HW1-Numerical Analysis

2024 Fall Section 007

Student: Ashley Wen, xw3421@nyu.edu
Git: github.com/Ashstar9527/AW-Numerical-Analysis
Lecturer: Prof. Jason Kaye

Problem 1 and other problems may include extra coding files. For reference, the link of Git Repository with all the coding files has been attached above.

Problem 1

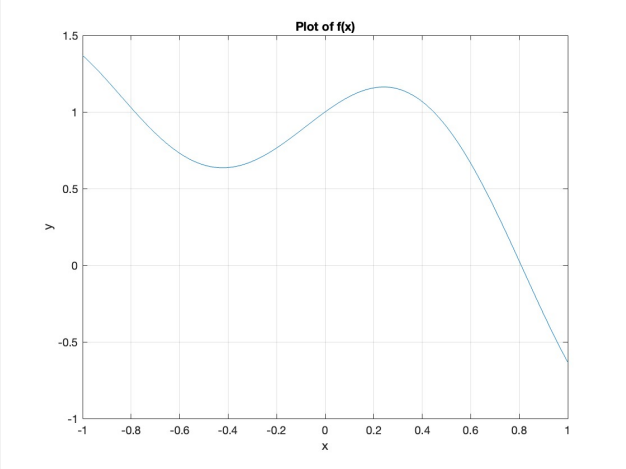
Let the function defined as:

$$f(x) = x^3 - 2x + \sin(\pi x) + \exp\left(-x^2\right).$$

The root can be found as

$$x^* \approx 0.80781$$

The plot of f(x):



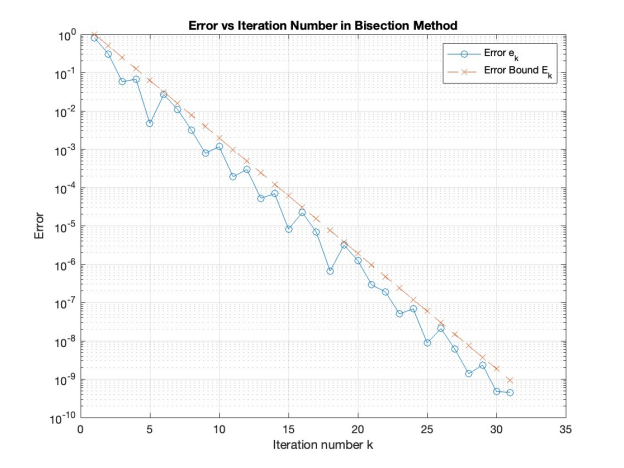
Codes are as followed:

```
% Define the function
f = @(x) x.^3 - 2 * x + sin(pi*x) + exp(-x.^2);

% Create a range of x values from -1 to 1
x = linspace(-1, 1, 2000);

% Evaluate the function over the range
y = f(x);

% Plot the function
figure;
plot(x, y);
xlabel('x');
ylabel('y');
title('Plot of f(x)');
grid on;
```

```
% Use fzero to find the root
root = fzero(f, [-1,1]);
x_exact = root;

% Display the root
disp(['The root is: ', num2str(root)]);
```

(b): Log-lin plot

We observe that no errors e_k exceed the corresponding error bound E_k. In the plot, the blue line represents the log of the actual error of each iteration, and the red line represents the log of the corresponding error bound.




```
% Setting
a = -1;
b = 1;
itermax = 30;

% Function call
[x_bisect, xiter] = bisection(f, a, b, itermax);

% Calculate the error
errors = abs(xiter - x_exact);

% Plot the error on a semilogarithmic scale
semilogy(1:itermax+1, errors, 'o-', 'DisplayName', 'Error e_k');
hold on;

% Calculate and plot the error bound
error_bound = (b - a) ./ (2 .^ (1:itermax+1));
semilogy(1:itermax+1, error_bound, 'x--', 'DisplayName', 'Error Bound E_k');

% Add labels, title, and legend
xlabel('Iteration number k');
ylabel('Error');
title('Error vs Iteration Number in Bisection Method');
legend;
```

```
grid on;
hold off;
```

