

Problem 1

Calculate the centered finite difference approximation of $f''(x)$ for the following functions on the interval $x \in [-1, 1]$. Show the numerical and the analytical results in the same plot using a grid size $h = 0.02$. Show the log-log plots of the errors versus grid sizes. Use at least 4 grid sizes for the error plot

As requested, the initial values are $h = 0.02$, an interval of $[-1, 1]$ with interval gaps of h . Finally `grid_size = 4`

```
1 h = 0.02; % sets h
2 number_of_grids = 4; % grid sizes for error plot
3 x_interval = -1+h : h : 1-h; % sets interval [-1 to 1] for xi with h gap
```

Initialization

a) $f(x) = e^x \sin(\frac{\pi x}{2})$

Described in the function, `problem_1.f(x)`:

```
1 function result = problem_1.f(x)
2     result = exp(x)*sin((pi*x)/2); % Returns f(x) for Problem A
3 end
```

`problem_1.f(x)`

The actual second derivative of $f(x)$ is...

$$f''(x) = \frac{-1}{4}e^x[(\pi^2 - 4)\sin(\frac{\pi x}{2}) - 4\pi \cos(\frac{\pi x}{2})]$$

Described in the function, `problem_1.f_double_prime(x)`:

```
1 function result = problem_1.f_double_prime(x)
2     result = -(1/4) * exp(x) * (((pi^2)-4)*sin(pi*x/2)-4*pi*cos(pi*x/2)); % Returns
3     actual f''(x) for Problem A
end
```

`problem_1.f_double_prime(x)`

However using the 2nd derivative centered difference approximation:

$$f''(x_i) \approx u''_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}$$

We can also approximate the second derivative. The 2nd derivative centered difference approximation is described in the function, `f_double_prime(ui_plus,ui,ui_minus,h)`:

```
1 function result = f_double_prime(ui_plus,ui,ui_minus,h)
2     result = (ui_plus - (2*ui) + ui_minus)/(h^2); % centered finite difference
3     approximation of f''(x)
end
```

`f_double_prime(ui_plus,ui,ui_minus,h)`

In order to get all of the estimated and actual $f''(x_i)$ values, we call the `problem_1(x_interval,h)` function. The `problem_1(x_interval,h)` function takes the inputs, `x_interval` and `h`, which were initialized beforehand. The function returns...

- (a) `estimate_interval`: an array of estimated $f''(x)$ values (based off of the 2nd derivative centered difference approximation)
- (b) `actual_interval`: an array of actual $f''(x)$ values
- (c) `estimate_middle`: the value of the element in the center/middle of `estimate_interval` (to be used later for log-log plots)
- (d) `actual_middle`: the value of the element in the center/middle of `actual_interval` (to be used later for log-log plots)

```

1 [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
2 problem_1(x_interval, h); % returns estimate vs actual and middle estimate vs actual
   for reference

   calling problem_1(x_interval, h)

1 function [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
2     problem_1(interval, h) % Problem A
3     interval_length = length(interval); % saves number of xi's in x_interval
4     estimate_interval = zeros(1, length(interval)); % makes empty array for estimate f''(x)
5     actual_interval = zeros(1, length(interval)); % makes empty array for actual f''(x)
6     for n=1:interval_length
7         xi = interval(n); % goes through each xi
8         ui = problem_1_f(xi); % returns f(xi) for f''(xi) estimation
9         ui_plus = problem_1_f(xi + h); % returns f(xi+h) for f''(xi)
10        estimation
11        ui_minus = problem_1_f(xi - h); % returns f(xi-h) for f''(xi)
12        estimation
13        estimate_interval(n) = f_double_prime(ui_plus, ui, ui_minus, h); % estimate f''(xi)
14        actual_interval(n) = problem_1_f_double_prime(xi); % actual f''(xi)
15    end
16    estimate_middle = estimate_interval((interval_length+1)/2); % since
17    x_interval will always have an odd number of values,
18    actual_middle = actual_interval((interval_length+1)/2); % (
19    interval_length+1)/2 will always return middle estimate/actual
20 end

   problem_1(x_interval, h)

```

Plotting the estimate and actual intervals against x_interval, we get...

