1 Introduction

This report's focus is on eigenvalue and eigenvector (eigenpair) problems of a coefficient matrix.

$$A * x = \lambda * x$$

We have power methods, for finding the largest (or the smallest with inverse option) eigenpairs. These methods can only handle one eigenpair. However, there are also simultaneous subspace iterations methods for finding eigenpairs till k-th order.

- The Plain Simultaneous Subspace Iterations (SSI)
- Simultaneous Inverse Subspace Iterations (SII)

SSI is used for finding the largest eigen pairs, on the other hand, SII is used for the smallest eigenpairs. The MATLAB code plots the timing measurements of these methods, besides MATLAB built-in eigs() function result. Test matrices are "LUND A", "BFW62 B", and "PLAT362".

2 Implementation

```
clear all;
close all;
A = mmread('inputs/plat362.mtx');

%% parameters
k = 1;
X = ones(length(A), k);
max_iter = 100000;
tol = 1e-12;

%% matlab eigs functions for finding largest and smallest eigenpairs
tic;
[V1, D1] = eigs(A, [], k, 'largestabs');
eigs_max_time = toc;
V1 = abs(V1);
tic;
[V2, D2] = eigs(A, [], k, 'smallestabs');
V2 = abs(V2);
eigs_min_time = toc;

%% ssi(subspace iterations) and sii(subspace inverse iterations) algorithms
```

```
i = 0;
tic;
while (i < max_iter) & tol < norm(abs(V1) - abs(X))
Z = A*X;
[X, R] = qr(Z, 0);
i = i+1;
end
iteration_ssi = i;
ssivec = abs(X);
for i = 1:k
ssival_(:, 1) = A * X(:, i);
ssival(i,i) = ssival_(1, 1) / X(1, i);
end
ssi_time = toc;

X = ones(length(A), k);
i = 0;
tic;
A = inv(A);
while (i < max_iter) & tol < norm(abs(V2) - abs(X))
Z = A*X;
[X, R] = qr(Z, 0);
i = i+1;
end
iteration_sii = i;
siivec = abs(X);
for i = 1:k
siival_(:, 1) = A * X(:, i);
siival(i,i) = 1 / (siival_(1, 1) / X(1, i));
end
sii_time = toc;

%% figures
xa = categorical({'eigs()_l_a_r_g_e_s_t Time Spent','SSI Time ...
    Spent','eigs()_s_m_a_l_l_e_s_t Time Spent','SII Time Spent'});
xa = reordercats(xa, {'eigs()_l_a_r_g_e_s_t Time Spent','SSI Time ...
    Spent','eigs()_s_m_a_l_l_e_s_t Time Spent','SII Time Spent'});
ya = [eigs_max_time, ssi_time, eigs_min_time, sii_time];
b = bar(xa,ya, 0.1);
xtips1 = b(1).XEndPoints;
ytips1 = b(1).YEndPoints;
labels1 = string(b(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','center',...
'VerticalAlignment','bottom')
title('Time Consumption Comparison');
subtitle('Built-in eigs() with SSI and SII, tolerance value is 1e-12');
ylabel('Time (sec)')
ylim([0, max(ya)+max(ya)/10])
grid minor
```

Main code including all algorithms

Firstly parameters are set. Then, eigs() function are run with recording time for two of the methods. Then implemented algorithms are run with recording time. At the result, some eigenvector components are resulted in negative sign as regards to eigs() results. Therefore; abs() function is called for eigenvectors. At the and, plotting functions take part.

Computing Platform

Processor: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz
RAM: 16.0 GB (15.7 GB usable)
OS: Windows 11 Home
Software: MATLAB R2021b
version -blas: Intel(R) Math Kernel Library Version 2019.0.3 Product Build 20190125 for Intel(R) 64 architecture applications, CNR branch AVX512_E1
version -lapack: Intel(R) Math Kernel Library Version 2019.0.3 Product Build 20190125 for Intel(R) 64 architecture applications, CNR branch AVX512_E1, supporting Linear Algebra PACKage (LAPACK 3.7.0)

Test Matrices

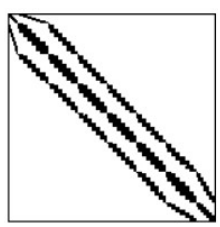

Figure 1: **LUND A, Original Harwell sparse matrix test collection**
147 x 147, 1298 entries

