Faster Solver for Multiple Linear Systems via Block Conjugate Gradients

Lu, William *
Mentor: Wechsung, Florian

*Applied Math Summer Research Experience (AM-SURE) 2021.
Courant Institute of Mathematical Sciences, New York University, wl1869@nyu.edu

Abstract

Iterative methods such as Conjugate Gradient and GMRES have long been the standard for solving the linear systems that arise from finite element discretisations of Partial Differential Equations. However, these methods usually solve one linear system at a time. Using Block Conjugate Gradient allows us to solve multiple linear system at once faster. We find out that BCG has a cheaper memory communication cost than that of CG. BCG reaches the solution in much fewer iterations than CG due to information sharing. This idea holds true for Preconditioned CG and Preconditioned BCG as well.

1 Contribution

This report is based on Hestenes and Stiefel's conjugate gradient [3] and O'Leary's block conjugate gradient method [4]. We further explain the mathematical details of the algorithms by using Cockett's [1] and Shewchuk's paper [5] and the textbook written by Elman, Silvester, and Wathen [2].

This report will give a general explanation of four analogous algorithms– conjugate gradient, block conjugate gradient, preconditioned conjugate gradient, and preconditioned block conjugate gradient. We compare equations that describe the time and memory costs for preconditioned conjugate gradient and preconditioned block conjugate gradient. We plot graphs for convergence and time spent of these algorithms. We also find out another method to avoid the singular matrix issue that we encounter in the block version algorithms.

2 Introduction

Given a linear system $Ax = b$, LU decomposition will be sufficient for us to find the solution provided that the matrix size is small. The time complexity for LU decomposition is approximately $O(n^3)$, which is not ideal if the matrix is too large.

Therefore, an iterative method that uses an initial guess to generate a sequence of improving approximate solutions is useful in this case, because each step only involves a matrix-vector product. The calculation is cheaper at each iterate than LU decomposition.

Steepest Descent is one of the iterative methods that finds the approximate true solution of a linear system. When we try to solve the linear system $Ax = b$, it is equivalent to find the minimum of the quadratic form $f(x) = \frac{1}{2}x^{\mathsf{T}}Ax - b^{\mathsf{T}}x$. $f'(x) = Ax - b$, according to Shewchuk [5, eqn. (6)&(7)]. Starting from the initial guess x_0, we find the sequence of the improving approximate solutions, $x_1, x_2, ...x_k$, by always finding the negative gradient of each iterate, $\nabla f = r_k = b - Ax_k$, because the direction of the negative gradient is the direction in which the function decreases most quickly from each x_k. Define the k^{th} error iterate as $e_k = x_k - x$. If we plug the error equation into the residual equation, we see that $r_k = -Ae_k$. So the residual direction is the steepest descent direction. However, this method is still not ideal. The same search direction may be taken several times instead of once.

Conjugate Gradient improves Steepest Descent by avoiding unnecessary "stair-like" paths. CG is an algorithm that solves a linear system with the matrix being symmetric and positive-definite. We advance steps in a set of orthogonal directions. In this way, we do not have to take steps in the same direction multiple times. The search direction p_k is determined from each iterate's residual vector and has to satisfy some properties. This new residual vector is orthogonal to the previous residuals and search directions. Hence, since we take each orthogonal direction once, CG is much faster than the method of steepest descent.

Now consider that there are multiple linear systems for us to solve. Given multiple linear systems $AX_\ell = B_\ell$, where $\ell \in \mathbb{N}$ is the block size of the right hand side, $X \in R^{n\times l}, B \in R^{n\times l}$. Using CG many times in this case works; however, when we have multiple linear systems, it sounds more ideal if we let our computer receive all problems at once rather than multiple times. We hope that providing all information at once will allow the CPU to solve them in fewer iterations. This will help saving the memory communication cost, the cost for the CPU to access the received information from the memory, as well. We use BCG for faster convergence.

In the case of incompressible fluid flow simulation, the matrix A could correspond to Stoke's equation. A could also be other PDEs, such as Elasticity equations. The right hand side B could represent different boundary conditions.