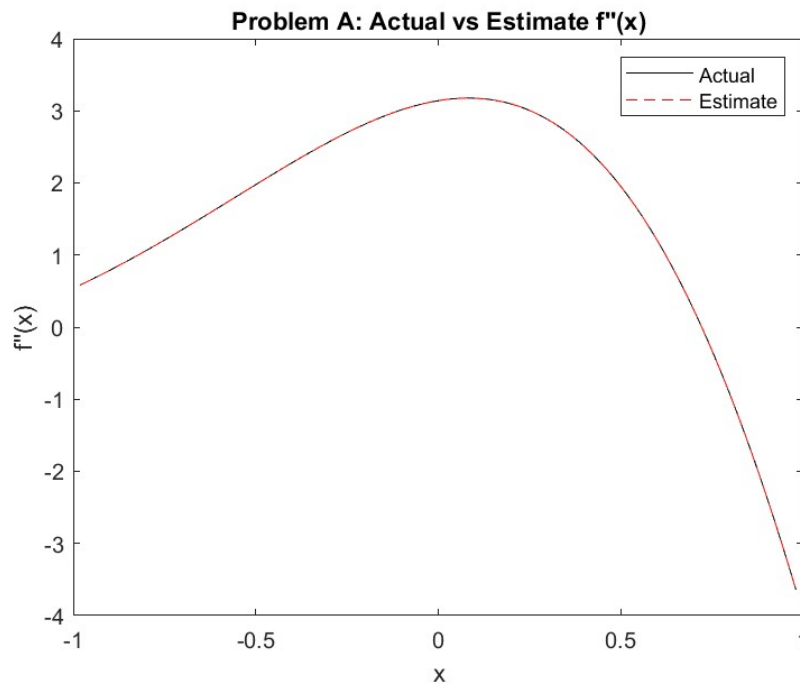


Figure 1: Problem A Actual vs Estimate $f''(x)$

```

1 % Plotting Problem A: Actual vs Estimate f''(x)
2 plot(x_interval, actual_interval, 'k-')
3 hold on;
4 plot(x_interval, estimate_interval, 'r--')
5 legend("Actual", "Estimate")

```

```

6 xlabel("x")
7 ylabel("f'('x)")
8 title("Problem A: Actual vs Estimate f'('x)")
9 hold off

```

Plotting Problem A

In order to display the log-log plot, we call the `grid_reduction(h, number_of_grids, problem_number)`, which takes in `h`, the number of grids, and the problem number (1 for A and 2 for B). It returns...

- (a) `h_interval`: an array of all `h` intervals (ex. $[h, \frac{h}{2}, \frac{h}{4}, \frac{h}{8}, \dots]$)
- (b) `error_list`: the error approximation for each $\frac{h}{2^n}$.

```

1 problem_number = 1; % 1 for Problem A
2 [h_interval, error_list] = grid_reduction(h, number_of_grids, problem_number);
3

```

calling `grid_reduction(h, number_of_grids, problem_number)`

```

1 function [h_interval, error_list] = grid_reduction(h, number_of_grids, problem_number)
2     h_interval = zeros(1, number_of_grids); % sets empty array for future h/# values
3     error_list = zeros(1, number_of_grids); % sets empty array for error
4     approximations
5     h_div = h; % initializes first h = h
6     for n=1:number_of_grids
7         h_interval(n) = h_div; % replaces 0 in empty array with actual h
8         /# value
9         interval = -1+h_div : h_div : 1-h_div; % makes fresh xi interval with h/# as gap
10        if problem_number == 1 % If problem A, returns estimate vs actual for each
11            xi based on h/# gap AND estimate and actual middle value for loglog graph
12            [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
13                problem_1(interval, h_div);
14        elseif problem_number == 2 % If problem B...
15            [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
16                problem_2(interval, h_div);
17        end
18        error = (abs(estimate_middle-actual_middle))^(1.0/number_of_grids); % error
19        approximation
20        error_list(n) = error; % replaces 0 in empty error_list array
21        with error approximation
22        h_div = h/(2^n); % sets new h value (h/(2^grid_size))
23    end
24 end

