

# MATH 6644

## Project 1

Stephan Boettcher

March 18, 2015

### Part 1

*Discretize the following differential equation:*

$$\begin{cases} -u'' = 2x - \frac{1}{2} & x \in [0, 1] \\ u(0) = 1, & u(1) = -1 \end{cases}$$

*by centered difference scheme with  $n$  interior mesh points. Solve the resulting linear system by Conjugate Gradient (CG), Preconditioned CG with Sine transform to construct the preconditioner.*

The discretized one-dimensional equation above was solved by the central difference scheme with  $n$  interior points. The resulting problem is then an  $(n+2) \times (n+2)$  matrix,  $A$ , and  $(n+2) \times 1$  vectors,  $x$  and  $b$ . The problem was solved using both the Conjugate Gradient (CG) method and the Preconditioned CG method with a Sine transform preconditioner,  $S$ . The  $S$  matrix was defined as:

$$[S]_{j,k} = \sqrt{\frac{2}{n+1}} \sin\left(\frac{\pi jk}{n+1}\right)$$

The  $A$  matrix formed from the discretized equation forms a tridiagonal matrix, with elements directly on the primary diagonal as well as elements one diagonal above and below the primary diagonal. This sparse matrix was then run with the Conjugate Gradient (CG) method. The CG method is one of the most prominent iterative methods for solving sparse matrices as it converges much more quickly than other iterative methods such as steepest descent. The Conjugate Gradient iterates in such a way that each new residual is orthogonal to all the previous residuals and search directions. Each new search direction is constructed is  $A$ -orthogonal to all the previous residuals and search directions, thus using the Krylov subspace to iterate more efficiently. The CG method was run with  $n$  interior points from 256 to 16384. As the values of  $n$  increased, the time necessary to complete the iteration increased proportionally. The final runtimes of the CG method can be found in Figure 2. The CG method was able to converge in roughly  $\frac{n}{2}$  iterations for the given problem set. While the iterations increase by a factor of  $\frac{n}{2}$ , the runtime increased at a rate closer to  $n^3$ . Thus, as the size of the matrix increases, the runtime grows larger at a much faster rate than the number of iterations required to converge. After the CG method completed, the  $S$  matrix, used in the PCG method, was formed.

The  $S$  matrix was formed in two different ways to determine the most efficient method. The first method was using Matlab's vector math and `parfor` to form the matrix in parallel. The parallel code was used was:

```

1 k=1:runs(a); %generates a vector from 1 to the length of n.
2 %runs contains the list of n values and a is the current iteration.
3 parfor j=1:runs(a) %Parallel for loop in matlab
4     S(j,k)=sqrt(2/(n+1))*sin((pi*j*k)/(n+1)); %form an entire column of values
5 end

```

This code was run with 4 workers in parallel, allowing for a theoretical 4x improvement over the serial code. Generating the  $S$  matrix has a great deal of potential parallelism, known as parallel slack, that can be exploited by a multicore system. Figure 1 shows the runtime for the parallel code vs serial code. As is common with parallel code, for small values of  $n$ , the serial code outperforms the parallel. However, for larger  $n$  values, the 4x speedup is achieved. The  $S$  matrix then was formed using Matlab's Discrete Sine Transform(DST) command, `dst(x)`. Figure 1 shows the runtime for the built in DST command which proved to be slower than the parallel code. Thus for the remainder of this portion of the project, the parallel code was used when possible.

n	Serial (sec)	Parallel For(sec)	DST (sec)
258	0.0194	0.0494	0.5190
514	0.0111	0.0564	0.0196
1026	0.0423	0.0771	0.0706
2050	0.1704	0.1439	0.2003
4098	0.5087	0.4694	1.1024
8194	8.1068	1.9299	3.9583
16386	37.3268	7.9876	29.3784