(c): Explain the curve
In Bisection method, after k iterations in interval $[a_k, b_k]$, we have:

$$\mathrm{E}_k = \frac{b_k - a_k}{2}$$

Therefore, within original interval [a, b], we have:

$$\mathrm{E}_k = \frac{b-a}{2^k}$$

$$\log(E_k) = \log(\text{b-a}) - \text{k } \log 2$$

Considering $e_k < E_k = \frac{b_k - a_k}{2}$

$$log(e_k) < log(E_k) = \log(\text{b-a}) - \text{k } \log 2$$

Where $k{\cdot}log2$ is a constant, therefore the relationship between e_k and number of iterations is close to a straight line.

The code for Problem 2(b) has been uploaded together with 2(a). A copy will be provided below:

Problem 2

(a) The three files have been uploaded.
(b) Results: all the errors are smaller than the tolerance tol = 10^{-10}. The errors in 15 digits are as below:

Bisection: 0.000000000021557
Newtons: 0.000000000000000
Secant: 0.000000000000000

The codes are as follows:

```
% Function
f = @(x) cos(exp(x));
itermax = 40;

% Find root
x_exact = fzero(f, [-1, 1]);

% Bisection setting
a = -1;
b = 1;
tol = 1e-10;

% Bisection Method
[x_bisect, xiter_bi, niter_bi] = bisection_tol(f, a, b, itermax, tol);
error_bisect = x_bisect - x_exact;
disp(['1. Error of Bisection Method: ', num2str(error_bisect)]);
result = (error_bisect < tol);
```

```
disp(['     Is error of Bisection smaller than the tolerance? ', num2str(result)]);

% Newton's setting
fp = @(x) -exp(x) * sin(exp(x));
x0 = -0.2;

% Newton's Method
[x_newt,xiter_newt, niter_newt] = newton_tol(f,fp,x0,itermax, tol);
error_newt = x_newt - x_exact;
disp(['2. Error of Newtons Method: ', num2str(error_newt)]);
result = (error_newt < tol);
disp(['     Is error of Newtons smaller than the tolerance? ', num2str(result)]);

% Secant setting
x0 = 0.1;
x1 = 0.2;

% Secant Method
[x_sec,xiter,niter_sec] = secant_tol(f,x0,x1,itermax,tol);
error_sec = x_sec - x_exact;
disp(['3. Error of Secant Method: ', num2str(error_sec)]);
result = (error_sec < tol);
disp(['     Is error of Secant smaller than the tolerance? ', num2str(result)]);
fprintf('%.15f\n', error_bisect);
fprintf('%.15f\n', error_newt);
fprintf('%.15f\n', error_sec);
```

(c)

- Bisection: For Bisection method, if the method converges very slowly, for example when the root is very close to the boundary, the step between the successive iterates can be very small even when it is in fact far from the root.
  To address the issue, we could take

$$\text{tol}_k = \frac{b_k - a_k}{2}$$

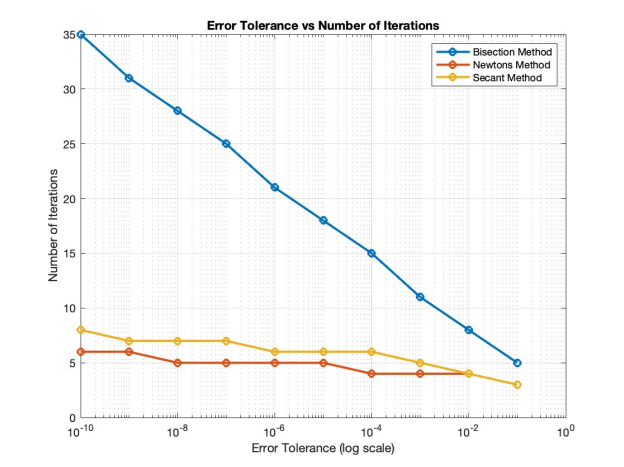
- Newton's: Poor initial guess can lead to slow convergence, and the rest is the same as the problem in bisection method. *To address the issue, we could take*