We will observe how the matrix-vector multiplication plays a role in terms of complexity and see why we prefer Block Conjugate Gradient for a large linear system and multiple RHS.

3 Algorithms

3.1 CG and BCG

We will first observe the differences between CG and BCG algorithms.

Algorithm 1 CG

Input: Matrix A, a guessed solution x_0, a RHS b, and a threshold.
$r_0 = b - Ax_0$
if r_0 is smaller than the threshold, return x_0.
$p_0 = r_0$
while true **do**
    $\alpha_k = \frac{r_k^\mathsf{T} r_k}{p_k^\mathsf{T} A p_k}$
    $x_{k+1} = x_k + \alpha p_k$
    $r_{k+1} = r_k - \alpha A p_k$
    if r_{k+1} is smaller than the threshold **then**
        exit the loop
    else
        $\beta_k = \frac{r_{k+1}^\mathsf{T} r_{k+1}}{r_k^\mathsf{T} r_k}$
        $p_{k+1} = r_{k+1} + \beta_k p_k$
End Repeat
return x_{k+1}

Algorithm 2 Block CG

Input: Matrix A, a guessed solution X_0, a RHS B, and a threshold.
$R_0 = B - AX_0$
if R_0 is smaller than the threshold, return X_0.
$P_0 = R_0$
while true **do**
    $\Lambda_k = (P_k^\mathsf{T} A P_k)^{-1} R_k^\mathsf{T} R_k$
    $X_{k+1} = X_k + P_k \Lambda_k$
    $R_{k+1} = R_k - A P_k \Lambda_k$
    if R_{k+1} is smaller than the threshold **then**
        exit the loop
    else
        $\Phi_k = (R_k^\mathsf{T} R_k)^{-1} R_{k+1}^\mathsf{T} R_{k+1}$
        $P_{k+1} = R_{k+1} + P_k \Phi_k$
End Repeat
return X_{k+1}

BCG deals with multiple RHS at the same time while CG deals with one column

vector per time. Besides, notice that on line 6 and 8 in CG, the matrix-vector product dominates the cost of the whole algorithm, whereas in BCG the matrix-matrix multiplication contributes the most to the cost of the algorithm. r_k and R_k both represent the residuals at kth step. p_k and P_k are the kth search directions. α_k and Λ_k are the step sizes for the search direction. And finally we use β_k and Φ_k to find the new search directions.

The search directions $p_0, p_1, ..., p_k$ are A-orthogonal to each other. That is, $p_i^T A p_j = 0$ for $i \neq j$. Following [2, Lemma 2.1, p. 74], we can write the solution as a linear combination of this set of search directions which could be written as a Krylov subspace $\mathcal{K}_k = span\{p_0, p_1, ..., p_k\} = span\{p_0, Ap_0, ..., A^{k-1}p_0\} = span\{r_0, Ar_0, ..., A^{k-1}r_0\}$. For BCG, the new iterate X_k will be contained in an expanding block-Krylov subspace $X_0 + span(R_0, AR_0, ...A^{k-1}R_0)$. Information sharing of Block CG is achieved by the Block CG subspace at below.

Definition 3.1. The block Krylov subspace is defined such that the i^{th} column of the iterate X_K can be expressed by the linear combination of as much as n number of Krylov subspace that created thus far. In other words, $X_K^{(i)}$ can be expressed through $X_0^{(i)} + \bigoplus_{i=1}^{n} span(R_0^{(i)}, AR_0^{(i)}, ...A^{k-1}R_0^{(i)})$, following [1, eqn. (24), p. 6].

For line 6 and 12 in BCG algorithm, instead of finding the inverse of $R_k^\intercal R_k$ and $P_k^\intercal A P_k$, we find the pseudo inverse of this product. This is due to the fact that the block B may contain a column of zeros or have two equivalent columns. In other words, the rank of P_k is less than the size of the block. This makes the product $P_k^\intercal A P_k$ become singular. The function from numpy, *numpy.linalg.pinv*, will compute the generalized inverse of a matrix using singular value decomposition. There are other methods to prevent this issue, such as the method from here [4, p. 301]. O' Leary proposed that if the columns of the search direction P_k lose their independence, we should delete the zeros or redundant columns of P_k, X_k, and R_k. In this way, the algorithm will converge, and the deleted columns will be updated separately.