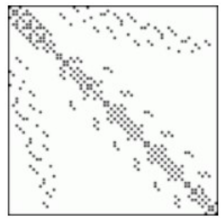

Figure 2: **BFW62B, Bounded Finline Dielectric Waveguide**
62 x 62, 342 entries

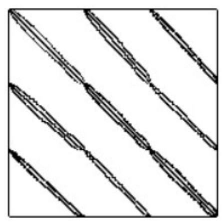

Figure 3: **PLAT362, Platzman's oceanographic models North Atlantic submodel**
362 x 362, 3074 entries

3 Results

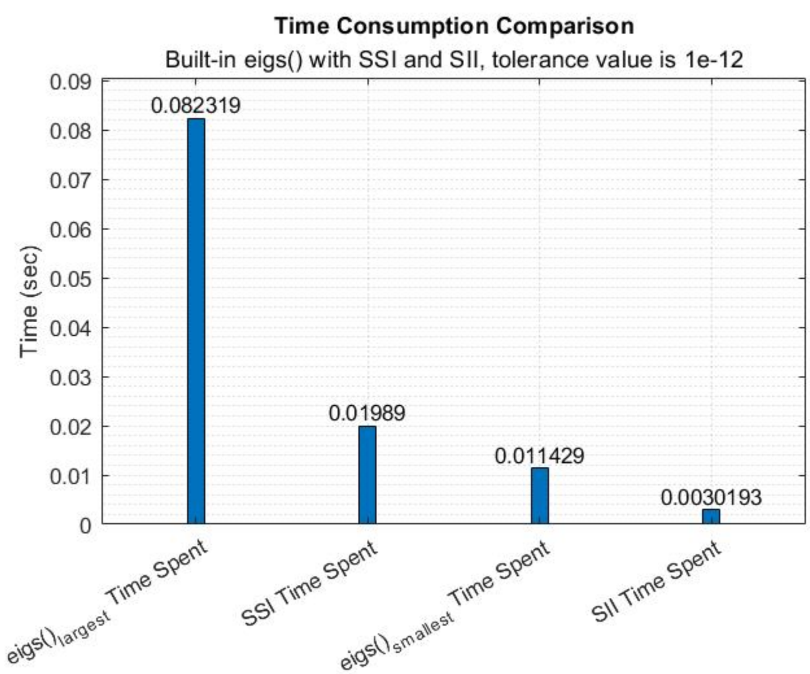

Figure 4: **LUND A, k = 1**
Iteration Number for SSI = 1687, Iteration Number for SII = 9

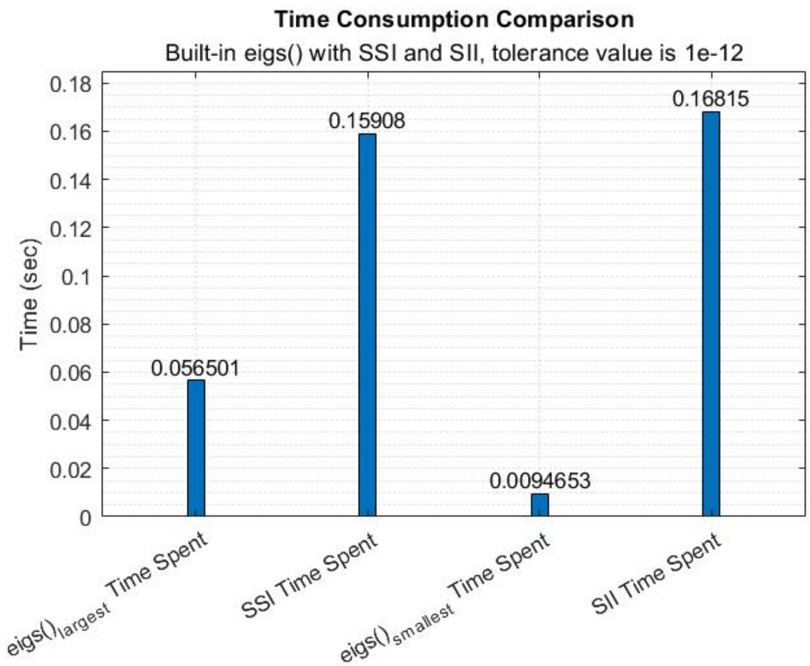

Figure 5: **LUND A, k = 3**
Iteration Number for SSI = 5041, Iteration Number for SII = 2833

