MATH466/462 Project 4. *Due in class on Wed, Mar 18, 2020*

Instruction: your project report should include necessary mathematical justification, description, and details of your algorithms/conclusions. Your MATLAB codes and generated outputs may be attached in the end of the report. Make sure you addressed all the questions in each problem. Both the report and codes will be graded. Please submit a printed hard-copy.

Problem A (20 pts): Circulant Preconditioners for Toeplitz Systems

Toeplitz matrix arises in many different applications. Its matrix-vector product can be computed efficiently via fast Fourier transform (FFT). Many circulant preconditioners have been proposed for solving Toeplitz systems. In this project, we will compare several circulant preconditioners for solving various Toeplitz systems.

Task 1: Study Toeplitz matrix, Circulant matrix, and the use of FFT (read our Textbook Chap 10.1).

Read Chapters 1 and 3: https://ee.stanford.edu/~gray/toeplitz.pdf

Understand more on FFT: https://arxiv.org/pdf/1805.05533v2.pdf

Task 2: play and understand the following codes:

For any Circulant matrix C, the matrix-vector products $x = C^{-1}b$ can also be computed via FFT:

```
n=10; C=gallery('circul',(1:n));%Construct a Circulant matrix using (1:n) as 1st row
b=ones(n,1); x2=C\b; %Direct solve: O(n^3) operations
ev=fft(C(:,1));%the eigenvalues of C by FFT of its first column
x1= ifft(fft(b)./ev); %Solve C\b using FFT: O(n log n) operations
norm(x1-x2,inf) %should be zero
```

By embedding a Toeplitz matrix T into a Circulant matrix, the product Tv can also be computed via FFT:

```
n=10;t=(n:-1:1); T=toeplitz(t,t'); %construct a full symmetric Toeplitz matrix: T'=T
v=rand(n,1);y2=T*v;%compute y2=T*v using direct multiplication: O(n^2) operations
gev = fft([t 0 t(n:-1:2)].');%the eigenvalues of the embeding larger Circulant matrix
y = ifft(fft([v;zeros(n,1)]).*gev);%compute y1=T*v using FFT: only O(n log n) operations
y1 = y(1:n); %take the first half of the long vector
norm(y1-y2,inf) %should be close to zero
```

Task 3: understand the construction of 3 circulant preconditioners.

Let T_n be an n-by-n Toeplitz matrix with $T_n(i,j) = t_{i-j}$, where $\{t_k\}_{k=1-n}^{n-1}$ are given diagonals. We can define at least 3 different circulant preconditioners as follows:

1. **Strang's Preconditioner**: Strang's preconditioner S_n with $S_n(i,j) = s_{i-j}$ is defined to be the circulant matrix obtained by copying the central diagonals of T_n and bringing them around to complete the circulant requirement. Assume $n = 2m$ is even, the diagonals s_k of S_n are given by

$$s_k = \begin{cases} t_k & \text{if } 0 \le k \le m-1 \\ 0 & \text{if } k = m \\ t_{k-n} & \text{if } m < k \le n-1 \\ s_{-k} & \text{if } (1-n) \le k < 0 \end{cases}.$$

2. **T. Chan's Preconditioner**: T. Chan's preconditioner C_n with $C_n(i,j) = c_{i-j}$ is defined through minimizing the difference between T_n and C_n over all circulat matrices. The diagonals c_k of C_n are given by (taking $t_{-n} = 0$)

$$c_k = \begin{cases} \frac{(n-k)t_k + k t_{k-n}}{n} & \text{if } 0 \le k \le n-1 \\ c_{n+k} & \text{if } (1-n) \le k < 0 \end{cases}.$$

3. **R. Chan's Preconditioner**: R. Chan's preconditioner R_n with $R_n(i,j) = r_{i-j}$ is defined to make uses of all the entries of T_n. The diagonals r_k of R_n are given by (taking $t_{-n} = 0$)

$$r_k = \begin{cases} t_k + t_{k-n} & \text{if } 0 \le k \le n-1 \\ r_{-k} & \text{if } (1-n) \le k < 0 \end{cases}.$$

(1) Use our PCG (`mypcgfun.m`) to solve the SPD Toeplitz systems $T_n x = b$ with $b = ones(n, 1)$ and

$$T_n(i,j) = \begin{cases} \pi^2/3 & \text{if } i == j \\ \frac{2(-1)^{j-i}}{(j-i)^2} & \text{if } i \neq j \end{cases}.$$

Test with no preconditioner and the above circulant preconditioners, and compare their iteration numbers and CPU times for different dimensions `n=1e3*(1:5)` (set max 10000 iterations and tolerance 10^{-7}).

You can start with construction of all related full matrices for smaller n, but eventually all matrix-vector products should be replaced by only FFT based codes for better efficiency and lower memory costs.

(2) Circulant preconditioners are attrative since they can be solved efficiently ($O(n \ln n)$ operations) via FFT. As we aready know, a tridiagonal matrix can also be solved very efficiently ($O(n)$ operations) via Thomas algorithm. Run your codes again with the following tridiagonal preconditioner, which works well for the given T_n:

```
Pn=gallery('tridiag',n,-1,2,-1);
```

For benchmarking your own codes, below are the iteration numbers based on my implementation:

n	None	Strang	T. Chan	R. Chan	Tridiag
1000	746	8	28	7	14
2000	1522	8	35	7	14
3000	2301	8	41	7	14
4000	3081	8	47	7	14
5000	3862	8	50	7	14

(3) Solve the same system using the Gaussian elimination method (`GEsolver.m`), what you observe in terms of CPU times growth, in comparison with the above PCG solvers?