ECS231 Final Project

Yu-Cheng Hwang
918954206

June 2022

1 Introduction

This project mainly focuses on Arnoldi Algorithm. Arnoldi Algorithm has two ways to find out eigenvalues and eigenvectors. One is Standard Arnoldi, we use arbitrary vector V to calculate Gram-Schmidt for k times to retrieve eigenvalues. The other one is re-orthogonal method. Because there are some errors(or deviation) when we calculate Gram-Schmidt, we need to proceed Gram-Schmidt procedure again to reduce the deviation. Lastly, we want to implement $A - \tau I$ to find out shift and inverse eigenvalues and eigenvectors.

2 Arnoldi Algorithm without re-orthogonalization

2.1 Code

```
load west0479
A = west0479;
%k = 20;

% exact eigenvalue
lam = eig(full(A));
figure(1)
hold on
plot(real(lam),imag(lam),'r+');

idx = 1;
%% start calculate range k
for k = 10:10:100
n = length(A);
V = zeros(n,k); % orthonormal basis
H = zeros(k,k); % upper Hessenberg matrix
v = ones(n,1);
```

```

V(:,1) = v/norm(v);

% Gram-Schmidt
for j = 1:k
V(:,j+1) = A*V(:,j); % compute w

for i = 1:j
H(i,j) = V(:,i)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-H(i,j)*V(:,i);
end
% normalization
H(j+1,j) = norm(V(:,j+1));

if H(j+1,j) == 0
    break;
else
V(:,j+1) = V(:,j+1)/H(j+1,j);
% compute residual
Ra(j,idx) = norm(A*V(:,j) - V(:,j+1)*H(:,j)');
Rb(j,idx) = norm(speye(k) - V(:,j+1)'*V(:,j+1));
end

end

H(k+1,:) = [];
Rz = eig(full(H));
plot(real(Rz),imag(Rz),'bo');

% calculate relative errors
lumk = max(lam);
muk = max(Rz);
relerrar(idx,:) = abs(lumk - muk) / abs(lumk);
idx = idx + 1;

%% end of the range k
end
hold off
```

Listing 1: Arnoldi-without-re-orthogonalization.m

2.2 Results

2.2.1 Real Eigenvalues

As we can see in the picture, red cross is the original eigenvalues calculated by Matlab, blue circle is the Arnoldi Algorithm's eigenvalues.

Arnodi Algorithm first calculate the eigenvalue from the outside, then converge into inner circle as we see in the figure.

