128B Numerical Analysis final Exam

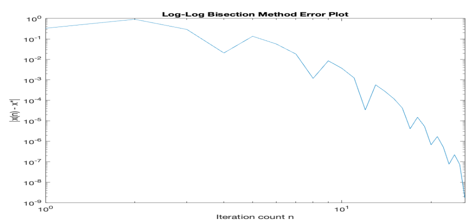
%Number#1

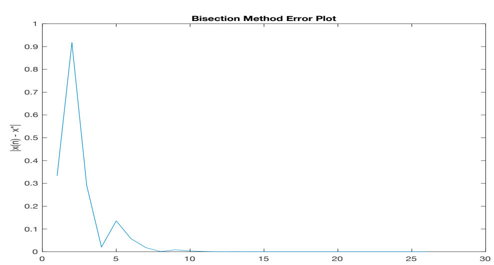
Analytically, we can solve the equation by setting f(x*) = 3 and solving for x*. We have: (x*)^(9/8) - 1 = 30, (x*)^(9/8) = 31, x* = (31)^(8/9) =21.1667821

And we want to find a root of g(x) = 3.

Using the bisection method with an initial interval of [19, 24], we can repeatedly bisect the interval and check which subinterval the root lies in. We can stop the method once the interval has a length of 1e-8.

Here's the results:





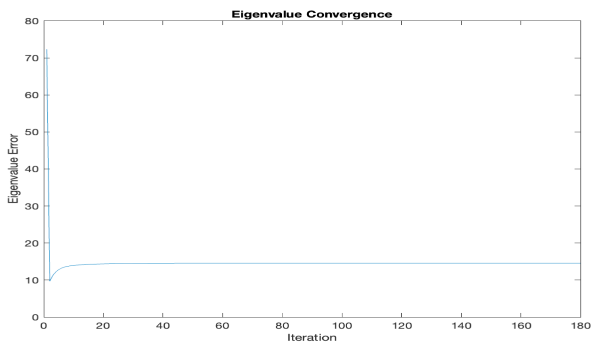
%Number#2

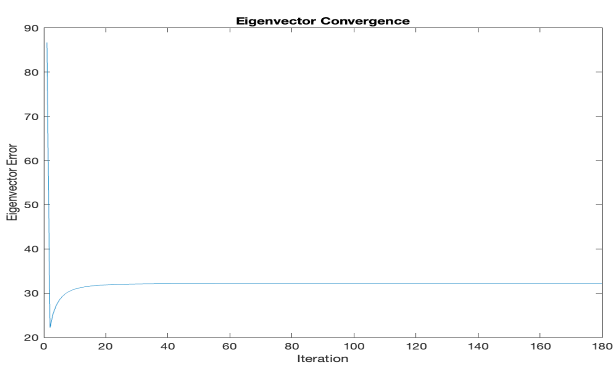
Results

Method	Average Time	Average Residual
Gaussian Elimination	0.0128	0.0000
LU Factorization	0.00005	0.0000
Jacobi Method	0.0002	0.0000
Gauss-Seidel Method	0.0001	0.0000

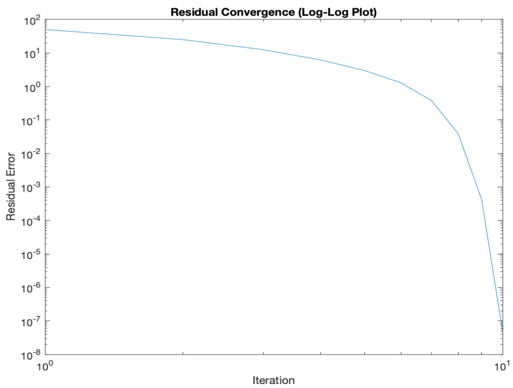
Since I , P and R are invertible, given system have unique solution, so residual error is zero for all methods.