Figure 1: Runtime for generating the  $S$

The  $S$  matrix acted as a preconditioner to the tridiagonal  $A$  matrix. A preconditioned is used to make a problem that is difficult to solve, simpler. The preconditioner is customized and tailored to the problem at hand. For this problem, preconditioning the  $A$  matrix by  $SAS^T$  results in a diagonal matrix. To achieve extremely fast convergence rates, the preconditioner matrix was split. After preconditioning  $A$  with the  $S$  matrix, the inverse,  $(SAS^T)^{-1}$ , was calculated and passed to the PCG method. As a result, the PCG method was able to achieve extremely fast convergence rates. While this method does require two matrix-matrix multiplications to setup the preconditioned  $A$  matrix, the resulting diagonal matrix was extremely easy to solve. The inverse of the diagonal matrix,  $(SAS^T)^{-1}$ , can be calculated in  $\mathcal{O}(n)$  time, as the inverse of a diagonal matrix is the inverse of each element on the diagonal. With the chosen preconditioner, each run of the PCG was able to converge in 1 iteration.

n	CG iterations	CG runtime (sec)	PCG iterations	PCG runtime (sec)
258	129	0.023895	1	0.294494
514	257	0.014975	1	0.023645
1026	513	0.193338	1	0.111805
2050	1025	2.468402	1	0.809114
4098	2049	18.554962	1	5.579513
8194	4097	145.323975	1	40.182528
16386	8193	1144.595261	1	326.499772

Figure 2: Runtimes and iteration counts for the CG and PCG methods

As can be seen in Figure 2, the PCG method is able to converge in one iteration and in much less time

than the CG method. When recording the timing for the PCG method, the time required to form the  $S$  matrix was not accounted for, but can be found in Figure 1. The recorded time for the PCG method did account for the two costly matrix multiplication operations, which were undoubtedly the source of much of the runtime. Regardless, the results are impressively conclusive. The PCG method is significantly faster than the regular CG method for this problem. The runtimes for both methods actually decrease for the  $n = 512$  interior mesh point case ( $n = 514$  total points). This is actually due to bandwidth/latency optimizations that are occurring at the processor/memory cache level. As  $n$  increases, the number of CG iterations also roughly doubles. However, for both the CG and the PCG method, the time to complete the method increase by a factor of 8. The primary driver of this increase is the matrix multiplication that occurs in each method.

## Part 2

*Perform the CG and PCG for Toeplitz systems using both Strang's and Chan's circulant matrices as the preconditioners.*

The a matrix is defined as a Toeplitz matrix if it has the following form:

$$A = \begin{bmatrix} a_0 & a_{-1} & \dots & a_{2-n} & a_{1-n} \\ a_1 & a_0 & a_{-1} & a_{2-n} & a_{1-n} \\ \vdots & a_1 & a_0 & \ddots & \vdots \\ a_{n-2} & & \ddots & \ddots & a_{-1} \\ a_{n-1} & a_{n-2} & \dots & a_1 & a_0 \end{bmatrix}$$

Toeplitz matrices are extremely useful and arise in solutions to differential and integral equations, as well as applications such as signal processing and optical analysis. A common special case of Toeplitz matrices are *circulant* matrices. A matrix is called circulant if it has the form of:

$$C = \begin{bmatrix} c_0 & c_{-1} & \dots & c_{-(n-2)} & c_{-(n-1)} \\ c_{-(n-1)} & c_0 & c_{-1} & \dots & \\ c_{-(n-2)} & c_{-(n-1)} & c_0 & & \vdots \\ \vdots & & & \ddots & \\ c_{-1} & c_{-2} & \dots & c_{-(n-1)} & c_0 \end{bmatrix}$$

These circulant matrices approximate and explain the behavior of Toeplitz matrices and thus are useful as preconditioners for the Preconditioned Conjugate Gradient (PCG) method. In order to understand the properties of circulant matrices, one must look at how their eigenvectors are created[2][4][5]. The eigenvalues,  $\lambda_k$ , and eigenvectors,  $\nu_k$  are given by  $C_n \nu = \lambda \nu$ . This can also be expressed as the  $n$  difference equations:

$$\sum_{k=0}^{m-1} c_{n-m+k} \nu_k + \sum_{k=m}^{n-1} c_k \nu_{k-(n-m)} = \lambda \nu_m \quad m = 0 : n-1$$

The above can be solved easily and since it is a linear equation with constant coefficients,  $\nu_k$  can be replaced with  $\nu_k = \rho^k$ . This is analogous to  $y(t) = e^{st}$  in time invariant differential equations. Substituting in  $\rho$  and canceling terms yields:

$$\sum_{k=0}^{n-1-m} c_k \rho^k + \rho^{-n} \sum_{k=n-m}^{n-1} c_k \rho^k = \lambda$$