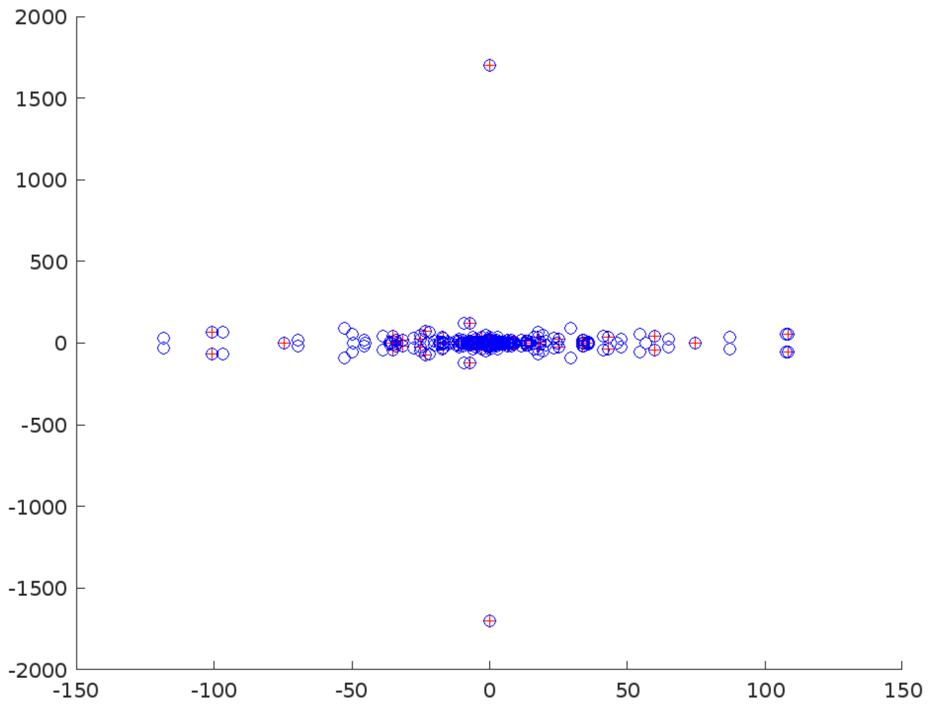


Figure 1: Real Eigenvalues and Arnoldi

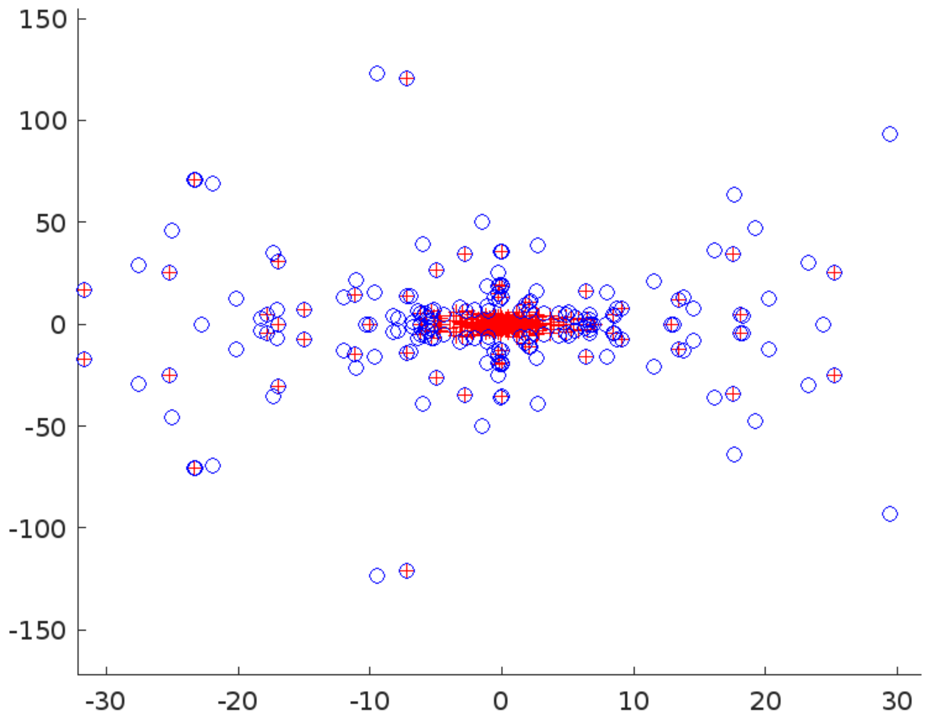


Figure 2: Closer Look

2.2.2 Residual

- $||AV_j - V_{j+1}\hat{H}_j||$

  When j increases from like 10 to 20, it will increase from $1.5216e^3$ to $5.3781e^3$, however, when we increase j to like 30, 40...till 100, the residual will become smaller to e^5, for example, when $j = 100$, the value will be $3.1201e^5$.

- $||I - J_{j+1}^H V_{j+1}||$

  For $j = 10, 20, 30...100$, the residual value will be $9, 19, 29...99$.

3 Arnoldi Algorithm with re-orthogonalization

3.1 Code

```
load west0479
A = west0479;
%k = 20;

% exact eigenvalue
lam = eig(full(A));
figure(2)
hold on
plot(real(lam),imag(lam),'r+');

idx = 1;
%% start calculate range k
for k = 10:10:100
n = length(A);
V = zeros(n,k); % orthonormal basis
H = zeros(k,k); % upper hessenberg matrix
v = ones(n,1);

V(:,1) = v/norm(v);

% Gram-Schmidt
for j = 1:k
V(:,j+1) = A*V(:,j); % compute w

for i = 1:j
H(i,j) = V(:,i)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-H(i,j)*V(:,i);
end
% normalization
H(j+1,j) = norm(V(:,j+1));

% -->reorthogonalization process HERE?
% without any if statement
for l = 1:j
```

```
mu = V(:,l)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-V(:,l)*mu;
H(l,j) = H(l,j) + mu;
end
H(j+1, j) = norm(V(:,j+1));

V(:,j+1) = V(:,j+1)/H(j+1,j);
% compute residual
Ra(j,idx) = norm(A*V(:,j) - V(:,j+1)*H(:,j)');
Rb(j,idx) = norm(speye(k) - V(:,j+1)'*V(:,j+1));
end

H(k+1,:) = [];
Rz = eig(full(H));
plot(real(Rz),imag(Rz),'bo');

% calculate relative errors
lumk = max(lam);
muk = max(Rz);
relerror(idx,:) = abs(lumk - muk) / abs(lumk);
idx = idx + 1;
%% end of the range k
end
hold off
```

Listing 2: Arnoldi-with-re-orthogonalization.m

3.2 Results

3.2.1 Real Eigenvalues and Arnoldi Reorthogonalization Algorithms

As we can see in the figure 2, the result is very close to Arnoldi without orthogonalization, but we will inspect the relative error to see the differences between both of them.

