MODIFYING CONJUGATE GRADIENT METHODS

Jiannan Jiang*
Department of Mathematics
University of California, Berkeley
jjn2015@berkeley.edu

ABSTRACT

In this paper, we have implemented the multiple variants of conjugate gradient method (CG). We have shown that various variants will converge and have better performance than the original CG on some datasets. Furthermore, we have mitigated the convergence issue of CA-CG on some datasets. All of the code used in this paper can be found in `http://github.com/~jonny97/CG_variants`.

***K*eywords** Conjugate Gradient · Parallel Computing · Communication-avoiding

1 Introduction

Conjugate gradient (CG) method is a popular method for solving large sparse linear systems [1]. The method is provably efficient when we have a sparse and well-conditioned matrix A. CG method is useful in many real-world scenarios such as solving partial differential equations and optimization problems. In past few decades, several communication-avoiding variant of CG has been devised and all suffer more or less from the convergence issue[2]. In this project, we have examined the convergence of CG and its communication avoiding variants and aim to make some improvements on top of the method.

2 Background

2.1 Conjugate Gradient Method

In contrast to traditional gradient-based methods, conjugate gradient method aims to reduce the number of iterations needed. The popular gradient-based framework updates the solution x along the direction of gradients. However, calculating the gradient requires at least 2 matrix-vector multiplication and the search directions may repeat itself in later iterations, which results in a high number of iterations to converge. In cases where numerical issues can be ignored, CG is absolutely better than exact line search as it takes at most n iterations (where n is the dimension of the square matrix), and each iteration only involves one matrix-vector multiplication.

The Conjugate gradient method relies on the Krylov subspace that provides an easy way to generate a set of A-conjugate vectors

$$K_i(A, d) = span\left\{d, Ad, A^2d, ..., A^{i-1}d\right\} \tag{1}$$

The conjugate gradient method chooses $x_i \in K_i(A, d)$ to minimize $(x - x^*)^T A(x - x^*)$ [3]. In each iteration i, we will need to update the next solution x_{i+1} and residual r_{i+1}, which consists of only one matrix-vector multiplication, one inner product of the residue vector, and several vector-vector additions [2]. The classic conjugate gradient method is shown in Algorithm 1. Note that in finite precision settings, CG is actually solving a problem with a slightly perturbed eigenvalue. Therefore, matrix with large condition number will be harder to converge.

*This is the final project for EE227C at University of California Berkeley in Spring 2019

Algorithm 1 Conjugate Gradient Method (CG)

Require: Approximate solution x_0 to $Ax = b$

$r_0 = b - Ax_0, d_0 = r_0$
for `i=0,1,...until convergence` **do**
  $\alpha_i = (r_i^T r_i)/(d_i^T A d_i)$
  $x_{i+1} = x_i + \alpha_i d_i$
  $r_{i+1} = r_i - \alpha_i A d_i$
  $\beta_{i+1} = (r_{i+1}^T r_{i+1})/(r_i^T r_i)$
  $d_{i+1} = r_{i+1} + \beta_{i+1} d_i$
end for
return x_{i+1}

2.2 Communication-Avoiding Conjugate Gradient Method (CA-CG)

Since movement of data plays a larger role than computation in limiting performance in modern computers [2], multiple performance tuning has been done for this method, including devising communication avoiding variants and pipelining the methods over the past few decades. Most of the communications of this method occur in Sparse matrix-vector multiplication(SpMV) and in calculating the inner products of vectors.

Since there is not much space for improvement on vector operations, Communication-Avoiding Conjugate Gradient(CA-CG) method will combine multiple iterations of CG together to reduce the communication for calculating inner products of vectors. Thus, CA-CG will be running matrix multiplications instead of multiple vector inner products, which will be faster since current BLAS-3 implementation can be much faster than the equivalent multiple executions of BLAS-2. In CA-CG, iterations are split into an inner loop over $0 \leq j < s$ and an outer loop over i, whose range depends on the number of steps until convergence. We index the iteration i in CG as the iteration $si + j$ in CA-CG [2]. The communication-avoiding conjugate gradient method is shown in Algorithm 2.