$$|x_k - x_{k-1}| < tol \text{ and } |f(x)| < tol$$

- Secant's: Other than the step problem, there could be multiple roots nearby, which leads to slow convergence. *To address the issue, we could take*

$$|x_k - x_{k-1}| < tol \text{ and } |f(x)| < tol$$

In general, all the three strategies are not completely robust when taking $x_k - x_{k-1}$ as the tolerance.
(d) Observation: When using Bisection method, there is an almost linear relationship between tolerance and number of iteration; in contrast, when using Newton's and Secant method, the number of iteration it takes to reach the optimal barely changes, displayed as a nearly flat line on the lin-log plot.



The codes used to generate this plot are as followed:

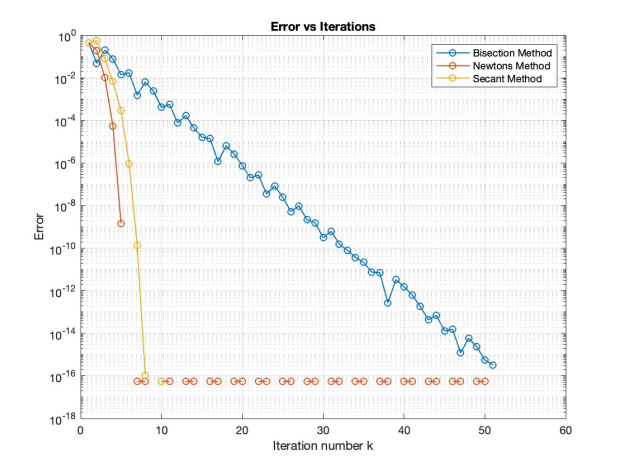
```
% Function
f = @(x) cos(exp(x));
itermax = 1000;
x_exact = fzero(f, [-1, 1]);

% Bisection: setting
a = -1;
b = 1;

% Bisection: Define the tolerances to test, from 10^-1 to 10^-10
tolerances = 10.^-(1:10);

% Bisection: Initialize vectors to store results
niter_bisect = zeros(length(tolerances),1);

% Bisection: Loop over each tolerance
for i = 1:length(tolerances)
    tol = tolerances(i);

    % Run bisection method
    [x_bisect, xiter, niter_bisect(i)] = bisection_tol(f, a, b, itermax, tol);
end

% Newton's: setting
fp = @(x) -exp(x) * sin(exp(x));
x0 = 0;
niter_newt = zeros(length(tolerances),1);

% Newton's: Loop
for i = 1:length(tolerances)
    tol = tolerances(i);
    [x_newt, xiter, niter_newt(i)] = newton_tol(f, fp, x0, itermax, tol);
end

% Secant: setting
x0 = 0;
```

```
x1 = 1;
niter_sec = zeros(length(tolerances),1);

% Secant: Loop
for i = 1:length(tolerances)
    tol = tolerances(i);
    [x_sec, xiter, niter_sec(i)] = secant_tol(f, x0, x1, itermax, tol);
end

% Plot
figure;
semilogx(tolerances, niter_bisect, '-o', 'LineWidth', 2);
hold on;
semilogx(tolerances, niter_newt, '-o', 'LineWidth', 2);
hold on;
semilogx(tolerances, niter_sec, '-o', 'LineWidth', 2);
hold off;

xlabel('Error Tolerance (log scale)');
ylabel('Number of Iterations');
title('Error Tolerance vs Number of Iterations');
grid on;
legend('Bisection Method', 'Newtons Method', 'Secant Method');
```

Problem 3 includes two graphs and the codes to generate them. The copies of the codes are attached below.

Problem 3

(a) Observation: Newton's method and Secant's method converge much faster than Bisection method. Bisection method has convergence order of 1, which leads to an almost linear relationship between log of errors and number of iteration; the other two have higher order.