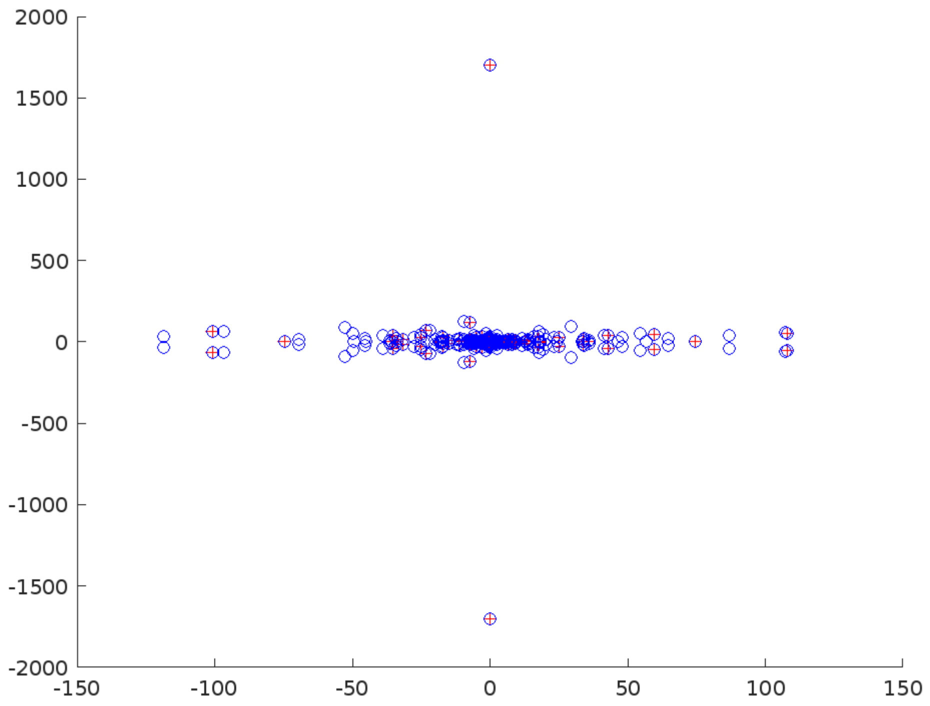


Figure 3: Real Eigenvalues and Arnoldi Reorthogonalization

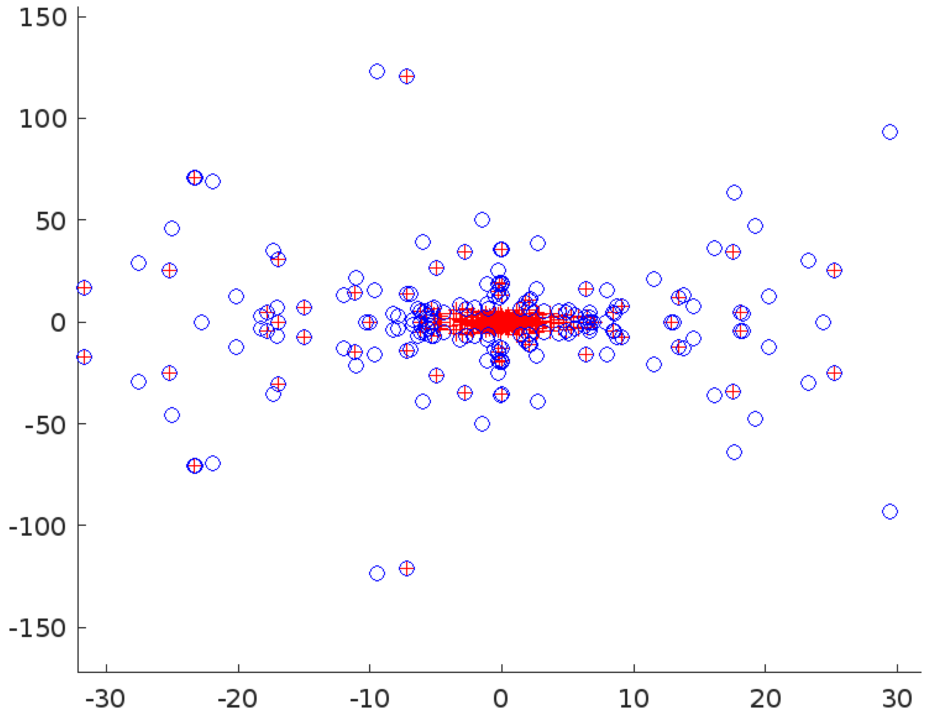


Figure 4: Closer Look

3.2.2 Residual

- $||AV_j - V_{j+1}\hat{H}_j||$

  We can find out that residual results are similar to Standard Arnoldi(without orthogonalization).When j increases from like 10 to 20, it will increase from $1.5216e^3$ to $5.3781e^3$, however, when we increase j to like 30, 40...till 100, the residual will become smaller to e^5, for example, when $j = 100$, the value will be $3.1201e^5$.

- $||I - J_{j+1}^H V_{j+1}||$

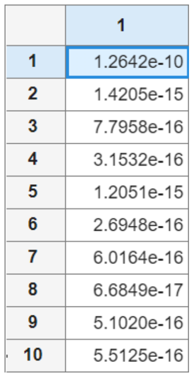
  For $j = 10, 20, 30...100$, the residual value will be $9, 19, 29...99$.

