quad \quad \rho_k \leftarrow \frac{\mu_{k-1} d_{k-1} \rho_{k-1}}{d_k}; \\
& \quad \quad x \leftarrow x + \rho_k c_k \\
& \quad \text{end if} \\
& \text{end for}
\end{align*}
\]