Codes used are as followed:

```
% Function
f = @(x) cos(exp(x));
itermax = 50;
```

```
% Find root
x_exact = fzero(f, [-1, 1]);

% Bisection: setting
a = -1;
b = 1;

% Bisection: Find root and calculate error
[x_bisect, xiter_bisect] = bisection(f, a, b, itermax);
errors_bisect = abs(xiter_bisect - x_exact);
errors_bisect = [errors_bisect; zeros(51 - length(errors_bisect), 1)];

% Newton's: setting
fp = @(x) -exp(x) * sin(exp(x));
x0 = 0;
[x_newt, xiter_newt] = newton(f, fp, x0, itermax);
errors_newt = abs(xiter_newt - x_exact);
errors_newt = [errors_newt; zeros(51 - length(errors_newt), 1)];

% Secant: setting
x0 = 0;
x1 = 1;

% Secant: Loop
[x_sec, xiter_sec] = secant(f, x0, x1, itermax);
errors_sec = abs(xiter_sec- x_exact);
errors_sec = [errors_sec; zeros(51 - length(errors_sec), 1)];

% Figure
figure;
semilogy(1:itermax+1, errors_bisect, 'o-','LineWidth', 1);
hold on;
semilogy(1:itermax+1, errors_newt, 'o-', 'LineWidth', 1);
hold on;
semilogy(1:itermax+1, errors_sec, 'o-', 'LineWidth', 1);
hold off

% Add labels, title, and legend
xlabel('Iteration number k');
ylabel('Error');
title('Error vs Iterations');
legend('Bisection Method', 'Newtons Method', 'Secant Method');

grid on;
```

(b)
From the definition, we know that

$$\lim_{x \to +} \frac{\epsilon_k}{{\epsilon_{k-1}}^p} = C$$

For sufficiently large k,

$$C{\epsilon_{k-1}}^p \approx \epsilon_k$$

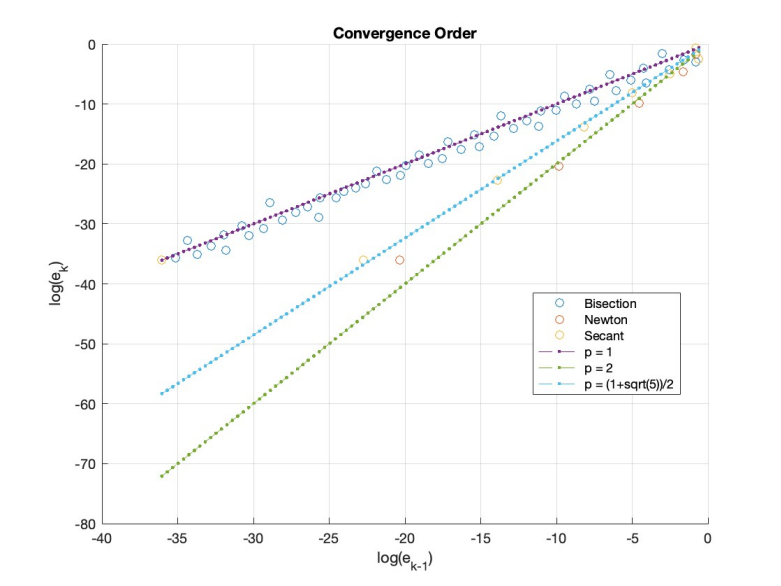
$$Therefore,$$

$$logC + p \cdot log\epsilon_{k-1} \approx log\epsilon_k$$

$$log\epsilon_k = c + p \cdot log\epsilon_{\epsilon_{k-1}}$$

(c)

Observations: it's easy to identify Bisection and Newton's method, since most of the points are close to the related lines. However, it's difficult to locate the convergence order of secant method as some of the points are scattered and far from the line.