4 Differences between Standard and re-Orthogonalization

4.1 Relative Errors

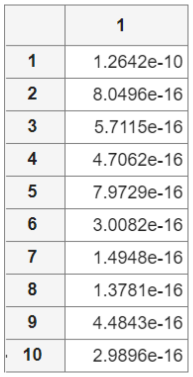
We can use relative errors to compare differences between Standard Arnoldi and re-orthogonalization.

$$\frac{|\lambda_k - \mu_k^{(j)}|}{|\lambda_k|} \; for \; k = 1, 2, 3...$$



	1
1	1.2642e-10
2	1.4205e-15
3	7.7958e-16
4	3.1532e-16
5	1.2051e-15
6	2.6948e-16
7	6.0164e-16
8	6.6849e-17
9	5.1020e-16
10	5.5125e-16

Figure 5: Standard Arnoldi



	1
1	1.2642e-10
2	8.0496e-16
3	5.7115e-16
4	4.7062e-16
5	7.9729e-16
6	3.0082e-16
7	1.4948e-16
8	1.3781e-16
9	4.4843e-16
10	2.9896e-16

Figure 6: re-Orthogonalization Arnoldi

Relative Errors

As we can see, for Standard Arnoldi, the relative error will decrease from e^{-10} to e^{-16} when we increase our times of loop k from 10 to 100. On the other hand, with re-orthogonalization, the relative error will decrease more than Standard Arnoldi.

4.2 Relative Errors Figures

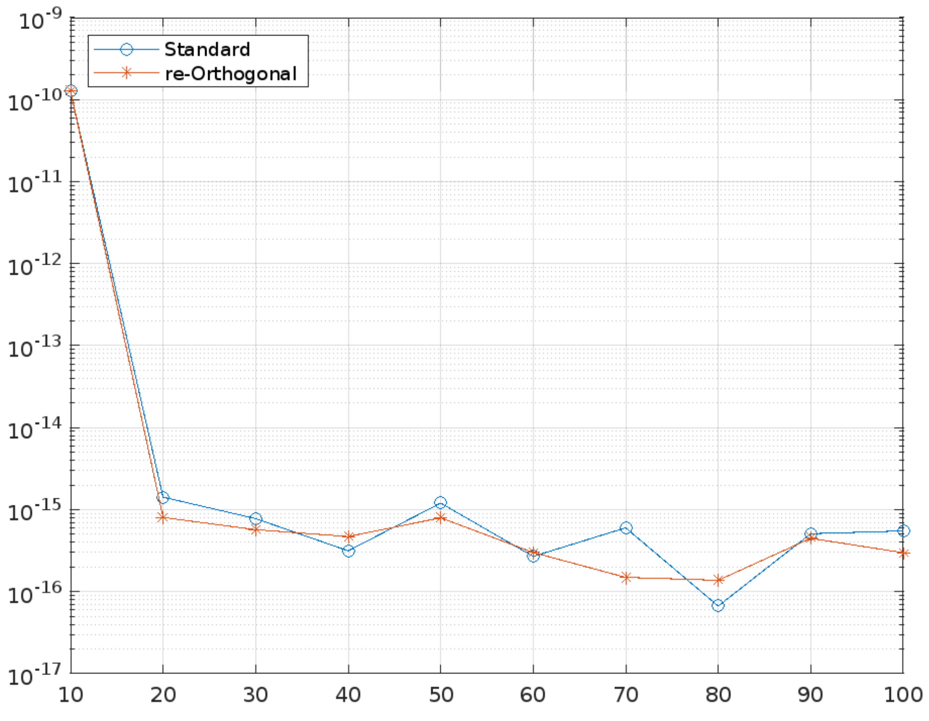


Figure 7: Comparing between Standard and re-Orthogonal Arnoldi

5 Two Large Test Metrices

5.1 HB/1138_bus

- Usage: Power Network Problem
- Describe: The optimal power flow problem is concerned with finding and optimal operating point of a power system, which minimized a certain objective function (e.g. power loss or generation cost) subject to network and physical constraints.
- Size: $1138 * 1138$

- None zeros: 4054

5.1.1 Code

```
in = load('1138bus.mat');
A = in.Problem.A;

idx = 1;
%% start calculate range k
for k = 10:10:100
n = length(A);
V = zeros(n,k); % orthonormal basis
H = zeros(k,k); % upper hessenberg matrix
v = ones(n,1);

V(:,1) = v/norm(v);

% Gram-Schmidt
for j = 1:k
V(:,j+1) = A*V(:,j); % compute w

for i = 1:j
H(i,j) = V(:,i)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-H(i,j)*V(:,i);
end
% normalization
%H(j+1,j) = norm(V(:,j+1));

% -->reorthogonalization process HERE?
% without any if statement
for l = 1:j
mu = V(:,l)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-V(:,l)*mu;
H(l,j) = H(l,j) + mu;
end
H(j+1, j) = norm(V(:,j+1));

V(:,j+1) = V(:,j+1)/H(j+1,j);
% compute residual
Ra(j,idx) = norm(A*V(:,j) - V(:,j+1)*H(:,j)');
Rb(j,idx) = norm(ones(k,k) - V(:,j+1)'*V(:,j+1));
end

H(k+1,:) = [];
V(:, k+1) = [];
Rz = eig(full(H));

% calculate relative errors
muk = max(Rz);
RzV = V*Rz;
nomi = norm(A*RzV - muk*RzV);
deno = (norm(full(A)) + abs(muk)) * norm(RzV);
relerrt6(idx,:) = nomi / deno;
idx = idx + 1;
%% end of the range k
```

```
end
```