We can see that by choosing values for rho, the corresponding eigenvectors will fall out. If we choose  $\rho = e^{\frac{-2\pi i m k}{n}}$ , the eigenvector that comes about is:

$$\nu_m = \frac{1}{\sqrt{n}} \left( 1, e^{\frac{-2\pi i m}{n}}, \dots, e^{\frac{-2\pi i m (n-1)}{n}} \right)$$

The curious thing about this particular value set for  $\rho$  is that this is the discrete Fourier Transform (DFT) of the sequence  $c_k$ . Thus, we can actually recover  $c_k$  by taking the inverse DFT of the eigenvalues,  $\lambda_k$ . Since the circulant matrix is made up of one row of values that have been shifted right as a function of their row number, the eigenvalues of  $C_n$  can be found by merely taking the DFT of the first column of  $C_n$ . This allows one to diagonalize a circulant matrix by the Fourier matrix,  $F_n$ :

$$C_n = F_n^* \Lambda_n F_n$$

where  $[F]_{j,k} = \frac{1}{\sqrt{n}} e^{\frac{2\pi i j k}{n}}$  for  $0 \leq j, k \leq n-1$  and  $\Lambda$  is a diagonal matrix holding the eigenvalues of  $C_n$ . The DFT operation is extremely efficient ( $\mathcal{O}(n \log n)$ ) compared to traditional matrix-vector multiplication ( $\mathcal{O}(n^2)$  at best). Once  $\Lambda$  has been obtained, the products  $C_n \vec{y}$  and  $C_n^{-1} \vec{y}$  can be computed in  $\mathcal{O}(n \log n)$ .

Circulant matrices can be used to speed up the Conjugate Gradient (CG) method, when solving a symmetric positive definite Toeplitz system:

$$A_n \vec{x} = \vec{b}$$

Two commonly used circulant preconditioners for speeding up the CG method are G. Strang's[3] circulant preconditioner and T. Chan's[5] circulant preconditioner. String's conditioner is defined by the rules:

$$\begin{cases} a_j & 0 \leq j \leq \lfloor \frac{n}{2} \rfloor \\ a_{j-n} & \lfloor \frac{n}{2} \rfloor < j < n \end{cases}$$

Where  $\lfloor \cdot \rfloor$  is the `floor()` algorithm. For a symmetric  $4 \times 4$  matrix, this would look like:

$$C_{strang} = \begin{bmatrix} c_0 & c_1 & c_2 & c_1 \\ c_1 & c_0 & c_1 & c_2 \\ c_2 & c_1 & c_0 & c_1 \\ c_1 & c_2 & c_1 & c_0 \end{bmatrix}$$

The diagonals containing the value  $c_1$  appear in the two corners, where as the  $A$  matrix had new, and possibly smaller, values  $a_{n-1}$ . In the Strang circulant, this information is left out. In order to precondition the  $A$  matrix, the inverse of Strang circulant must be applied to the  $A$  matrix. Inverting the Strang circulant is done with the following code:

```

1  a1=ifft(fft(Cn(:,1)).^ -1)'; %DFT first column of Cn, then invert, then iDFT
2  als=zeros(length(a1));
3      parfor j=1:length(A)
4          als(:,j)=circshift(a1,j-1,2); %Matlab's circular shifting algorithm
5      end

```

where  $C_n$  is the circulant, `fft` is the Fast Fourier Transform (FFT), and `ifft` is the inverse FFT. The Chan preconditioner is created in a similar manner.

The Chan circulant is defined as:

$$\begin{cases} \frac{(n-j)a_j + ja_{j-n}}{n} & 0 \leq j \leq n \\ a_{j-n} & \frac{n}{2} < j < n \end{cases}$$

which for a symmetric  $4 \times 4$  matrix would look like:

$$C_{chan} = \begin{bmatrix} c_0 & \alpha & c_2 & \alpha \\ \alpha & c_0 & \alpha & c_2 \\ c_2 & \alpha & c_0 & \alpha \\ \alpha & c_2 & \alpha & c_0 \end{bmatrix}$$

where  $\alpha = \frac{3a_1+a_3}{4}$ . For larger matrices the value of  $\alpha$  will change to follow the above set of rules. Unlike the Strang circulant, the information from  $a_3$  is not lost, but is incorporated in via the  $\alpha$  term. This is important as it allows this particular circulant to better mimic the original  $A$  matrix. This averaging term allows the spectrum of  $C^{-1}A$  to lie completely within that of the Strang circulant and hence potentially have a smaller condition number.