Algorithm 2 Communication-Avoiding Conjugate Gradient Method (CA-CG)

Require: Approximate solution x_0 to $Ax = b$

$r_0 = b - Ax_0, d_0 = r_0$
for `i=0,1,...until convergence` **do**
  `Compute` P_i, R_i, `let` $V_i = [P_i, R_i]$; assemble B_i
  $G_i = V_i^T V_i$
  $d_{i,0}^{'T} = [1, 0_{2s}^T], r_{i,0}^{'T} = [0_{s+1}^T, 1, 0_{s-1}^T], x_{i,0}^{'} = 0_{2s+1}$
  for `j=0,1,...,s-1` **do**
    $\alpha_{si+j} = (r_{i,j}^{'T} G_k r_{i,j}^{'})/(d_{i,j}^{'T} G_i B_i d_{i,j}^{'})$
    $x_{i,j+1}^{'} = x_{i,j}^{'} + \alpha_{si+j} d_{i,j}^{'}$
    $r_{i,j+1}^{'} = r_{i,j}^{'} - \alpha_{si+j} B_i d_{i,j}$
    $\beta_{si+j+1} = (r_{i,j+1}^{'T} G_i r_{i,j+1}^{'})/(r_{i,j}^{'T} G_i r_{i,j}^{'})$
    $d_{i,j+1} = r_{i,j+1}^{'} + \beta_{si+j+1} d_{i,j}^{'}$
  end for
  $x_{si+s} = V_i x_{i,s}^{'} + x_{si}, r_{si+s} = V_i r_{i,s}^{'}, d_{si+s} = V_i d_{i,s}^{'}$
end for
return x_{si+s}

2.3 Testing Data

In this project, we choose several large sparse matrices from SuiteSparse Matrix Collection [4]. For each test, we generate a true answer x either uniformly randomly from $[-1, 1]$ or from a standard normal distribution. We then compute the right hand side b via $b = Ax$, and use our implementations to find an answer that has a small residue. (Since some of the problems are actually ill-conditioned, we will only compare the magnitude of residue $r := b - Ax$ for convergence.)

3 Motivation of improvements

CG is purposely designed so that every search direction is A-conjugate to others. However, as shown in the algorithm, the search direction and the residue of CG in each iteration is calculated only based on the data of the previous iteration. Therefore, the errors in each iteration will be accumulated. This issue is well-aware since the invention of the method, and a common trick is to "restart" the algorithm by setting the residue to b-Ax every several iterations. However, the convergence issue of CG is not fully resolved: the assumption that we have picked a search direction that is A-orthogonal to all previous search directions is completely violated after several iterations. This error accumulates across all iterations and making the residue of CG oscillating for each iteration.

Even though in high dimensions, the orthogonality of vectors is impossible to guarantee due to the limitations of floating point number representation, we still want to make a good "guess" for the new search direction given that our previous search directions are not perfect due to truncation errors. The original version of conjugate gradient method trust blindly that the previous calucations are all correct, and although this seems to be the most reasonable choice to make, we suspect better choices can be made in cases where conjugate gradient method performs poorly.

In our project, we have modified the line 7 of the original algorithm with only additional vector-vector operations and smoothes the residue error of CG over all datasets we have tested.

4 Modifications

As discussed above, the choice of the search direction may no longer be optimal due to finite arithmetic. Instead, we will replace the line 7 of the original algorithm with a linear combination of r and d. As follows,

Algorithm 3 Smoothing variant of Conjugate Gradient Method (CG)

Require: Approximate solution x_0 to $Ax = b$
1: $r_0 = b - Ax_0, d_0 = r_0$
2: **for** `i=0,1,...until convergence` **do**
3: $\alpha_i = (r_i^T r_i)/(d_i^T A d_i)$
4: $x_{i+1} = x_i + \alpha_i d_i$
5: $r_{i+1} = r_i - \alpha_i A d_i$
6: $d_{i+1} = \texttt{span}(r_{i+1}, d_i)$
7: **end for**
8: **return** x_{i+1}

Note that the optimal linear combination of r_{i+1} and d_i is left for later discussion.

4.1 Intuition

After this modification, the method does not seem like CG anymore but more of a variant of steepest descent. Specifically, we just picked a direction d, and performs a update of the solution x in the direction d. Therefore, we can expect convergence via the tools we have for exact line search. On the other hand, if the linear combination we perform here is close enough to the original conjugate gradient method, we should expect the number of iterations roughly the same as CG, or even better.