Listing 3: hb1138.m

5.1.2 Results

Relative Errors

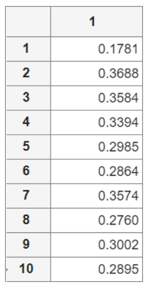
Because we can not calculate the relative errors with:

$$\frac{|\lambda_k - \mu_k^{(j)}|}{|\lambda_k|} \ for \ k = 1, 2, 3...$$

We need to replace relative errors with relative residual norms:

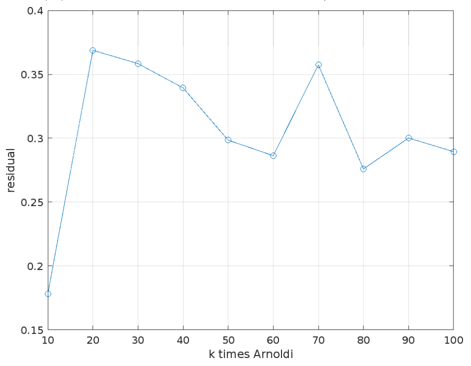
$$\frac{||A\hat{u}_k^{(j)} - u_k^j \hat{u}_k^{(j)}||}{(||A|| + |u_k^j|)||\hat{u}_k^{(j)}||}$$

We need to calculate Ritz values and Ritz vectors.
Figure 8 shows the relative errors.
And 8b shows its residual error figure.



	1
1	0.1781
2	0.3688
3	0.3584
4	0.3394
5	0.2985
6	0.2864
7	0.3574
8	0.2760
9	0.3002
10	0.2895

(a) Relative Errors for HB/1138_bus



(b) Relative Errors Figure for HB/1138_bus

Figure 8: HB/1138_bus

5.2 GD06_Java

- Usage: Directed Graph Problem
- Describe: Find out network flows and calculate each node and link's distance, weight, and wastes.
- Size: $1538 * 1538$
- None zeros: 8032

5.2.1 Code

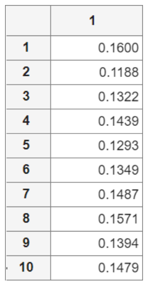
```
in = load('1138bus.mat/GD06Java.mat'); %input 1138but or GD60Java .
    mat file
A = in.Problem.A;

idx = 1;
%% start calculate range k
for k = 10:10:100
```

```
n = length(A);
V = zeros(n,k); % orthonormal basis
H = zeros(k,k); % upper hessenberg matrix
v = ones(n,1);

V(:,1) = v/norm(v);

% Gram-Schmidt
for j = 1:k
V(:,j+1) = A*V(:,j); % compute w

for i = 1:j
H(i,j) = V(:,i)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-H(i,j)*V(:,i);
end
% normalization
%H(j+1,j) = norm(V(:,j+1));

% -->reorthogonalization process HERE?
% without any if statement
for l = 1:j
mu = V(:,l)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-V(:,l)*mu;
H(l,j) = H(l,j) + mu;
end
H(j+1, j) = norm(V(:,j+1));

V(:,j+1) = V(:,j+1)/H(j+1,j);
% compute residual
Ra(j,idx) = norm(A*V(:,j) - V(:,j+1)*H(:,j)');
Rb(j,idx) = norm(ones(k,k) - V(:,j+1)'*V(:,j+1));
end

H(k+1,:) = [];
V(:, k+1) = [];
Rz = eig(full(H));

% calculate relative errors
muk = max(Rz);
RzV = V*Rz;
nomi = norm(A*RzV - muk*RzV);
deno = (norm(full(A)) + abs(muk)) * norm(RzV);
relerrt6(idx,:) = nomi / deno;
idx = idx + 1;
%% end of the range k
end
```

Listing 4: gd06java.m

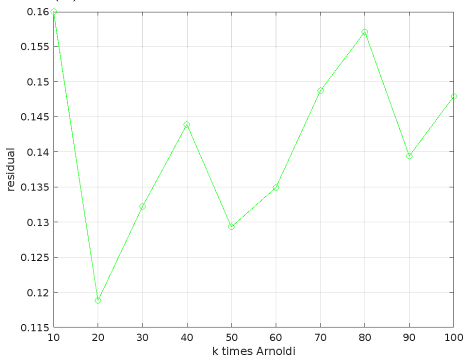
5.2.2 Results

Figure 9 shows the relative errors.
And 9b shows its residual error figure.



	1
1	0.1600
2	0.1188
3	0.1322
4	0.1439
5	0.1293
6	0.1349
7	0.1487
8	0.1571
9	0.1394
10	0.1479

(a) Relative Errors for GD06_Java



(b) Relative Errors Figure for GD06_Java

Figure 9: GD06_Java

6 Shift and invert the Eigenvalues

We are not calculating the original matrix A. Instead, we want to calculate the matrix after shifting and inversing.

$$(A - \tau I)^{-1}x = (\lambda - \tau I)^{-1}x$$

Here, we will set τ to $10, -10, 5$.