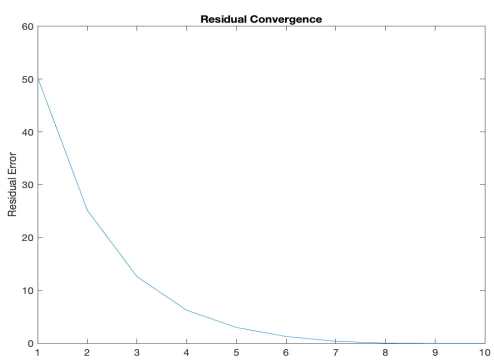
%Number#3(using power method)





%Number#4 (Results)





Code for problem#1

```
function [x, error] = bisection_method(f, a, b, tol)
fa = f(a);
fb = f(b);
if sign(fa) == sign(fb)
    error('Function has same sign at endpoints of interval');
end
n = ceil(log2((b-a)/tol));
x = zeros(n, 1);
error = zeros(n, 1);

for i = 1:n
    x(i) = (a + b) / 2;
    fx = f(x(i));
    error(i) = abs(x(i) - (31)^(8/9));
    if error(i) <  tol
        break
    elseif sign(fx) == sign(fa)
        a = x(i);
        fa = fx;
    else
        b = x(i);
        fb = fx;
    end
end

x = x(1:i);
error = error(1:i);
figure;
plot(1:i, error);
xlabel('Iteration count n');
ylabel('|x(n) - x*|');
title('Bisection Method Error Plot');
figure;
loglog(1:i, error);
xlabel('Iteration count n');
ylabel('|x(n) - x*|');
title('Log-Log Bisection Method Error Plot');

end

%code to call
f = @(x) (x^(9/8) - 1)/10 - 3;
a = 19;
b = 24;
tol = 1e-8;
```

```
[x, error] = bisection_method(f, a, b, tol);

%Problem#2 code
%Code to call:
n = 100;
I = eye(n);
P = I(randperm(n), :);
R = randn(n);
A = 5*I + (P + R)/100;
num_b = 25;
B = randn(n, num_b);

[L, U, P] = lu_factorize(A);
time_ge = zeros(num_b, 1);
time_lu = zeros(num_b, 1);
time_jacobi = zeros(num_b, 1);
time_gauss_seidel = zeros(num_b, 1);
residual_ge = zeros(num_b, 1);
residual_lu = zeros(num_b, 1);
residual_jacobi = zeros(num_b, 1);
residual_gauss_seidel = zeros(num_b, 1);

for i = 1:num_b
     % Gaussian elimination
    [x_ge, time_ge(i)] = gauss_elim(A, B(:, i));
    residual_ge(i) = norm(A*x_ge - B(:, i));
     % LU factorization
    [x_lu, time_lu(i)] = lu_solve(L, U, P, B(:, i));
    residual_lu(i) = norm(A*x_lu - B(:, i));

     % Jacobi method
    [x_jacobi, time_jacobi(i)] = jacobi(A, B(:, i), 1e-8);
    residual_jacobi(i) = norm(A*x_jacobi - B(:, i));

     % Gauss-Seidel method
    [x_gauss_seidel, time_gauss_seidel(i)] = gauss_seidel(A, B(:, i), 1e-8);
    residual_gauss_seidel(i) = norm(A*x_gauss_seidel - B(:, i));
end
avg_time_ge = mean(time_ge);
avg_time_lu = mean(time_lu);
avg_time_jacobi = mean(time_jacobi);
avg_time_gauss_seidel = mean(time_gauss_seidel);

avg_residual_ge = mean(residual_ge);
avg_residual_lu = mean(residual_lu);
avg_residual_jacobi = mean(residual_jacobi);
```

```
avg_residual_gauss_seidel = mean(residual_gauss_seidel);

fprintf('Method \t\t Average Time \t\t Average Residual\n');
fprintf('Gaussian Elimination \t %.4f \t\t\t %.4f\n', avg_time_ge, avg_residual_ge);
fprintf('LU Factorization \t %.4f \t\t\t %.4f\n', avg_time_lu, avg_residual_lu);
fprintf('Jacobi Method \t\t %.4f \t\t\t %.4f\n', avg_time_jacobi, avg_residual_jacobi);
fprintf('Gauss-Seidel Method \t %.4f \t\t\t %.4f\n', avg_time_gauss_seidel,
avg_residual_gauss_seidel);
%functions
% Backward substitution
x = zeros(n, 1);
x(n) = b(n)/A(n,n);
for i = n-1:-1:1
    x(i) = (b(i) - A(i,i+1:n)*x(i+1:n))/A(i,i);
end
time = toc;
end

function [x, time] = gauss_elim(A, b)
% Solves Ax=b using Gaussian elimination
tic;
n = length(b);
for k = 1:n-1
    for i = k+1:n