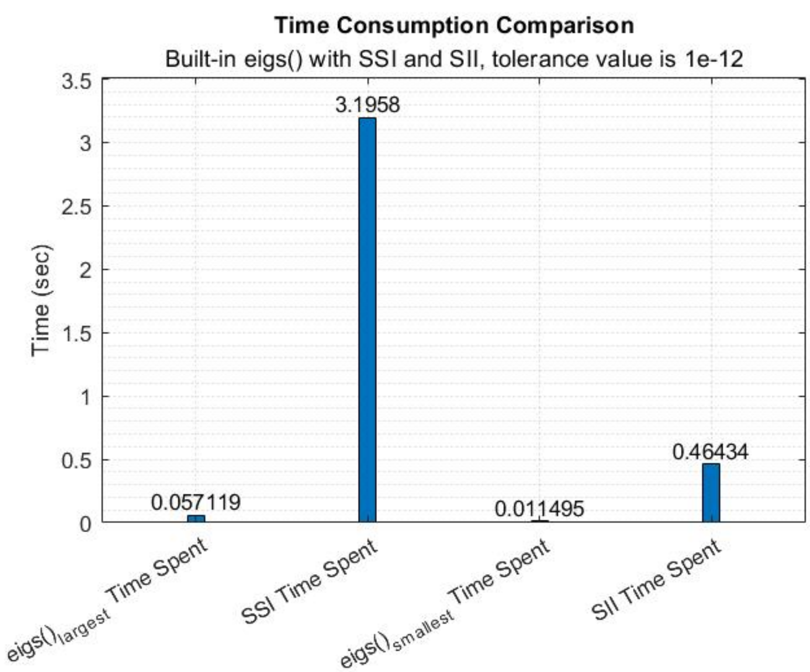

Figure 6: **LUND A, k = 10**
Iteration Number for SSI = 34269, Iteration Number for SII = 2834

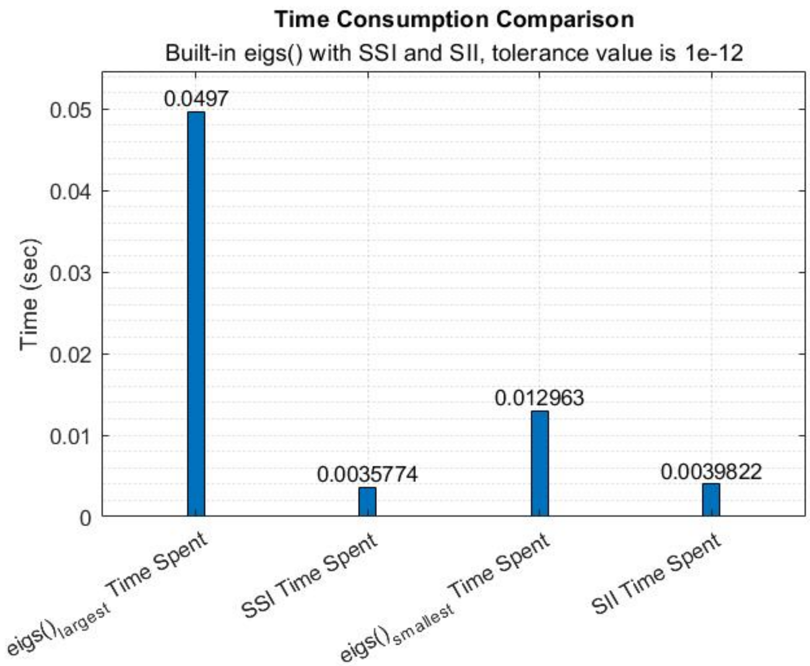

Figure 7: **BFW62B, k = 1**
Iteration Number for SSI = 1149, Iteration Number for SII = 905