As the Chan circulant has the same eigenvalue properties as the Strang circulant, it can be inverted using the same code as presented above. Once inverted, the circulant can then be passed to the Preconditioned Conjugate Gradient method for processing.

The Preconditioned Conjugate Gradient method is nearly identical to the Conjugate Gradient method. In fact, they share 95% of the same code. The only real difference is in the preconditioner that is applied to the matrix  $A$  and the creation and use of a  $z$  parameter. The PCG method utilizes the stability of the CG method and attempts to further speed it up with a preconditioner. The role of the preconditioner in this process is to reduce the condition number of the problem so that it may be solved more readily. Iterative methods such as the PCG and CG methods are most suited for use with sparse matrices. If  $A$  is dense, it may be more worthwhile to attempt gaussian elimination, unless a proper preconditioner can be found. If a preconditioner,  $C$ , can be found that diagonalizes the dense matrix  $A$  then the much faster PCG method can be employed.

For this project, both the Strang ( $C_s$ ) and the Chan ( $C_c$ ) circulant matrices were used to condition two different  $A$  matrices. The solution to  $A\vec{x} = \vec{b}$  was then solved with the CG and PCG method and the results were compared. The first symmetric Toeplitz system solved was defined as:

$$a_k = |k+1|^{-p} \quad p = 2, 1, \frac{1}{10}, \frac{1}{100}$$

where  $a_k$  is the lower triangular portion of  $A_n$ . Each  $A$  matrix generated for this case was tested to see if it maintained its symmetric positive definite (SPD) characteristics. This was done by checking the eigenvalues of the system for each  $p$  value. Even for  $p \leq 1$ , this system did remain SPD.

For each value of  $p$ , an  $n \times n$   $A$  was generated, where the value of  $n$  were set to  $n = 50, 100, 200, 400, 800, 1600, 3200$ . Thus, for each value of  $p$ , 7 different  $A$  matrices were solved, generating a total of 28 solutions per iterative method tested. The value of  $\vec{b}$  for these systems was generated by a random number generator and changed for each iteration. To evaluate the performance of each iterative method, the wall clock time required to solve the system of equations was evaluated. Unfortunately, Matlab is unable to easily collect FLOPS statistics for a given function and wall clock time was required to be used for comparison.

The CG method was used as a baseline to compare PCG method with the Strang preconditioner and the Chan preconditioner. The results for the CG method are interleaved into the results for the two PCG methods, for easy of comparison. The PCG method using the Strang preconditioner is found in Figures 3, 4, 5, and 6. The PCG method used the inverse of the Strang preconditioner,  $C_s^{-1}$ , as the actual preconditioning matrix for this problem. Using the inverse of the  $C_s$  matrix results in drastically faster convergence rates for the PCG method.

Strang Preconditioner with p=2				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	5	17	0.000466383	0.000459963
100	5	19	0.000630086	0.000533019
200	5	21	0.001326217	0.000754883
400	5	21	0.004078817	0.001038358
800	5	21	0.016797975	0.002623913
1600	5	21	0.091472245	0.03101379
3200	5	21	0.315383559	0.120564579

Figure 3: The results of the CG iterative method and the PCG iterative method with a Strang circulant preconditioner and  $p=2$

Strang Preconditioner with p=1				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	6	24	0.000446854	0.000532729
100	6	29	0.000817126	0.001463598
200	6	36	0.003186477	0.001471787
400	7	43	0.003667413	0.002059279
800	7	49	0.016777624	0.006602936
1600	7	56	0.113062667	0.080089573
3200	7	62	0.417199922	0.32945909

Figure 4: The results of the CG iterative method and the PCG iterative method with a Strang circulant preconditioner and  $p=1$

As can be seen in Figures 3 and 4, for small values of  $n$ , and for the better conditioned matrices ( $p = 2, p = 1$ ) the CG method is actually marginally more efficient then the PCG method, when

only considering runtimes. However, when comparing the PCG number of iteration necessary to converge, the numbers are somewhat striking. The number of iterations required to converge for the PCG method was significantly less than the CG method. As the  $p$  value used in the creation of the  $A$  matrix decreases, the CG method rapidly loses its advantage but the PCG method retains its relatively quick convergence times. Figures 5 and 6 show how the number of iterations required for the CG method to converge increases. Unlike the CG method, the PCG method, with the Strang preconditioner, remains quite stable in the number of iterations required. This stability allows for the runtimes of the method to also remain small.