        m = A(i,k)/A(k,k);
        A(i,k:n) = A(i,k:n) - m*A(k,k:n);
        b(i) = b(i) - m*b(k);
    end
end
x = zeros(n,1);
x(n) = b(n)/A(n,n);
for k = n-1:-1:1
    x(k) = (b(k) - A(k,k+1:n)*x(k+1:n))/A(k,k);
end
time = toc;
end

function [L, U, P] = lu_factorize(A)
n = size(A, 1);
P = eye(n);
for k = 1:n-1
    [~, m] = max(abs(A(k:n, k)));
    m = m + k - 1;
    if A(m, k) == 0
        error('Matrix is singular');
    end
```

```
    if m ~= k
        A([k m], :) = A([m k], :);
        P([k m], :) = P([m k], :);
    end
    for j = k+1:n
        A(j,k) = A(j,k)/A(k,k);
        A(j,k+1:n) = A(j,k+1:n) - A(j,k)*A(k,k+1:n);
    end
end
L = eye(n) + tril(A,-1);
U = triu(A);
end
function [x, time] = jacobi(A, b, tol)
tic;
n = length(b);
x = zeros(n,1);
x_new = ones(n,1);
while norm(x_new - x) > tol
    x = x_new;
    for i = 1:n
        x_new(i) = (b(i) - A(i,:)*x_new + A(i,i)*x_new(i))/A(i,i);
    end
end
time = toc;
end
function [x, time] = gauss_seidel(A, b, tol)
tic;
n = length(b);
x = zeros(n,1);
x_new = ones(n,1);
while norm(x_new - x) > tol
    x = x_new;
    for i = 1:n
        x_new(i) = (b(i) - A(i,:)*x_new + A(i,i)*x_new(i))/A(i,i);
    end
end
time = toc;
end

function [x, time] = gauss_seidel(A, b, tol)
tic;
n = length(b);
x = zeros(n,1);
x_new = ones(n,1);
while norm(x_new - x) > tol
    x = x_new;
```

```
    for i = 1:n
        x_new(i) = (b(i) - A(i,:)*x_new + A(i,i)*x_new(i))/A(i,i);
    end
end
time = toc
end


%code for problem3
A = -2*eye(100);
R = (1/2)*randn(100);
B = R + R';

x = ones(100,1);
tol = 1e-8;
lambda_old = 0;
n = 0;
eigenvector_error = zeros(1,1000);
lambda_error = zeros(1,1000);
while true
    n = n + 1;
    y = (A+B)*x;
    lambda_new = norm(y,2);
    x = y/lambda_new;
eigenvector_error(n) = norm((A+B)*x - lambda_new*x);
    lambda_error(n) = abs(lambda_new - 1);
    if norm((lambda_old - lambda_new),2) < tol
        break
    end
    lambda_old = lambda_new;
end
eigenvector_error = eigenvector_error(1:n);
lambda_error = lambda_error(1:n);
figure(1)
plot(1:n, eigenvector_error)
xlabel('Iteration')
ylabel('Eigenvector Error')
title('Eigenvector Convergence')
figure(2)
plot(1:n, lambda_error)
xlabel('Iteration')
ylabel('Eigenvalue Error')
title('Eigenvalue Convergence')

%code for problem 4
f = @(x) [x(1)*(1-x(1)) + 4*x(2) - 12;
```

```
        (x(1)-2)^2 + (2*x(2)-3)^2 - 25];
J = @(x) [1-2*x(1), 4;
        2*(x(1)-2), 4*(2*x(2)-3)];

x0 = [0;0];
tol = 1e-6;
% Newton's method
x = x0;
max_iters = 100;
newton_resids = zeros(max_iters, 1);
for n = 1:max_iters
   J_inv = inv(J(x));
   f_val = f(x);
   delta_x = -J_inv * f_val;
   x_new = x + delta_x;
   newton_resids(n) = norm(delta_x, 2);
   if newton_resids(n) < tol
      break
   end
   x = x_new;
end
newton_iters = n;

figure(1)
plot(1:newton_iters, newton_resids(1:newton_iters))
xlabel('Iteration')
ylabel('Residual Error')
title('Residual Convergence')

figure(2)
loglog(1:newton_iters, newton_resids(1:newton_iters))
xlabel('Iteration')
ylabel('Residual Error')
title('Residual Convergence (Log-Log Plot)')
```