6.1 Code

```
load west0479
A = west0479;
n = length(A);
%k = 50;

% shift
tau = 10;
```

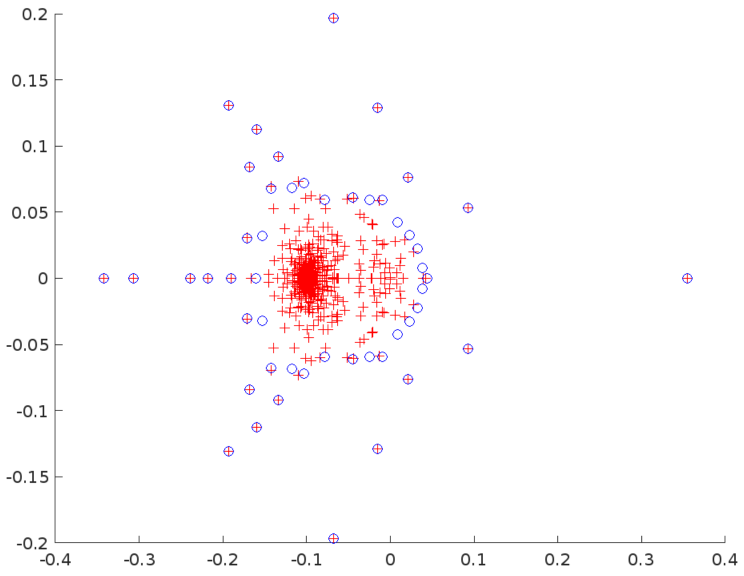
```matlab
A = (A - tau * speye(n));
[L,U,P] = lu(A);
A = inv(U) * inv(L) * inv(P');
%y = L\(P*b);
%x = U\y;

lam = eig(full(A));
figure(5)
hold on
plot(real(lam),imag(lam),'r+');

idx = 1;
%% k
for k = 10:10:50
V = zeros(n,k); % orthonormal basis
H = zeros(k,k); % upper hessenberg matrix
v = ones(n,1);

V(:,1) = v/norm(v);

% Gram-Schmidt
for j = 1:k
V(:,j+1) = A*V(:,j); % compute w

for i = 1:j
H(i,j) = V(:,i)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-H(i,j)*V(:,i);
end
% normalization
H(j+1,j) = norm(V(:,j+1));

% -->reorthogonalization process HERE?
% without any if statement
for l = 1:j
mu = V(:,l)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-V(:,l)*mu;
H(l,j) = H(l,j) + mu;
end
H(j+1, j) = norm(V(:,j+1));

V(:,j+1) = V(:,j+1)/H(j+1,j);
% compute residual
Ra(:,j) = norm(A*V(:,j) - V(:,j+1)*H(:,j)');
Rb(:,j) = norm(speye(k) - V(:,j+1)'*V(:,j+1));
end

H(k+1,:) = [];
Rz = eig(full(H));
% relative error
lumk = max(lam);
muk = max(Rz);
relerror(idx,:) = abs(lumk - muk) / abs(lumk);
idx = idx + 1;

%% end k
end

```

```
Rz = eig(full(H));
plot(real(Rz),imag(Rz),'bo');
hold off
```

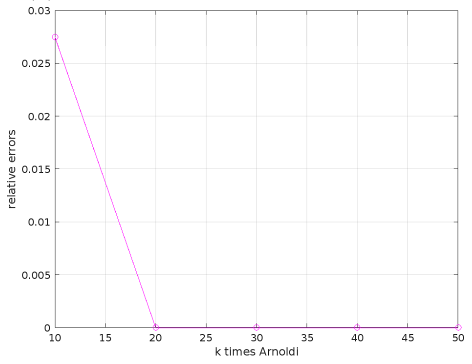
Listing 5: shiftandinverse.m

6.2 $\tau = 10$

In figure 10, we can find out that for larger k, we will have 0 relative errors.