4.2 Convergence

Suppose we are solving $Ax = b$ where A is real positive semi-definite and therefore symmetric. We can think of it as solving an optimization problem with a quadratic objective. Consider $f(x) = \frac{1}{2}x^T Ax - bx$. Then clearly, $\arg\min f(x)$ satisfies $Ax = b$.

4.2.1 Optimal choice of α_i

Suppose we have already picked an arbitrary direction d_i and we will perform an update

$$x_{i+1} = x_i + \alpha_i d_i \tag{2}$$

The optimal α_i to minimize $f(x)$ should satisfy $\frac{df(x_{i+1})}{d\alpha_i} = 0$. Therefore,

$$0 = \frac{df(x_{i+1})}{d\alpha_i} = \frac{df(x_{i+1})}{dx_{i+1}}^T \frac{x_{i+1}}{d\alpha_i} = (Ax_{i+1} - b)^T d_i = (A(x_i + \alpha_i d_i) - b)^T d_i \tag{3}$$

After simplification, we can get

$$\alpha_i = \frac{d_i^T r_i}{d_i^T A d_i} \tag{4}$$

Based on the update rule $d_i = c_1 r_i + c_2 d_{i-1}$ (where c_1, c_2 are arbitrary values), we have

$$\alpha_i = \frac{c_1 r_i^T r_i}{d_i^T A d_i} + \frac{\beta_i d_{i-1}^T r_i}{d_i^T A d_i} \tag{5}$$

Note that equation (3) can also be interpreted as $r_{i+1}^T d_i = 0$. Thus, by induction, we have $r_i^T d_{i-1} = 0$ in the previous iteration. Therefore,

$$\alpha_i = \frac{c_1 r_i^T r_i}{d_i^T A d_i} \tag{6}$$

is the optimal value to minimize $f(x_{i+1})$ regardless of the value of β_i.

4.2.2 Convergence of Conjugate Gradient

Note that we have always picked α_i to be the value to minimize $f(x_{i+1})$. We have:

$$f(x_{i+1}) < f(x_i) \forall i \tag{7}$$

Therefore, the function value f is decreasing and lower bounded by 0 and the algorithm mush converge. Since the Krylov subspace forms a basis, the function value f is guaranteed to decrease every n iterations if f is not a local minima. Thus, the method will always converge, and is a descent method for minimizing function value of f.

4.2.3 Error Bounds

We have not developed a fully theory of error bounds yet. However, as indicated from the above section, we need at most n times more iterations than CG to converge. As shown in the experiment below, this variant converges usually within twice as many iterations as CG and this factor does not seem to grow with respect to the problem size. This is expected, as in infinite arithmetic, CG has always picked the optimal direction. However, the experimental results for the variant has much lower residue than CG when both methods are on their way of convergence.

5 Experimental results

5.1 Choice of update rule

CG has picked β_i so make the directions to be A-conjugate to all the previous directions. Since the optimal direction after a few iterations of imperfect iterations will is expensive to find, any small modifications of β will not help the convergence both initially. Experimentally we have found the following update rule to be most effective:

$$\beta_i = (r_i^T r_i)/(r_{i-1}^T r_{i-1}) \tag{8}$$

$$d_i = (1 + \beta_i) r_i + \beta_i^2 d_{i-1} \tag{9}$$

Note that by the update rule for β, we expect it to be close to 1 for all iterations. so we can think of the update rule to be roughly:

$$d_i = (r_i) + (r_i + \beta_i^2 d_{i-1}) \tag{10}$$

Note that based on the above analysis, the optimal α_i will be $\frac{r_i^T r_i}{d_i^T A d_i}$ for both direction r_i and direction $(r_i + \beta_i^2 d_{i-1})$. Therefore, the effective update rule becomes:

$$x_{i+1} = x_i + \alpha_i d_i = x_i + \alpha_i \hat{d}_i - \alpha_i r_{i+1} \tag{11}$$

where d_i denotes the actual directions we use, $\hat{d}_i$ denotes the direction $(r_i + \beta_i^2 d_{i-1})$. Therefore, the update can be viewed as combining CG and gradient descent within one step at virtually no extra cost. With the choice of α being optimal for both directions. (However, it could be argued that individually optimal is no longer the optimal for the combination).