Codes used are as followed:

```
% Function
f = @(x) cos(exp(x));
itermax = 50;

% Find root
x_exact = fzero(f, [-1, 1]);

% Bisection: setting
a = -1;
b = 1;

% Bisection: Find root and calculate error
[x_bisect, xiter_bisect] = bisection(f, a, b, itermax);
errors_bisect = abs(xiter_bisect - x_exact);

% Newton's: setting
fp = @(x) -exp(x) * sin(exp(x));
```

```
x0 = 0;
[x_newt, xiter_newt] = newton(f, fp, x0, itermax);
errors_newt = abs(xiter_newt - x_exact);

% Secant: setting
x0 = 0;
x1 = 1;

% Secant: Loop
[x_sec, xiter_sec] = secant(f, x0, x1, itermax);
errors_sec = abs(xiter_sec- x_exact);

% Adding small constants to avoid log(0)
c = eps;
log_bisect = log(max(errors_bisect, c));
log_newt = log(max(errors_newt, c));
log_sec = log(max(errors_sec, c));

% Figure
figure;
scatter(log_bisect(1:end-1), log_bisect(2:end));
hold on;
scatter(log_newt(1:end-1), log_newt(2:end));
hold on;
scatter(log_sec(1:end-1), log_sec(2:end));

% Add the lines y = px
x_vals = linspace(min([log_bisect; log_newt; log_sec]), ...
    max([log_bisect; log_newt; log_sec]), 100);

plot(x_vals, x_vals, '.--');                    % Bisection
plot(x_vals, 2*x_vals, '.--');                  % Newton
plot(x_vals, ((1+sqrt(5))/2)*x_vals, '.--');    % Secant


% Customize plot
xlabel('log(e_{k-1})');
ylabel('log(e_k)');
legend('Bisection', 'Newton', 'Secant', 'p = 1', 'p = 2', 'p = (1+sqrt(5))/2', 'Location', 'best'
title('Convergence Order');
grid on;
```

Problem 4

(b)
Using the codes uploaded, we can get it takes (2+4)=6 Newton's iterations or 38 Bisection method iterations.

Problem 5

I'm sorry but I really don't want to type all the variables and functions here again. please see the hand-written solutions on next page.

5. (a) $x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k)$

$$= \frac{x_k f(x_k) - x_k f(x_{k-1}) - x_k f(x_k) + x_{k-1} f(x_k)}{f(x_k) - f(x_{k-1})}$$

$$= \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$$

(b) $\varphi(x_k, x_{k-1}) = \frac{x_{k+1} - x^*}{(x_k - x^*)(x_{k-1} - x^*)}$ from 5.(a)

$$= \frac{\frac{x_k f(x_{k-1}) - x_{k-1} f(x_k)}{f(x_{k-1}) - f(x_k)} - x^*}{(x_k - x^*)(x_{k-1} - x^*)}$$

$$= \frac{1}{f(x_{k-1}) - f(x_k)} \times \frac{x_k f(x_{k-1}) - x_{k-1} f(x_k) - x^* f(x_{k-1}) + x^* f(x_k)}{(x_k - x^*)(x_{k-1} - x^*)}$$

$$= \frac{1}{f(x_{k-1}) - f(x_k)} \times \frac{f(x_{k-1})(x_k - x^*) - f(x_k)(x_{k-1} - x^*)}{(x_k - x^*)(x_{k-1} - x^*)}$$

$$= \frac{1}{f(x_{k-1}) - f(x_k)} \times \left(\frac{f(x_{k-1})}{x_{k-1} - x^*} - \frac{f(x_k)}{x_k - x^*} \right)$$

$\because f(x^*) = 0$

$$\therefore \varphi(x_k, x_{k-1}) = \frac{1}{f(x_{k-1}) - f(x_k)} \times \left(\frac{f(x_{k-1}) - f(x^*)}{x_{k-1} - x^*} - \frac{f(x_k) - f(x^*)}{x_k - x^*} \right)$$

$$\therefore \lim_{x_k \to x^*} \varphi(x_k, x_{k-1}) = \lim_{x_k \to x^*} \frac{1}{f(x_{k-1}) - f(x_k)} \times \left(\frac{f(x_{k-1}) - f(x^*)}{x_{k-1} - x^*} - \frac{f(x_k) - f(x^*)}{x_k - x^*} \right)$$

$$= \lim_{x_k \to x^*} \frac{1}{f(x_k) - f(x_{k-1})} \times \left(\frac{f(x_k) - f(x^*)}{x_k - x^*} - \frac{f(x_{k-1}) - f(x^*)}{x_{k-1} - x^*} \right)$$

$$= \frac{f'(x^*) - \frac{f(x_{k-1}) - f(x^*)}{x_{k-1} - x^*}}{f(x^*) - f(x_{k-1})}$$