(a) Eigenvalues and Arnoldi Algorithm

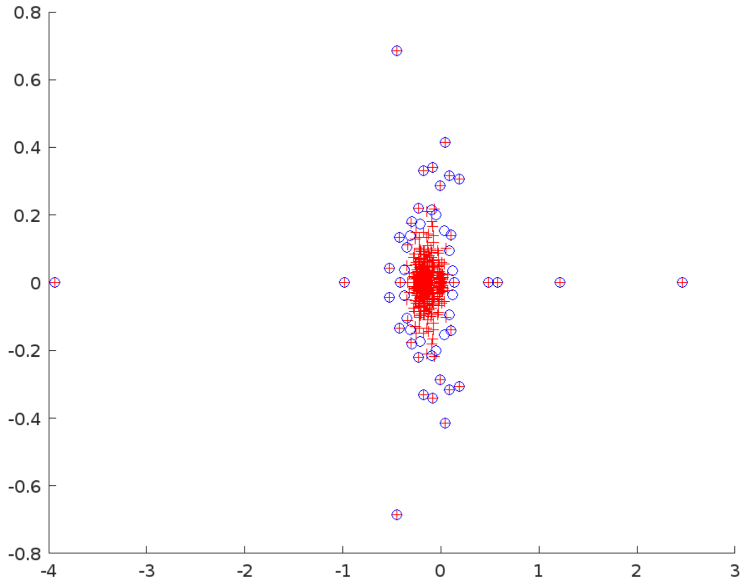


(b) Relative Errors

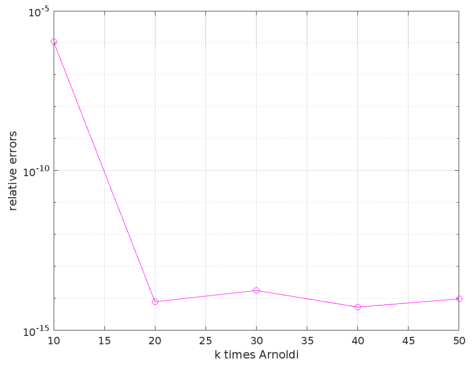
Figure 10: $\tau = 10$

6.3 $\tau = 5$

In figure 11, we can find out that for larger k, we will have relative errors close to 0.



(a) Eigenvalues and Arnoldi Algorithm

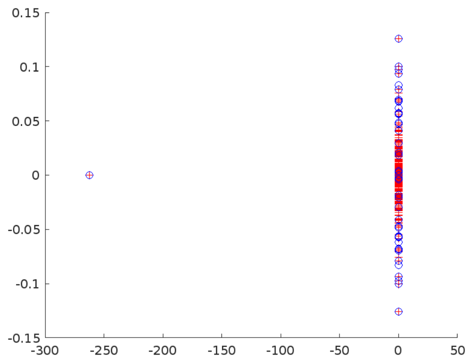


(b) Relative Errors

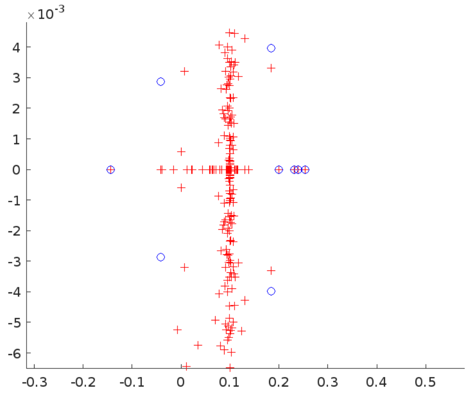
Figure 11: $\tau = 5$

6.4 $\tau = -10$

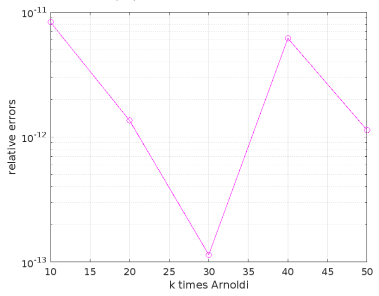
In figure 12, we can find out that for larger k, we will have relative errors close to 0. And the the most of eigenvalues are near at $X = 0$.



(a) Eigenvalues and Arnoldi Algorithm



(b) Closer look



(c) Relative Errors

Figure 12: $\tau = -10$

7 Shift and invert the Eigenvalues:Large Matrix

7.1 Code

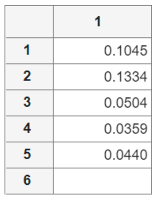
```
in = load('1138bus.mat/GD06Java.mat'); %input 1138but or GD60Java .
    mat file
A = in.Problem.A;
n = length(A);

% shift
tau = -10;
A = (A - tau * speye(n));
[L,U,P] = lu(A);
A = inv(U) * inv(L) * inv(P');

idx = 1;
%% start calculate range k
for k = 10:10:50
V = zeros(n,k); % orthonormal basis
H = zeros(k,k); % upper hessenberg matrix
v = ones(n,1);

V(:,1) = v/norm(v);

% Gram-Schmidt
for j = 1:k
V(:,j+1) = A*V(:,j); % compute w

for i = 1:j
H(i,j) = V(:,i)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-H(i,j)*V(:,i);
end
% normalization
%H(j+1,j) = norm(V(:,j+1));

% -->reorthogonalization process HERE?
% without any if statement
for l = 1:j
mu = V(:,l)'*V(:,j+1);
V(:,j+1) = V(:,j+1)-V(:,l)*mu;
H(l,j) = H(l,j) + mu;
end
H(j+1, j) = norm(V(:,j+1));

V(:,j+1) = V(:,j+1)/H(j+1,j);
% compute residual
Ra(j,idx) = norm(A*V(:,j) - V(:,j+1)*H(:,j)');
Rb(j,idx) = norm(ones(k,k) - V(:,j+1)'*V(:,j+1));
end

H(k+1,:) = [];
V(:, k+1) = [];
Rz = eig(full(H));

% calculate relative errors
muk = max(Rz);
```

```
RzV = V*Rz;
nomi = norm(A*RzV - muk*RzV);
deno = (norm(full(A)) + abs(muk)) * norm(RzV);
relerrt6(idx,:) = nomi / deno;
idx = idx + 1;
%% end of the range k
end

figure(81)
X = 10:10:50;
semilogy(X, relerrt6, 'm-o')
xlabel('k times Arnoldi')
ylabel('relative errors')
grid on
```

