Investigating the Effects of MINRES with Local Reorthogonalization

Danielle C. Maddix
CME 338: Large Scale Optimization
Final Project

June 5, 2015

1 Introduction

The minimum residual method, MINRES, is an iterative method for solving a n x n linear system, $Ax = b$, where is A is a symmetric matrix. It searches for a vector, x_k, in the k^{th} Krylov subspace that minimizes the residual, $r_k = b - Ax_k$. Due to the symmetric nature, the vectors are computed to be the Lanczos vectors, which simplifies to a three-term recurrence, where v_{k+1} is computed as a linear combination of the previous two Lanczos vectors v_k and v_{k-1}. To update the new vector in the $(k+1)^{th}$ Krylov subspace, a QR decomposition is maintained and each iteration, we just must compute a new plane or Givens rotation [3]. In exact arithmetic, all the Lanczos vectors are orthogonal to each other. However, the problem that arises is numerically, this orthogonality is not maintained.

This is contrasted by the generalized minimum residual method, GMRES, which is an iterative method for solving the same linear system that minimizes the residual, except now A may be an unsymmetric matrix. Here, the new vector v_{k+1} in the orthogonal basis for the Krylov subspace is calculated in the Arnoldi recurrence process, which requires storing all the previous vectors. Thus, GMRES requires more storage and work than MINRES does. However, some users prefer to use GMRES with a good preconditioner, M, even on symmetric systems, even though MINRES is designed to take advantage of the symmetric properties. The reason for this being that since GMRES uses a modified Gram-Schmidt orthogonalization in the Arnoldi process, the vectors in the basis, V_k, for the k^{th} Krylov subspace remain orthogonal and so do not lose the numerical orthogonality property as happens in MINRES. Thus, fewer total iterations may be required of GMRES on the symmetric saddle-point systems than MINRES. Therefore, we investigate a process to reorthogonalize the vectors in MINRES, by storing an input parameter, $localSize$ of them and explicitly making the new v_{k+1} orthogonal to the previous $localSize$ Lanczos vectors. The theory that we will test is that this will decrease the number of iterations and make MINRES the more preferable method on symmetric systems, as it was originally designed.

2 Implementation

To add local reorthogonalization to MINRES, we follow the procedure of the local reorthogonalization from the LSMR algorithm, which is a method to solve rectangular least squares systems using the Golub-Kahan process, while still minimizing the residual [1] . We first download the minres.m Matlab file from the SOL website [3]. An additional input and output feature have been added. The additional output, $resvec$, is a vector, which simply stores the residual at each iteration for plotting purposes to be shown in the next section. The additional input is the $localSize$ parameter. Setting $localSize = 0$ or [ ] results the regular MINRES algorithm without any reorthogonalization, whereas on the opposite end of the spectrum setting $localSize$ = Inf runs a full reorthogonalization of all of the Lanczos vectors. Of course, setting $localSize$

equal to a scalar is the integer number of prior Lanczos vectors stored. The modified function definition is given below:

```
function [ x, istop, itn, rnorm, Arnorm, Anorm, Acond, ynorm resvec] = ...
          minres( A, b, M, shift, show, check, itnlim, rtol, localSize )
```

In the initialization process, we add an initialization for the local reorthogonalization, using some helpful boolean variables, $localOrtho$ and $localVQueueFull$. $localOrtho$ is used to determine whether local reorthogonalization is turned on or off. It is initialized to false. It is set to be true, only if the $localSize$ input parameter is strictly larger than 0. In the case that this holds, $localPointer$ is initialized to 0, which will be used to mark the column of where the old v_k will be stored. Another boolean to be used is $localVQueueFull$ which tells whether the circular buffer, $localV$, that stores the $localSize$ number of the Lanczos vectors is full. This also gets initialized to false. Lastly, $localV$ is the described array initialized to be the zero matrix of size n x $localSize$, which stores these previous Lanczos vectors. This code excerpt is shown below:

```
%Initialization for local reorthogonalization
localOrtho = false; %boolean to tell whether localReOrtho is on based on the value of localSize
if localSize > 0
    localPointer = 0; %tells number of prior Lanczos vectors stored
    localOrtho = true; %turn on local reorthogonalization
    localVQueueFull = false; %boolean that tells whether have stored all prior localSize lanczos vectors
    %Preallocate storage for the number of the latest v_k's
    localV = zeros(n, min(localSize,n)); %can't store more then min dimension of the matrix, n
end
```