3.2 Preconditioned CG and Preconditioned BCG

Conjugate gradient is a fast iterative method; however, it can be faster by using a preconditioner.For example, we find another matrix M such that $M^{-1} \approx A^{-1}$. Multiplying the inverse of the preconditioner will transform A into an identity matrix and change the linear system from $Ax = b$ to $M^{-1}Ax = M^{-1}b$.

Algorithm 3 PCG

1: **Input**: Matrix A, a preconditioner M, a guessed solution x_0, a RHS b, and a threshold.
2: $r_0 = b - Ax_0$
3: $z_0 = M^{-1}r_0$
4: if r_0 is smaller than the threshold, return x_0.
5: $p_0 = r_0$
6: **while** true **do**
7: $\quad \alpha_k = \frac{r_k^\intercal z_k}{p_k^\intercal A p_k}$
8: $\quad x_{k+1} = x_k + \alpha p_k$
9: $\quad r_{k+1} = r_k - \alpha A p_k$
10: $\quad$ **if** r_{k+1} is smaller than the threshold **then**
11: $\quad\quad$ exit the loop
12: $\quad$ **else**
13: $\quad\quad z_{k+1} = M^{-1}r_{k+1}$
14: $\quad\quad \beta_k = \frac{r_{k+1}^\intercal z_{k+1}}{r_k^\intercal z_k}$
15: $\quad\quad p_{k+1} = z_{k+1} + \beta_k p_k$
16: End Repeat
17: return x_{k+1}

Algorithm 4 PBCG

1: **Input**: Matrix A, a preconditioner M, a guessed solution X_0, a RHS B, and a threshold.
2: $R_0 = B - AX_0$
3: $Z_0 = M^{-1}R_0$
4: if R_0 is smaller than the threshold, return X_0.
5: $P_0 = R_0$
6: **while** true **do**
7: $\quad \Lambda_k = (P_k^\intercal A P_k)^{-1} R_k^\intercal Z_k$
8: $\quad X_{k+1} = X_k + P_k \Lambda_k$
9: $\quad R_{k+1} = R_k - A P_k \Lambda_k$
10: $\quad$ **if** R_{k+1} is smaller than the threshold **then**
11: $\quad\quad$ exit the loop
12: $\quad$ **else**
13: $\quad\quad Z_{k+1} = M^{-1}R_{k+1}$
14: $\quad\quad \Phi_k = (R_k^\intercal Z_k)^{-1} R_{k+1}^\intercal Z_{k+1}$
15: $\quad\quad P_{k+1} = Z_{k+1} + \Phi_k P_k$
16: End Repeat
17: return x_{k+1}

We do this because preconditioning will make the calculation much easier. Now these two preconditioned algorithms work almost the same as CG and BCG algorithms. The only difference is on line 3 and 13, where we multiply the preconditioner with the residual. The preconditioners that we use are found by using Jacobi method.

3.3 Convergence

Definition 3.2. We define the energy-norm or A-norm of every error iterate as $||e||_A = (e^T A e)^{\frac{1}{2}}$ [2, eqn. (1.112)].

The convergence theorem for Conjugate Gradient following [5, eqn. (52)] is:

$$||e_k||_A \leqslant 2(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^k ||e_0||_A \tag{1}$$

where $\kappa = \frac{\lambda_n}{\lambda_1}$ is the condition number of the matrix A. λ_n and λ_1 represent the largest and the smallest eigenvalue of the matrix A respectively. But for block conjugate gradient, we use $\kappa_\ell = \frac{\lambda_n}{\lambda_\ell}$ for the convergence theorem. λ_ℓ is the ℓ^{th} largest eigenvalue of the matrix A. We apply the Frobenius norm for error iterates. In the spectrum of

matrix A, λ_n is the largest eigenvalue, and λ_ℓ is the ℓ^{th} largest eigenvalue. According to O' Leary [4, p. 312], the convergence theorem becomes

$$||E_k|| \leqslant 2 \cdot (\frac{1-\sqrt{\kappa_\ell^{-1}}}{1+\sqrt{\kappa_\ell^{-1}}})^k ||E_0|| = 2(\frac{\sqrt{\kappa}_\ell - 1}{\sqrt{\kappa}_\ell + 1})^k ||E_0|| \tag{2}$$

Suppose the block size is one and the initial guesses are the same for both CG and BCG. Then, the error norm for the initial guesses should be the same as well. κ_ℓ will be equal to κ, as ℓ is one.