```

`grid_reduction(h, number_of_grids, problem_number)`

We can then create the log-log plot with respect to `h_interval` and `error_list...`

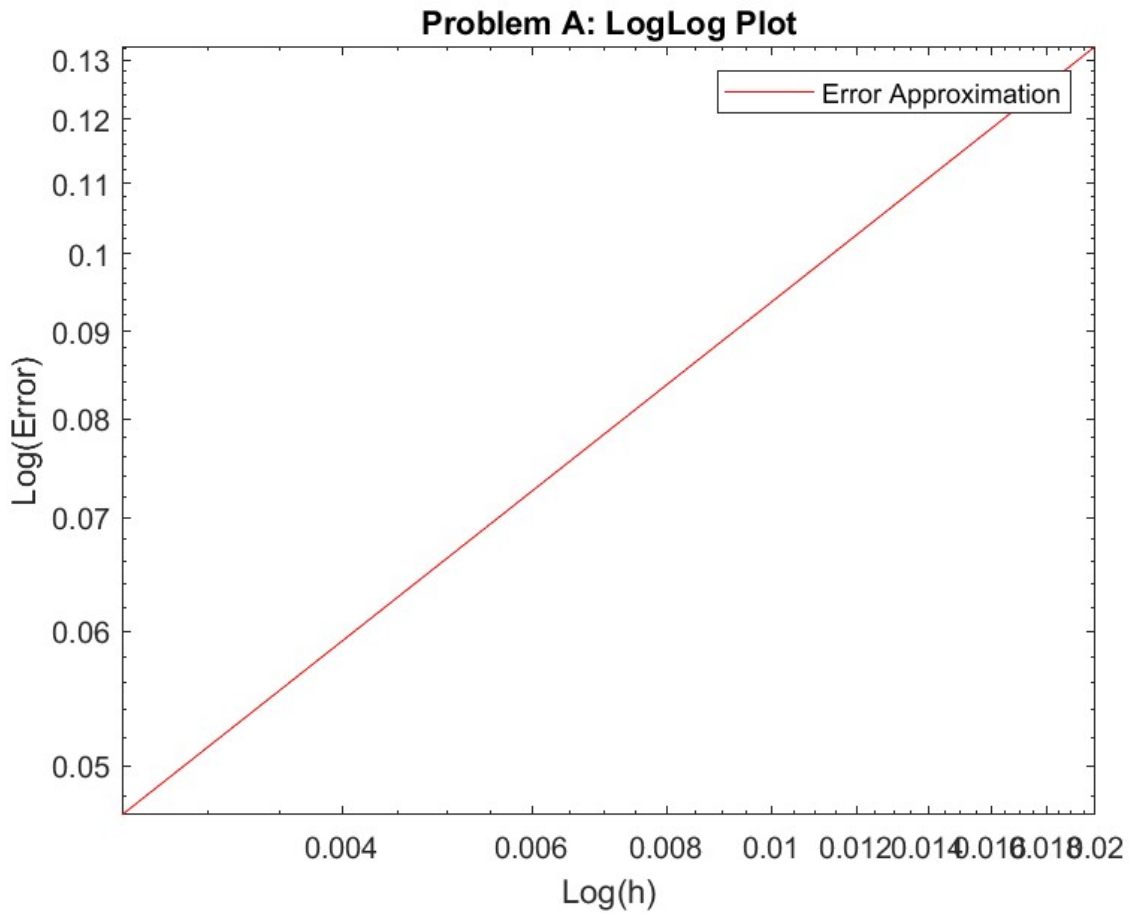


Figure 2: Problem A LogLog Plot

```
1 % Plotting Problem A: LogLog Plot
2 figure(2)
3 loglog(h_interval, error_list, 'r-')
4 legend("Error Approximation")
5 xlabel("Log(h)")
6 ylabel("Log(Error)")
7 title("Problem A: LogLog Plot")
8 hold off
9
```

Plotting Problem A LogLog Plot

b) $f(x) = 2\cos^2(\pi x) - 1$

Described in the function, problem_2.f(x):

```
1 function result = problem_2.f(x)
2     result = 2*(cos(pi*x)^2)-1; % Returns f(x) for Problem B
3 end
```

problem_2.f(x)