As in the LSMR code, we also add in two helpful nested functions, which make use of some of these booleans inside of the the minres main function [1]. The first such function given below is used to store the old v_k in the appropriate column. First it checks if the $localV$ circular buffer is full. If it is, we reset $localPointer$ to be 1 and set $localVQueueFull$ to be true. Otherwise, if $localPointer$ is less then $localSize$, we increment $localPointer$ by one. In both cases, we store v_k in the updated $localPointer$ column of $localV$. It is clear that this procedure stores the prior $localSize$ Lanczos vectors by first filling up the first $localSize$ columns and then overwriting the oldest vector in column one, the new oldest vector in column two and so on circularly.

```
%this function stores v into the circular buffer localV
function localVEnqueue(v)
    if localPointer < localSize %localPointer counts the number currently stored
        localPointer = localPointer + 1; %not full yet
    else
        localPointer = 1; %remain orthogonal to previous localSize so erase first one and continue circularly
        localVQueueFull = true; %set boolean to true for being full
    end
    localV(:, localPointer) = v; %store v in the column corresponding to localPointer
end %nested function localVEnqueue
```

The second nested function, as shown below, uses the vectors stored in $localV$ to perform the local reorthogonalization, by orthogonalizing v_{k+1} with respect to each of those vectors one by one in a modified Gram-Schmidt process. This ensures that v_{k+1} is numerically orthogonal to the prior $localSize$ Lanczos vectors. The iteration counter is determined by using the $localVQueueFull$ boolean variable. If it is set to true, we reorthogonalize with respect to all of the previous $localSize$ vectors. Otherwise, in the case at the beginning that we have less than $localSize$ vectors, we reorthogonalize up to the $localPointer$ number of Lanczos vectors. Note that no normalization is required here, since that explicitly happens in the Lanczos part of the code and so remains numerically, contrary to the orthogonalization.

```
%Perform local reorthogonalization of v
```

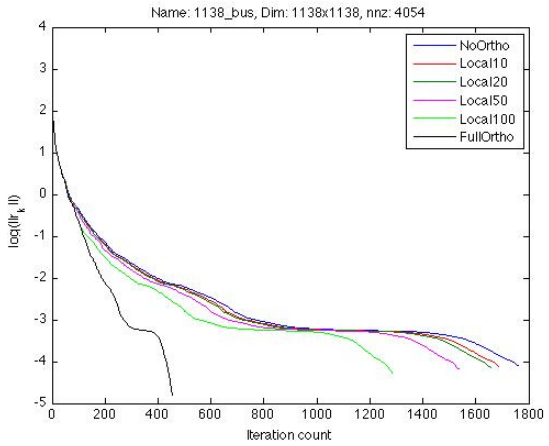
```
function vOutput = localVOrtho(v)
    vOutput = v;
    if localVQueueFull
        localOrthoLimit = localSize; %calculate where to terminate loop
    else
        localOrthoLimit = localPointer; %localPointer < localSize
    end
    for localOrthoCount = 1:localOrthoLimit
        vtemp = localV(:,localOrthoCount);
        %reorthogonalize 1 by 1
        vOutput = vOutput - (vOutput' * vtemp) * vtemp;
        %orthogonalize to each stored vector- note we don't have to normalize since
        %it is explicitly done in the code and so we don't need to redo it
    end
end %nested function localVOrtho
```

Lastly, we must call these nested functions within the Lanczos procedure. Once the prior v is normalized in y, it is stored, by calling *localVEnqueue*, only if the boolean *localOrtho* is true and that the local reorthogonalization process is activated. Then, the new non-normalized vector is updated using the Lanczos procedure. It is re-orthogonalized amongst the prior Lanczos vectors by calling *localVOrtho*, again only if the *localOrtho* boolean is set to be true. This Lanczos part of the code that is within the main while loop is shown below. Note that the Givens rotation update portion and the rest of the procedure to calculate x_{k+1} remains unchanged with the additional local reorthogonalization.