Strang Preconditioner with $p=1/10$				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	8	32	0.000468971	0.000616765
100	7	41	0.000699164	0.001121214
200	7	61	0.001852036	0.002006947
400	8	83	0.009854201	0.004022144
800	8	126	0.019734505	0.012925167
1600	8	165	0.128087046	0.213525804
3200	8	207	0.475609174	1.244652737

Figure 5: The results of the CG iterative method and the PCG iterative method with a Strang circulant preconditioner and  $p=\frac{1}{10}$

Strang Preconditioner with $p=1/100$				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	8	33	0.000574337	0.000754939
100	7	48	0.000748602	0.001653844
200	7	66	0.001587019	0.002245573
400	7	90	0.004442643	0.004294148
800	7	136	0.020105372	0.015915269
1600	8	224	0.114234595	0.26337328
3200	7	265	0.426624346	1.58550617

Figure 6: The results of the CG iterative method and the PCG iterative method with a Strang circulant preconditioner and  $p=\frac{1}{100}$

By analyzing the runtimes of the PCG method over the various runs and with multiple  $p$  values, an overall growth rate of the runtimes can be determined. Ideally, the PCG method runtimes grow at a rate of  $\mathcal{O}(n \log n)$ . For many runs, the growth pattern was  $\mathcal{O}(3n \log n) \geq \mathcal{O}(x) \geq \mathcal{O}(\frac{1}{2} n \log n) \approx \mathcal{O}(n \log n)$  where  $x$  is the growth rate for a given subset of runs. Since the number of iterations remained fairly constant, calculating these qualities becomes much more straight forward. Thus, the PCG method with the Strang circulant achieved a good speedup over the traditional CG method when the  $A$  matrix became less friendly.

The PCG method utilizing the Chan preconditioner is found in Figures 7, 8, 9, and 10. As can be seen in the two Figures, the PCG method is faster than the CG method. Close analysis of the use of Chan's preconditioner vs Strang's preconditioner reveals that both methods are roughly equivalent. The runtimes for both methods are nearly identical for the two preconditions, as are the number of

Chan Preconditioner with p=2				
n	PCG iteration	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	5	17	0.000462841	0.000459963
100	5	19	0.000707112	0.000533019
200	5	21	0.000935643	0.000754883
400	5	21	0.002468322	0.001038358
800	5	21	0.014531127	0.002623913
1600	5	21	0.085933745	0.03101379
3200	5	21	0.316112478	0.120564579

Figure 7: The results of the CG iterative method and the PCG iterative method with Chan's circulant preconditioner and  $p=2$

Chan Preconditioner with p=1				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	6	24	0.000488519	0.000532729
100	6	29	0.000805501	0.001463598
200	6	36	0.00129328	0.001471787
400	7	43	0.002867162	0.002059279
800	7	49	0.019386416	0.006602936
1600	7	56	0.115887958	0.080089573
3200	7	62	0.427869129	0.32945909

Figure 8: The results of the CG iterative method and the PCG iterative method with Chan's circulant preconditioner and  $p=1$



iterations required to converge. As stated before, the structure of the Chan preconditioner allows it to be in the subspace of the Strang preconditioner and thus has a better condition number. This theoretically allows for a faster convergence of the iterative method. However, for the problem size and complexity of this project, the two methods were roughly equivalent.