Therefore, when the block size is one, they have the same convergence rate. When the block sizes increase, κ_ℓ gets closer to 1. BCG converges faster if the spread of eigenvalues of the matrix is more clustered [4, p. 295]. Since κ_ℓ is closer to 1 than κ, BCG requires fewer iterations. This works the same for PBCG as well.

4 Cost

Define the time cost notation $T_{\{algorithm,blocksize\}}$ as the time spent for an algorithm to solve for a certain block size. Define $Iter_{\{algorithm,blocksize\}}$ as the number of iterations required for an algorithm to solve a certain block size. Given ℓ linear systems with the same matrix A but different column vectors b, the time cost is:

$$T_{\{PCG,\ell\}} = \ell \cdot T_{\{PCG,1\}} \approx \ell \cdot Iter_{\{PCG,1\}} \cdot T_{\{MatVec,1\}}$$

where the time cost of the matrix-vector product $T_{\{MatVec,1\}} \approx T_{\{A,1\}} + T_{\{P,1\}}$.

$T_{\{A,1\}}$ is time for calculating the matrix vector product of A and one column vector. $T_{\{P,1\}}$ is time for calculating the matrix vector product of the preconditioner and one column vector.

The time cost for solving l RHS with PBCG is:

$$T_{\{PBCG,\ell\}} = Iter_{\{PBCG,\ell\}} \cdot T_{\{MatMat,\ell\}}$$

where the time cost of the matrix-matrix product $T_{\{MatMat,\ell\}} \approx T_{\{A,\ell\}} + T_{\{P,\ell\}}$.

$T_{\{A,\ell\}}$ is time for calculating the matrix-matrix product of A and l column vectors. $T_{\{P,\ell\}}$ is time for calculating the matrix-matrix product of the preconditioner P and l column vectors. From the above two equations, we observe that there are two ways that BCG is faster than CG, $T_{\{PBCG,\ell\}} < T_{\{PCG,\ell\}}$, for solving a linear system with ℓ RHS. First, $Iter_{\{PBCG,\ell\}} < Iter_{\{PCG,1\}}$. Second, $T_{\{MatMat,\ell\}} < \ell \cdot T_{\{MatVec\}}$.

In the first case, the number of iterations for BCG and CG depends on the spread of eigenvalues of matrix A. Define the eigenvalues of the matrix A as $\lambda_1, \lambda_2, ...\lambda_n$. Then, its condition number is $\kappa = \frac{\lambda_n}{\lambda_1}$. When using PBCG, we use the ratio $\kappa_\ell = \frac{\lambda_n}{\lambda_\ell}$. By equation (1) and (2), we see that the block convergence is faster because κ_ℓ is closer to 1. Thus, $Iter_{\{PBCG,\ell\}} < Iter_{\{PCG,1\}}$.

In the second case, suppose A is a sparse matrix of size n that has $s_A << n$ non-zeros per row. Assume the preconditioner is a sparse matrix of size n that has $s_P << n$ non-zeros per row. For block size $\ell = 1$, the computational cost of matrix A with a vector is $T^{flops}_{\{A,1\}} = O(s_A \cdot n)$. The memory communication cost of matrix A is $O(s_A \cdot n)$, and the memory communication cost of a vector b is $O(n)$. In total, $T^{mem}_{\{A,1\}} = O(s_A \cdot n + n)$. We apply the same rule for calculating the memory communication cost and the computational cost for the preconditioner.