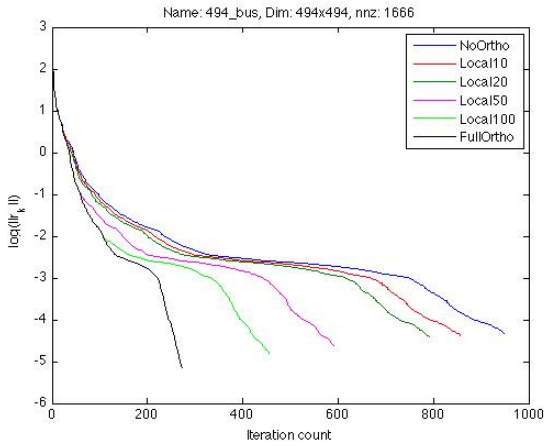
```
    s = 1/beta;                % Normalize previous vector (in y).
    v = s*y;                   % v = vk if P = I

    %if localOrtho turned on store old v for local reorthogonalization of new v
    if localOrtho
        localVEnqueue(v);
    end
    y = minresxxxA( A,v ) - shift*v; %shift is 0 otherwise solving A - shift*I
    if itn >= 2, y = y - (beta/oldb)*r1; end %normalization is the division r1 by oldb

    alfa   = v'*y;             % alphak
    y      = (- alfa/beta)*r2 + y; %normalization of r2/ beta = v
    if localOrtho
        % v will be normalized through y later- this is explicit
        % orthogonalizing it versus the previous localSize lanczos vectors
        y = localVOrtho(y);
    end
    r1     = r2; %r1 is unnormalized vold
    r2     = y; %r2 is unnormalized v
    if precon,  y = minresxxxM(M,r2);  end
    oldb   = beta;             % oldb = betak
    beta   = r2'*y;            % beta = betak+1^2
    if beta < 0, istop = 9;  break;  end
    beta   = sqrt(beta);
    tnorm2 = tnorm2 + alfa^2 + oldb^2 + beta^2;
```

3 Analysis and Results

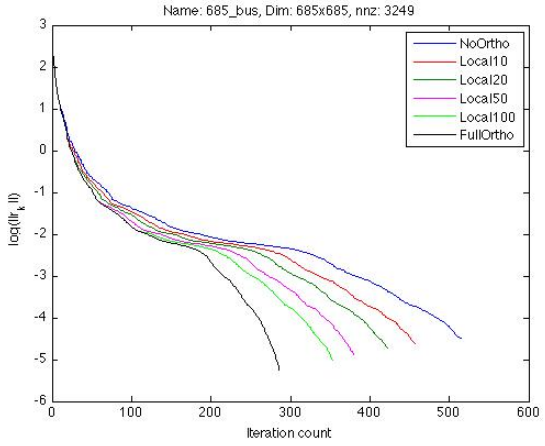
The effect of the reorthogonalization of MINRES is tested on matrices from Tim Davis' sparse matrix collection [2]. An additional function, namely, minresLocalReOrthoTest.m is defined for testing and plotting purposes. It loads one of the sparse matrix files and sets x = 1./(1:n)', where n = size(A,1) and then b = Ax so that a solution exists. There is no preconditioner, so M = [ ] and no shift operation, that is, shift = 0. We set the iteration limit to be n*5 and the tolerance to be 1.0e-10. Then, the minres function is called for various *localSize* parameters and the plots are shown in the figure below:



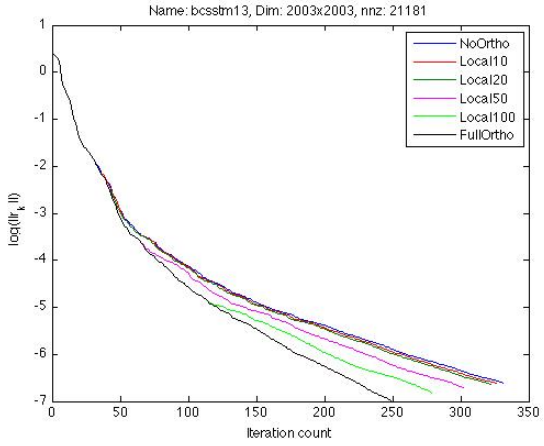
(a) real spd, id = 1



(b) real spd, id = 2



(c) real spd, id = 4



(d) real symmetric, $A \not\succ 0$, id = 68

Figure 1: Plot of the log of the residual versus the number of iterations displaying MINRES with local reorthogonalization of $localSize = 0, 20, 100, n$ vectors from V_k

It is clear from the all of the plots that as the $localSize$ parameter increases, the total number of iterations required decreases. All the matrices tested are clearly symmetric and hence square. The first three are positive definite matrices, $A \succ 0$. Figure 1.a shows that little speedup results from $localSize$ =10 and 20, whereas there is sufficient speedup from 50 and 100 vectors and a over 4x reduction of iterations from storing all the vectors. We see an analogous result in Figure 1.b, where storing 10 of the previous vectors has a partial speedup reducing the total iteration count by 100 and 20 has a partial speedup of reducing the iterations by 200, whereas storing 50 reduces the count by 400, 100 reduces the count by 500 and storing all the vectors reduces it by even more. Figure 1.c shows each increased $localSize$ storage reducing the iteration count by approximately 100 iterations. The last matrix tested on in Figure 1.d is symmetric, but not positive definite. For this matrix, we do see that $localSize = 10, 20$ reduced the iteration count, but not significantly, whereas $localSize = 50$ reduces by 50 and all the vectors reduce by only 100. Hence, the iteration gap from storing all the vectors to storing none of them is smaller for this matrix, explaining the less total reduction for the smaller $localSize$ storage case.

4 Conclusion

It is evident that MINRES with local reorthogonalization reduces the number of iterations required to solve $Ax = b$ for symmetric A, whether it be positive or negative definite or even indefinite. A future question to investigate is regarding the balance of the storage costs versus fewer iterations. Clearly, due to the high storage costs, it is not advisable to store all of the Lanczos vectors, even though this produces the fewest number of iterations. This potential speedup also depends on the computational cost of computing the matrix vector product, Av. Thus, finding the optimal number of vectors to store to reduce the iteration count is key. From this work, it is clear that at least 10-20 Lanczos vectors would be necessary. Moreover, storing 10 or 20 of the previous Lanczos vectors will still be efficient on problems of arbitrary size and do reduce the iteration count. Even though this smaller value of $localSize$ does not reduce the number of iterations as much as in the larger value case, due to storage issues, 10 or 20 may be the best compromise. It is clear that loss of orthogonality numerically in the original MINRES does have an effect on the total number of iterations required and significantly increases them. Hence, MINRES with local reorthogonalization reduces the iteration count, making it a viable choice for saddle point problems that, contrary to GMRES, will retain the symmetry.

References

[1] D.C.-L. Fong and M. A. Saunders. LSMR: An iterative algorithm for least-squares problems. *SIAM J. Sci. Comput.*, 33(5):2950–2971, 2011. `http://stanford.edu/group/SOL/software.html`.

[2] T. Davis, Y. Hu and Yahoo! Labs. The University of Florida Sparse Matrix Collection. `http://www.cise.ufl.edu/research/sparse/matrices/list_by_id.html`.

[3] C.C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975. `http://stanford.edu/group/SOL/software.html`.