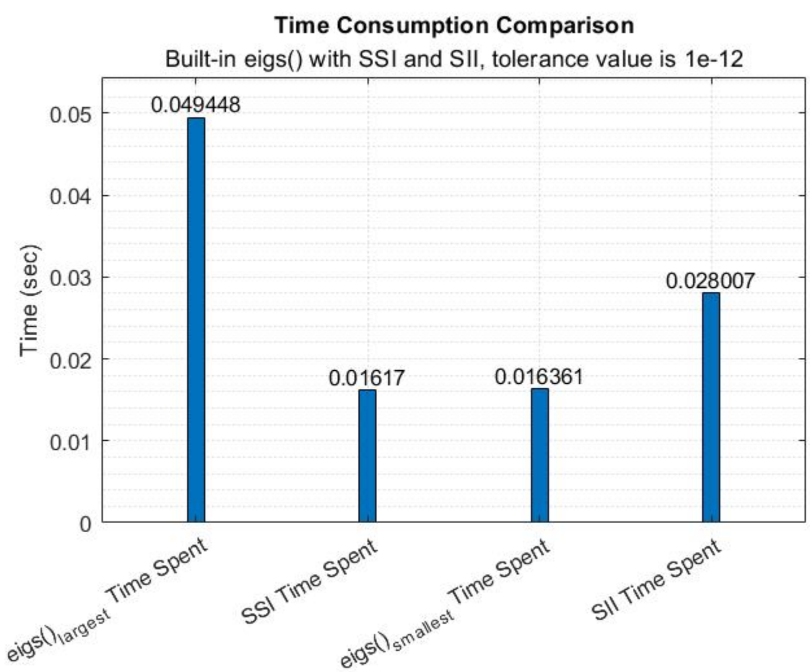

Figure 8: **BFW62B, k = 3**
Iteration Number for SSI = 2721, Iteration Number for SII = 3790

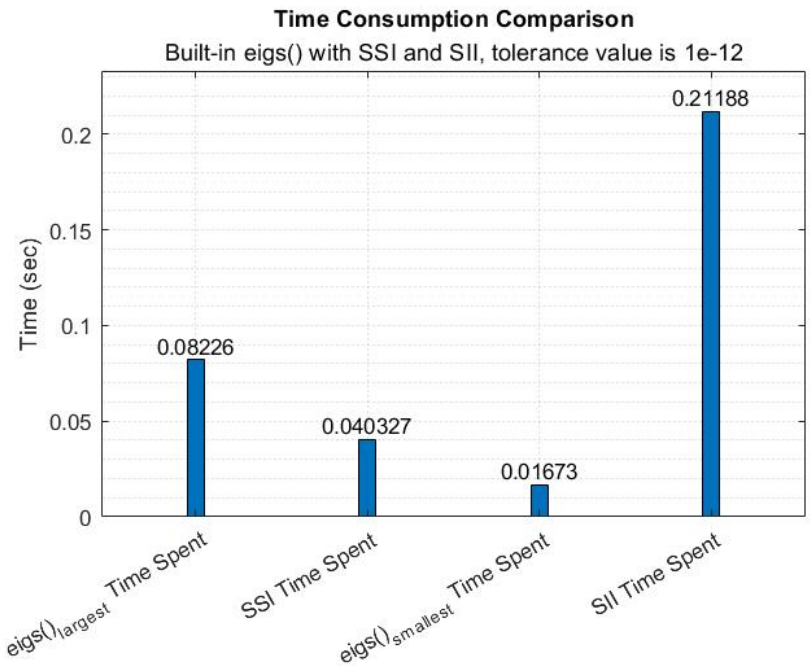

Figure 9: **BFW62B, k = 10**
Iteration Number for SSI = 2721, Iteration Number for SII = 11922