Listing 6: task8.m

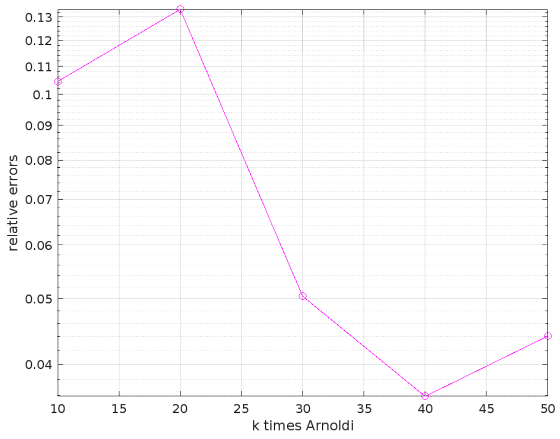
7.2 Results for HB_1138

7.2.1 $\tau = -10$



	1
1	0.1045
2	0.1334
3	0.0504
4	0.0359
5	0.0440
6	

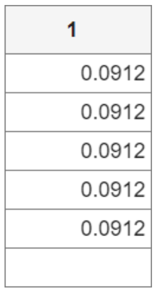
(a) Convergence



(b) Convergence Figure

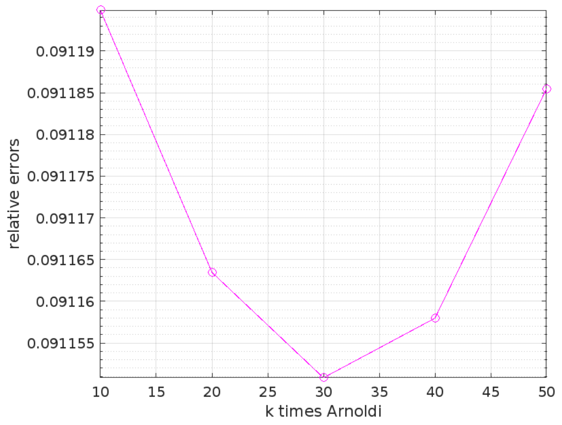
Figure 13: Convergence for HB_1138

7.2.2 $\tau = 10$



1
0.0912
0.0912
0.0912
0.0912
0.0912

(a) Convergence

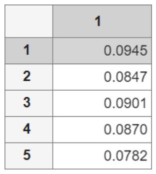


(b) Convergence Figure

Figure 14: Convergence for HB_1138

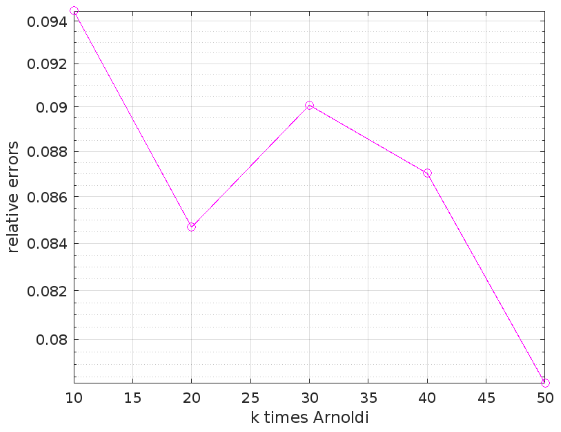
7.3 Results for GD06_JAVA

7.3.1 $\tau = -10$



	1
1	0.0945
2	0.0847
3	0.0901
4	0.0870
5	0.0782

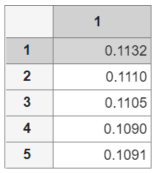
(a) Convergence



(b) Convergence Figure

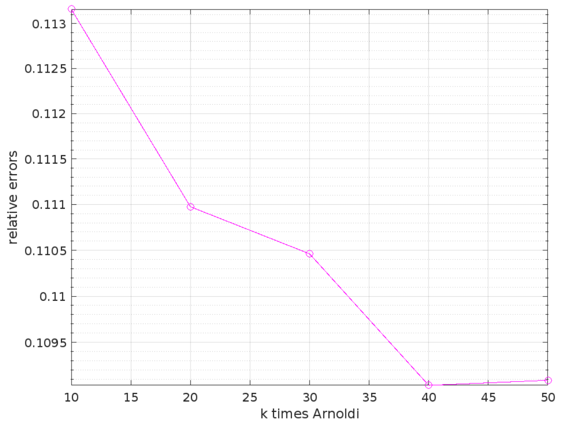
Figure 15: Convergence for GD06_Java

7.3.2 $\tau = 10$



	1
1	0.1132
2	0.1110
3	0.1105
4	0.1090
5	0.1091

(a) Convergence



(b) Convergence Figure

Figure 16: Convergence for GD06_Java