For block size $\ell > 1$, if we do CG multiple times, then the costs become:

$$\ell \cdot T^{flops}_{\{A,1\}} = \ell \cdot O(s_A \cdot n) = O(\ell \cdot s_A \cdot n) \tag{3}$$

$$\ell \cdot T^{mem}_{\{A,1\}} = \ell \cdot O(s_A \cdot n + n) = O(\ell \cdot s_A \cdot n + n \cdot \ell) \tag{4}$$

$$\ell \cdot T^{flops}_{\{P,1\}} = \ell \cdot O(s_P \cdot n) = O(\ell \cdot s_P \cdot n) \tag{5}$$

$$\ell \cdot T^{mem}_{\{P,1\}} = \ell \cdot O(s_P \cdot n + n) = O(\ell \cdot s_P \cdot n + n \cdot \ell) \tag{6}$$

However, if we do PBCG for block RHS once, the computational cost and the memory communication cost are:

$$T^{flops}_{\{A,\ell\}} = O(s_A \cdot n \cdot \ell) \tag{7}$$

$$T^{mem}_{\{A,\ell\}} = O(s_A \cdot n + n \cdot \ell) \tag{8}$$

$$T^{flops}_{\{P,\ell\}} = O(s_P \cdot n \cdot \ell) \tag{9}$$

$$T^{mem}_{\{P,\ell\}} = O(s_P \cdot n + n \cdot \ell) \tag{10}$$

We see from (3) and (7) that the computational cost for using PBCG once and using PCG l times is the same. This is true for (5) and (9); however, comparing (4) and (8), we observe that PBCG has a smaller memory communication cost. The lowwer memory communication cost motivates us to apply PBCG. When the linear system is large, $T^{mem}_{\{P,\ell\}}$ will dominate the cost due to the fact that $s_P > s_A$. Still, comparing equation (6) and (10), we see that the memory communication cost, $T^{mem}_{\{P,\ell\}}$, will be smaller than $\ell \cdot T^{mem}_{\{P,1\}}$.

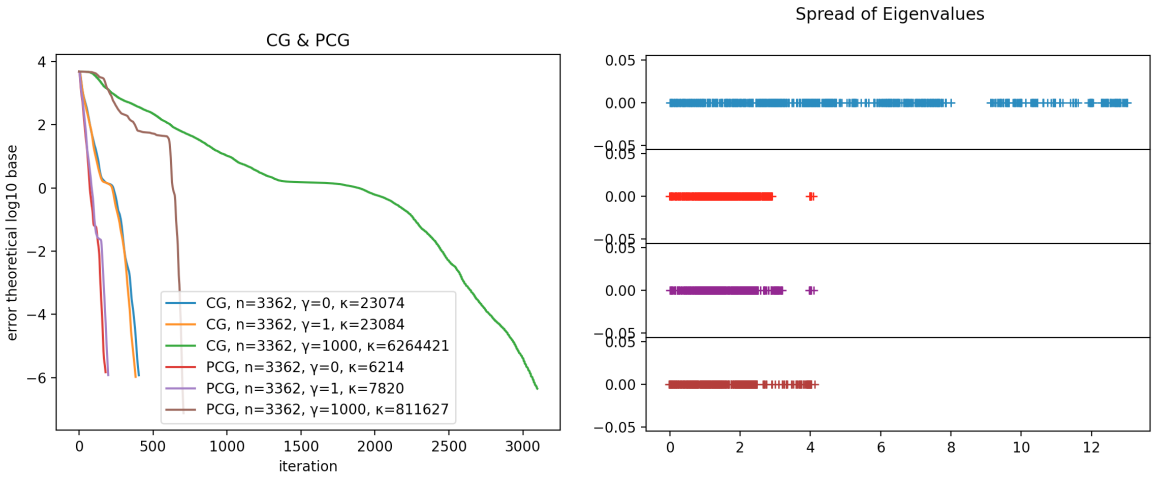
5 Results

In this section, we will first investigate what effect a preconditioner has on the algorithm. We will see how the spectrum of the matrix will influence the convergence rate. Then, we see the relationship between increasing block size and convergence. Finally, we plot the time spent for PBCG and PCG to solve ℓ linear systems and the time spent for each iteration of the two algorithms.

We consider matrices $A_{n,\gamma} \in \mathbb{R}^{n*n}$, where $n \in \{882, 3362, 13122\}$ and $\gamma \in \{0, 1, 1000\}$. Here γ is a parameter, and large γ corresponds to a more complex system. n is the size of the matrix.

