MATH 6644
Project 1

Stephan Boettcher

March 15, 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 one-dimensional equation given above can be solved with numeral method analysis or directly as a function of x. The solved equation then can be used to check the validity of the numerical methods. When solved directly, the equation becomes:

$$u(x) = -0.333333x^3 + 0.25x^2 - 1.91667x + 1$$

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:

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

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 results of the CG method can be found in Figure 2. Figure **??** shows the CG method approaching 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. This was achieved with the following code:

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

This code was run with 4 workers in parallel, allowing for a theoretical 4x improvement over the serial code. 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.

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.

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

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

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, λ_k, and eigenvectors, ν_k are given by $C_n\nu = \lambda\nu$. This can also be expressed as the n

difference equations:

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

The above can be solved easily and since it is a linear equation with constant coefficients, ν_k can be replaced with $\nu_k = \rho^k$. This is analogous to $y(t) = e^{st}$ in time invariant differential equations. Substituting in ρ 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 $\rho = e^{\frac{-2\pi imk}{n}}$, the eigenvector that comes about is:

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

The curious thing about this particular value set for ρ 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, λ_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 $[F]_{j,k} = \frac{1}{\sqrt{n}}e^{\frac{2\pi ijk}{n}}$ for $0 \le j,k \le n-1$ and Λ 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 Λ has been obtained, the products $C_v\vec{y}$ 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:

$$\begin{cases} a_j & 0 \le j \le [\frac{n}{2}] \\ a_{j-n} & [\frac{n}{2} < j < nc_{n+j}0 < -j < n \end{cases}$$

Where $[\ \cdot\ ]$ is the `floor()` algorithm. For a symmetric 4×4 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. The inverse of Strang circulant can be applied to the A matrix in order to precondition it. Inverting the Strang circulant is done with the following code:

```
a1=ifft(fft(Cn(:,1)).^-1)'; %DFT first column of Cn, then invert, then iDFT
a1s=zeros(length(a1));
      parfor j=1:length(A)
            a1s(:,j)=circshift(a1,j-1,2); %Matlab's circular shifting algorithm
      end
```

where C_n is the circulant, `fft` is the Fast Fourier Transform (FFT), and `ifft` is the inverse FFT. Once inverted, the circulant can then be passed to the Preconditioned Conjugate Gradient method for processing. The Chan preconditioner is created in a similar manner.

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

which for a symmetric 4×4 matrix would look like:

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

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.