(c) $\lim_{k \to \infty} \varphi(x_k, x_{k-1}) = \frac{f'(x^*)(x_{k-1} - x^*) - \left(f(x_{k-1}) - f(x^*)\right)}{\left(f(x^*) - f(x_{k-1})\right)(x_{k-1} - x^*)}$

$$= \frac{f'(x^*)(x^* - x_{k-1}) - (\underbrace{f(x^*)}_{=0} + f(x_{k-1}))}{(x^* - x_{k-1})(f(x^*) - f(x_{k-1}))}$$

$$\lim_{x_{k-1} \to x^*} \lim_{x_k \to x^*} \varphi(x_k, x_{k-1}) = \frac{-f'(x^*) + f'(x_{k-1})}{-f'(x_{k-1})(x^* - x_{k-1}) + \left[f(x^*) - f(x_{k-1})\right] \cdot (-1)}$$

$$= \frac{-f'(x^*) + f'(x_{k-1})}{-f'(x_{k-1})(x^* - x_{k-1}) - \underbrace{f(x^*)}_{=0} + f(x_{k-1})}$$

$$= \frac{f''(x_{k-1})}{-f''(x_{k-1})(\underbrace{x^* - x_{k-1}}_{\to 0}) + f''(x_{k-1}) - f''(x_{k-1})}$$

$$= \boxed{\frac{f''(x_{k-1})}{2f'(x_{k-1})}}$$

(d)

$$\frac{x_{k+1}-x^*}{(x_k-x^*)^p} = \frac{(x_{k-1}-x^*)}{(x_k-x^*)^{p-1}} \cdot \frac{(x_{k+1}-x^*)\cdot}{(x_k-x^*)\cdot(x_{k-1}-x^*)}$$

$$= \frac{(x_{k-1}-x^*)}{(x_k-x^*)^{p-1}} \cdot \varphi(x_k, x_{k-1})$$

$x_k - x^*$

$$\lim_{k\to\infty} \frac{|x_{k+1}-x^*|}{|x_k-x^*|^p} = \lim_{k\to\infty} \left| \varphi(x_k, x_{k-1}) \cdot \frac{x_{k-1}-x^*}{(x_k-x^*)^{p-1}} \right|$$

$\because k\to\infty, \quad \therefore x_k \to x^*, \; x_{k-1} \to x^*,$

$$\therefore \varphi(x_k, x_{k-1}) \to \frac{f''(x_{k-1})}{2f'(x_{k-1})} = C.$$

$$\therefore \lim_{k\to\infty} \frac{|x_{k+1}-x^*|}{(x_k-x^*)} = \lim_{k\to\infty} \frac{C(x_{k-1}-x^*)}{(A(x_{k-1}-x^*)^p)^{p-1}}$$

$$= \lim_{k\to\infty} \frac{C}{A^{p-1}(x_{k-1}-x^*)^{p^2-p-1}} = A$$

$\therefore p^2-p-1=0$ must hold

$$p-1-\frac{1}{p}=0 \Rightarrow p=\frac{1+\sqrt{5}}{2}$$

$$\Rightarrow \frac{C}{A^{p-1}} = A$$

$$C = A^p$$

$\because p = 1+\frac{1}{p}$

$$C = A^{\frac{p+1}{p}}$$

$$\Rightarrow A = C^{\frac{p}{p+1}}$$

$$A = \left(\frac{f''(x^*)}{2f'(x^*)}\right)^{\frac{p}{p+1}}$$