5.1 CG vs PCG

Figure 1(a) gives us six convergence lines for CG and PCG. The size of these six matrices are 3362 by 3362, but the value of γ of these matrices are distinct. For CG, we find the condition number of the matrix A. For PCG, we find the condition number of the product of the matrix A and the preconditioner P. Figure 1(b) is the eigenvalue graph for four of the matrices. The spread of the eigenvalues is shown by a horizontal axis.



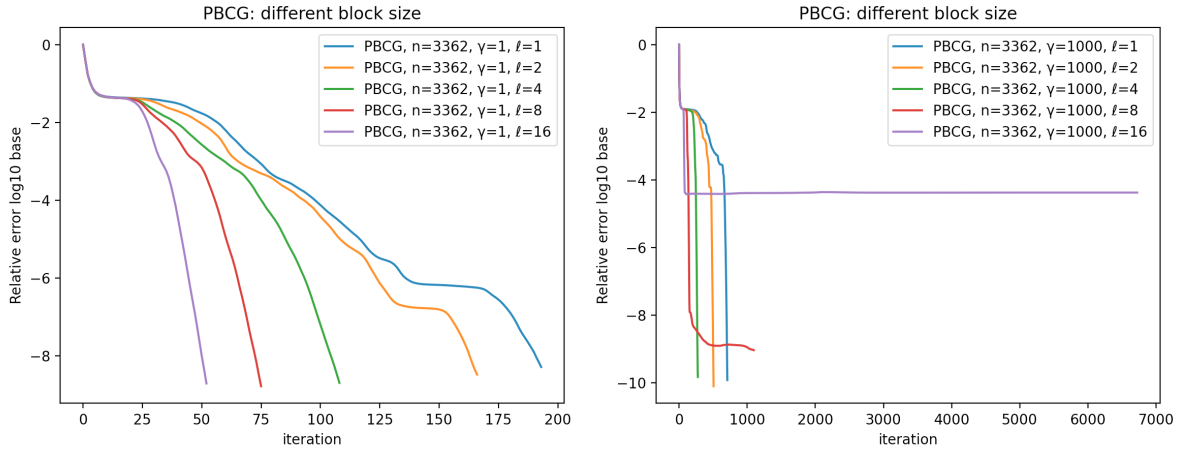
(a) CG and PCG on three matrices

(b) Spectrum of matrices

Figure 1: *In (a), we conclude that PCG is faster than CG. The condition number serves as an evidence to support the statement. In (b), the spectrum of a matrix corresponds to the lines in (a) by color. The more clustered the eigenvalues, the faster the convergence rate. The red line converges in around 200 iterations, and its eigenvalues are more clustered than the blue, purple, and brown one.*

5.2 Gamma and the block size effects

In figure 2, we observe how the block size influences the number of iterations and how γ might make a difference for the convergence. In figure 2(a), it only requires one fourth of the number of iterations for $\ell = 16$ than for $\ell = 1$. In figure 2(b), we concludes that the number of iterations decreases if the block size increase. Nevertheless, when block size is 8 and 16, the number of iterations required exceeds the number of iterations required for smaller block sizes. We suspect that this peculiarity might be due to the use of pseudo inverse in algorithm 4.



(a) (b)