Remark: We have not yet found a convincing argument for why this update rule is the best yet.

5.2 Experimental convergence

As preluded in the above, our update rule is better for initial convergence, but has a relatively low overall convergence rate than CG. Note that both methods converge to machine error given long enough time. In the first matrix, *cant*, a

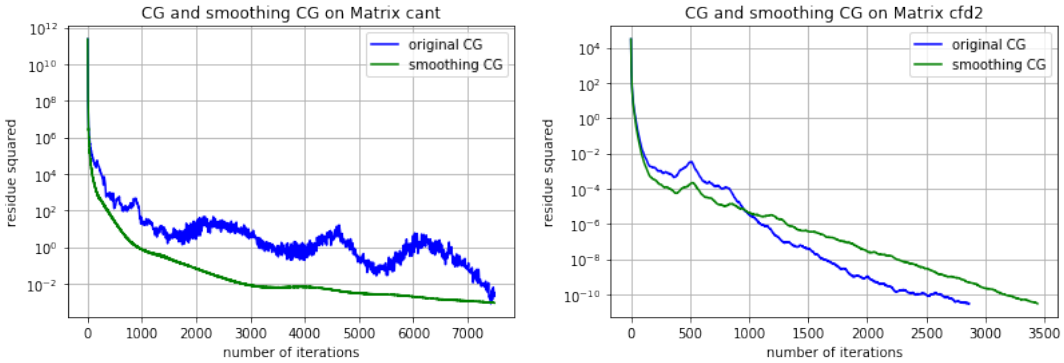

Figure 1: Convergence behavior of CG and the smoothing-variant of CG on dataset *cant* and *cfd2*

62451 by 62451 PSD matrix with 4 million nonzeros, the convergence of smoothing CG is much better than original CG, whereas in the second matrix, *cfd2*, a 914231 by 914231 PSD matrix with 4.45 million nonzeros, the performance of CG is better. This indicates that our variants are better than original CG on harder problems (the condition number of *cant* is much higher than that of *cfd2*). And in the easier problems, our variants still converge with slightly more iterations.

It might be argued that the this smoothing behavior is simply the result of using a gradient decent, and the following is a comparison between CG, our variants, gradient descent with fixed α, and gradient decent with α chosen according to equation (6):

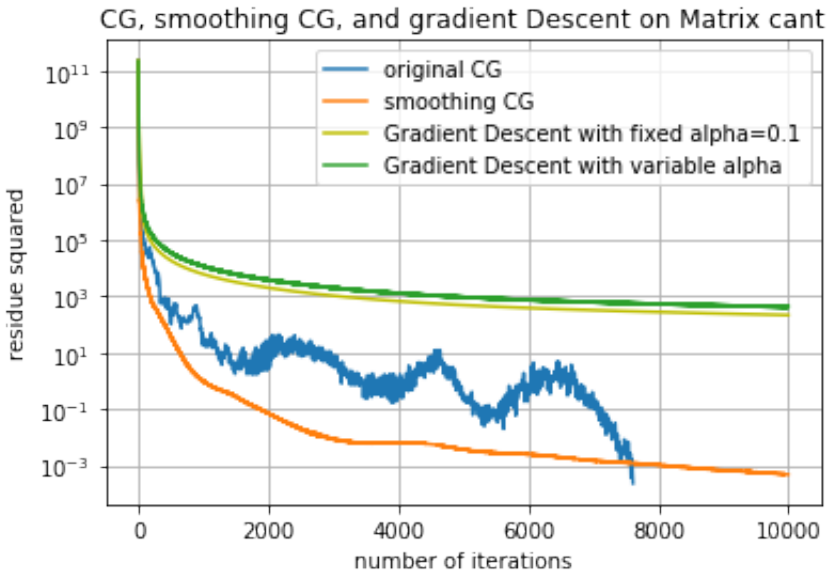

Figure 2: Convergence behavior of 4 different methods are matrix *cant*

It is clear that the convergence of both CG and our variants are much better than gradient method. Note in the above graph, the errors are tested based on the l2 norm of the error, while the gradient base method are calculating gradient of function f here.