Chan Preconditioner with p=1/10				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	8	32	0.000561755	0.000616765
100	7	41	0.000769179	0.001121214
200	7	61	0.001229141	0.002006947
400	8	83	0.003507623	0.004022144
800	8	126	0.020692512	0.012925167
1600	8	165	0.126109452	0.213525804
3200	8	207	0.475551361	1.244652737

Figure 9: The results of the CG iterative method and the PCG iterative method with Chan's circulant preconditioner and  $p=\frac{1}{10}$

Chan Preconditioner with p=1/100				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	8	33	0.000618757	0.000754939
100	7	48	0.000800265	0.001653844
200	7	66	0.001232447	0.002245573
400	7	90	0.002785346	0.004294148
800	7	136	0.018245726	0.015915269
1600	8	224	0.134216589	0.26337328
3200	7	265	0.483545973	1.58550617

Figure 10: The results of the CG iterative method and the PCG iterative method with Chan's circulant preconditioner and  $p=\frac{1}{100}$

The second symmetric Toeplitz system is defined by:

$$a_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\theta) e^{-ik\theta} d\theta, \quad k = 0, \pm 1, \pm 2, \dots,$$

where  $f(\theta) = \theta^4 + 1$  for  $-\pi \leq \theta \leq \pi$ . This means that  $a_k$  is the Fourier coefficients of  $f(\theta)$  and were computed via FFT. For the second system, the PCG and CG method were each run for the 6 different values of  $n$  only. Once again, the CG method was compared to the PCG method with Chan's preconditioner and Strang's preconditioner. For this new  $A$  matrix based upon the Fourier coefficients of  $f(\theta)$ . These coefficients were generated by first creating an array of  $f(\theta)$  values, and then performing a FFT on the values. The result was an array of complex values that were then used to generate the  $A$  matrix. Once the  $A$  matrix had been formed, Chan and Strang's circulant matrices were generated. In order to achieve fast convergence rates, the  $A$  matrix was preconditioned with each of these circulant matrices. The final preconditioned matrix passed to the PCG method was  $C_n A C_n'$  and the matrix used in the PCG calculation was  $(C_n A C_n')^{-1}$ . Figures 11 and 12 show the final runtimes and number of iterations used by each method.

PCG method with Strang's preconditioner				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	1	58	0.000369356	0.002065425
100	1	128	0.000257824	0.005651115
200	1	274	0.000469347	0.016995163
400	1	592	0.001852011	0.220975618
800	1	1202	0.010599408	1.585321792
1600	1	2450	0.057285458	13.87002449
3200	1	4966	0.267209028	112.810705

Figure 11: The final runtimes of the CG iterative method and the PCG iterative method with a Strang circulant preconditioner.

As can be seen in Figures 11 and 12, the PCG method greatly outpaces the CG method in both runtime and number of iterations required to converge. However, once again the Chan and Strang circulant preconditioners are roughly equivalent. Both methods are able to converge within one iteration and both methods are able to handle the large matrices without issue. A brief analysis of the runtimes show that the  $\approx \mathcal{O}(n \log n)$  growth rate discussed above is maintained with the second symmetric Toeplitz system analyzed here.

PCG method with Chan's preconditioner				
n	PCG iterations	CG Iterations	PCG runtime (s)	CG Runtime(s)
50	1	58	0.000209005	0.002065425
100	1	128	0.000296378	0.005651115
200	1	274	0.000897669	0.016995163
400	1	592	0.002127928	0.220975618
800	1	1202	0.011197852	1.585321792
1600	1	2450	0.054131305	13.87002449
3200	1	4966	0.277730117	112.810705

Figure 12: The final runtimes of the CG iterative method and the PCG iterative method with a Chan circulant preconditioner.

For this project, the Conjugate Gradient method and Preconditioned Conjugate Gradient method were compared and contrasted for multiple problems. The effects of different circulant matrices were explored it was found that for the given trade space of problems, Chan's preconditioner and Strang's preconditioner were both equally effective. Regardless of this relative equality, the implications of this project are quite clear. If a proper preconditioner matrix is able to be constructed, the PCG method can solve a large, complex system of equations extremely quickly and efficiently.

## References

- [1] J. Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Carnegie Mellon University 1994.
- [2] R. Gray, *Toeplitz and Circulant Matrices: A review*. Stanford University 2006.
- [3] G. Strang, *A Proposal for Toeplitz Matrix Calculations*. Stud. Appl. Math., 74(1986), pp171-176.
- [4] R. Chan and G. Strang, *Toeplitz Equations by Conjugate Gradients with Circulant Preconditioner*. SIAM J. Sci. Stat. Comput., 10(1989), pp 104-117.
- [5] T. Chan, *An Optimal Circulant Preconditioner for Toeplitz Systems*. SIAM J. Sci. Stat. Comput. 9 (1988) pp 766-771.