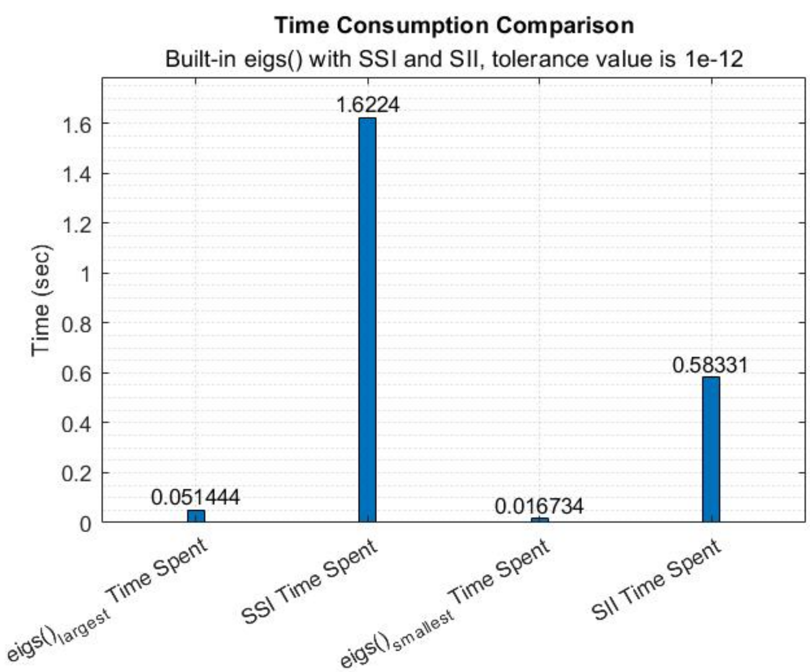

Figure 10: **BFW62B, k = 25**
Iteration Number for SSI = 36317, Iteration Number for SII = 11850

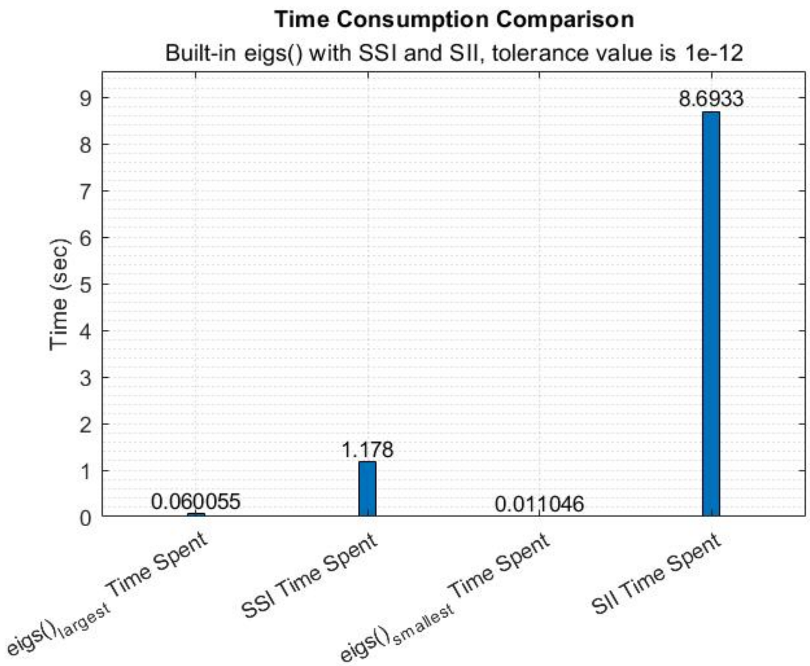

Figure 11: **PLAT362, k = 1**
Iteration Number for SSI & SII = 100000 (max iterations allowed)
PLAT362 coefficient matrix does not converge in implemented methods; however, eigs() function can solve that matrix.

SSI is basically finds the largest eigenpairs. Since it always make operations with k-column matrices, the multiplication operation started to take long time as k increases. SII is a mixed method composed of inverse power method, and SSI. It finds the smallest eigenpairs since we take the inverse of the A matrix. Same situation happens for SII. They do not utilize parallelism.

MATLAB built-in eigs() function, on the other hand, makes an intense parallelism. It seems that when k=1, implemented SSI and SII even shows better timing performance than eigs() function. As k increases, the effectiveness of parallelism increases.

Proposed method is, we separate each eigenpairs by shifting the original matrix A with different amount. After that, we algorithms work each column independently by different processor units. However, by this way, we can lose some eigen pairs or we can reach the same pairs from different parallel processes.

Acknowledgements

To convert matrices taken from the Matrix Market, ”MatrixMarket I/O Functions for Matlab, mmread.m” function is included to the homework codes.
(https://math.nist.gov/MatrixMarket/mmio/matlab/mmiomatlab.html)
1st link tells that the eigs() function’s parallelism approach.,

References

[1] Parallelism on eigs() function:
https://www.mathworks.com/help/matlab/ref/eigs.html

[2] Improvement guide to implement proposed method:
https://www.mathworks.com/help/parallel-computing/parallel.gpu.cudakernel.html

[3] LUND A matrix:
https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/smtape/lund_a.html

[4] BFW62B matrix:
https://math.nist.gov/MatrixMarket/data/NEP/bfwave/bfw62b.html

[5] PLAT362 matrix:
https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/platz/plat362.html