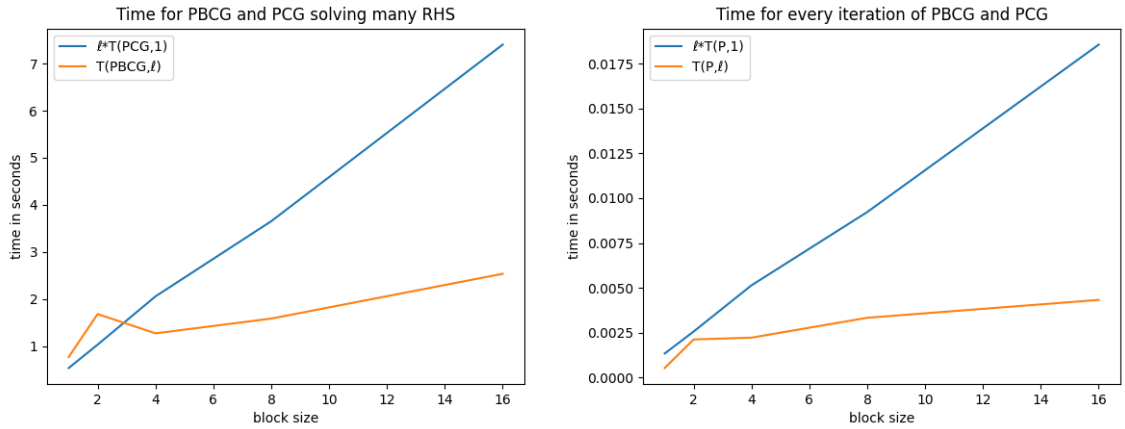
Figure 2: *These two graphs show how $A_{3362,1}$ and $A_{3362,1000}$ behave under PBCG. When the value of gamma is large and the block size is big, the corresponding iterations will be fewer. The application of pseudo inverse still works when γ is small and block size is < 16.*

5.3 Use PBCG once or PCG several times?

Finally, we answer the ultimate question by plotting the time graph for PBCG and PCG. In figure 3(a), as block size increases, the time required for PBCG to solve ℓ linear systems once is less than the time required for PCG to solve ℓ linear systems. We assume that the gap in the left corner is due to the use of pseudo inverse.

From figure 3(b), we have found out that as the block size increases, the time required for each iteration of PBCG is much less than the time required for PCG. The time required for each iteration of PCG behaves linearly with a smaller slope, but the time required for each iteration of PBCG increases much slower. This is a direct effect of the information sharing property for search directions.

What could happen if we increase the block size ℓ more and more? The size of the product that we will perform pseudo inverse is $\ell \times \ell$. Suppose that $T_{\{pinv\}}$ is the time required for finding the pseudo inverse of a matrix. When ℓ is small, $T_{\{pinv\}}$ is not too expensive. But as ℓ increases, $T_{\{pinv\}}$ will be expensive. Hence, we should not allow the block size to be too large and let $T_{\{pinv\}}$ potentially dominate the cost .



(a) (b)

Figure 3: *(a) illustrates the time difference between PBCG to solve a RHS with size ℓ and PCG to solve ℓ linear systems. $T_{\{PBCG,\ell\}}$ increases slower than $\ell \cdot T_{\{PCG,1\}}$. (b) compares the time spent for each iteration of PBCG and PCG. Notice that for each iteration of PBCG, the preconditioner-matrix product will dominate the cost. Its cost is represented by $T(P, \ell)$.*

6 Conclusions

To summarize, when we want to test multiple boundary conditions in a linear system, rather than applying CG or PCG multiple times, it's faster and requires fewer iterations for us if we use BCG or PBCG. Applying BCG and PBCG, we solve multiple linear systems at once. We need fewer iterations, and each iteration is cheaper for BCG and PBCG when the linear system is large. This result can be seen from the time Figure 3(a) and (b). There are several ways to make the problem easier. For example, if we increase the block size of the RHS, the number of iterations will decrease. If we can design a proper preconditioner, the linear systems will be easier for us to solve. Also, using pseudo inverse can deal with singular matrices that may arise in BCG and PBCG.

Thanks to the cheaper memory communication cost and the information sharing property of BCG and PBCG, we now can solve multiple linear systems with a faster iterative method.

Acknowledgement

Many thanks to my mentor Florian Wechsung for the insightful lessons and great guidance on this paper and the presentation.

References

[1] Rowan Cockett. *The block conjugate gradient for multiple right hand sides in a direct current resistivity inversion*. 2015.

[2] Howard C Elman, David J Silvester, and Andrew J Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Oxford Science Publications, 2014.

[3] Magnus R Hestenes, Eduard Stiefel, et al. *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS Washington, DC, 1952.

[4] Dianne P O'Leary. "The block conjugate gradient algorithm and related methods". In: *Linear algebra and its applications* 29 (1980), pp. 293–322.

[5] Jonathan R Shewchuk. *An introduction to the conjugate gradient method without the agonizing pain*. 1994.