5.3 Improvements on CA-CG

The convergence of communication-avoiding conjugate gradient method is a serious issue. For example, in [5], CA-CG on matrix cant does not converge as well as CG:

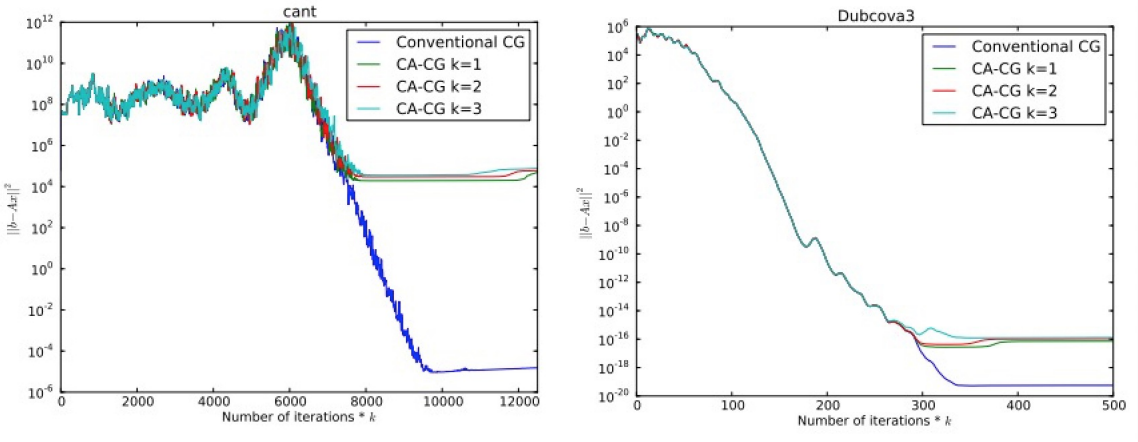

Figure 3: Convergence behavior of CA-CG on dataset *cant* and *dubcova3*

With our implementation, we can get the folllowing convergence for CA-CG:

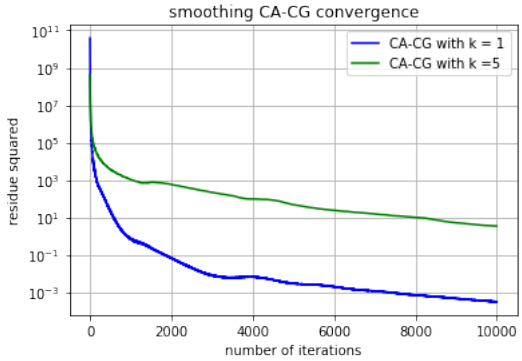

Figure 4: Convergence behavior of smoothing CA-CG on dataset *cant*

Note that on *cant*, the original convergence stops at 10^4 and with our new implementation, the error becomes 10^{-3}. Note that the convergence issue still exists if we pack more inner iterations.

6 Conclusion and future work

As shown on the results, our suggested update rule makes the initial decent of CG more smooth, and the residue seems to be decreasing at the same asymptotic rate as the original conjugate gradient method. This update seems to be robust and can be used in various variant of CG.

Currently we have no stability proofs for this variant. Although based on the testing, this variant alone will need 50% more iterations to converge, the variant seems give better convergence consistently on hard problems where convergence to machine errors are not required.

7 Rubric checks:

Discussion of related work: in section 2.
What we've done: devising the new variant in section 3 and 4, and experiments in section 5.
How the topic is related to the course: as discussed in section 4, we can view solving $Ax = b$ as a minimization problem of minimizing $f(x) = \frac{1}{2}x^T Ax - bx$. This project also relates to the iterative methods we discussed in the lecture. Generally, the point of these variants are figuring out best ways to solve optimization questions.

References

[1] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[2] Erin Carson, Nicholas Knight, and James Demmel. An efficient deflation technique for the communication-avoiding conjugate gradient method. *Electronic Transactions on Numerical Analysis*, 43(125141):09, 2014.

[3] Gene H Golub and Dianne P OâĂŹLeary. Some history of the conjugate gradient and lanczos algorithms: 1948–1976. *SIAM review*, 31(1):50–102, 1989.

[4] Tim Davis, Yifan Hu, and Scott Kolodziej. The suitesparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, 2011.

[5] Jeffrey Morlan, Shoaib Kamil, and Armando Fox. Auto-tuning the matrix powers kernel with sejits. In *International Conference on High Performance Computing for Computational Science*, pages 391–403. Springer, 2012.