The actual second derivative of $f(x)$ is...

$$f''(x) = 4\pi^2[(\sin^2(\pi x)) - (\cos^2(\pi x))]$$

Described in the function, problem_2.f_double_prime(x):

```
1 function result = problem_2.f_double_prime(x)
2     result = 4*(pi^2)*((sin(pi*x)^2)-(cos(pi*x)^2)); % Returns actual f''(x) for Problem
   B
3 end
```

problem_2.f_double_prime(x)

However using the 2nd derivative centered difference approximation:

$$f''(x_i) \approx u''_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}$$

We can again approximate the second derivative.

Problem B works exactly like Problem A at this point, except using a different function, problem_1(x_interval,h) to solve for $f''(x_i)$. Only thing different is that it calls problem_2.f(xi) and problem_2.f_double_prime(xi).

```
1 function [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
2     problem_2(interval, h) % Problem B
3     interval_length = length(interval); % ...
4     estimate_interval = zeros(1, length(interval));
5     actual_interval = zeros(1, length(interval));
6     for n=1:interval_length
7         xi = interval(n);
8         ui = problem_2.f(xi);
9         ui_plus = problem_2.f(xi + h);
10        ui_minus = problem_2.f(xi - h);
11        estimate_interval(n) = f_double_prime(ui_plus, ui, ui_minus, h);
12        actual_interval(n) = problem_2.f_double_prime(xi);
13    end
14    estimate_middle = estimate_interval((interval_length+1)/2);
15    actual_middle = actual_interval((interval_length+1)/2);
16 end
```

problem_2.f_double_prime(x)

The actual vs estimate plot for Problem B looks like ...

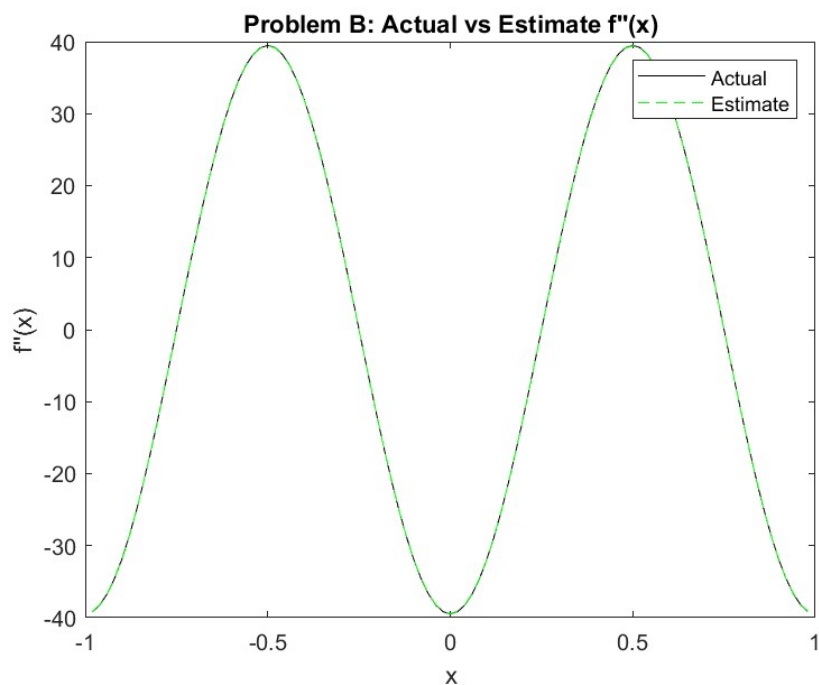


Figure 3: Problem B Actual vs Estimate $f''(x)$

```

1
2 % Plotting Problem B: Actual vs Estimate f''(x)
3 figure(3)
4 plot(x_interval, actual_interval, 'k-')
5 hold on;
6 plot(x_interval, estimate_interval, 'g--')
7 legend("Actual", "Estimate")
8 xlabel("x")
9 ylabel("f''(x)")
10 title("Problem B: Actual vs Estimate f''(x)")
11 hold off

```

Plotting Problem A

The LogLog plot for Problem B looks like ...

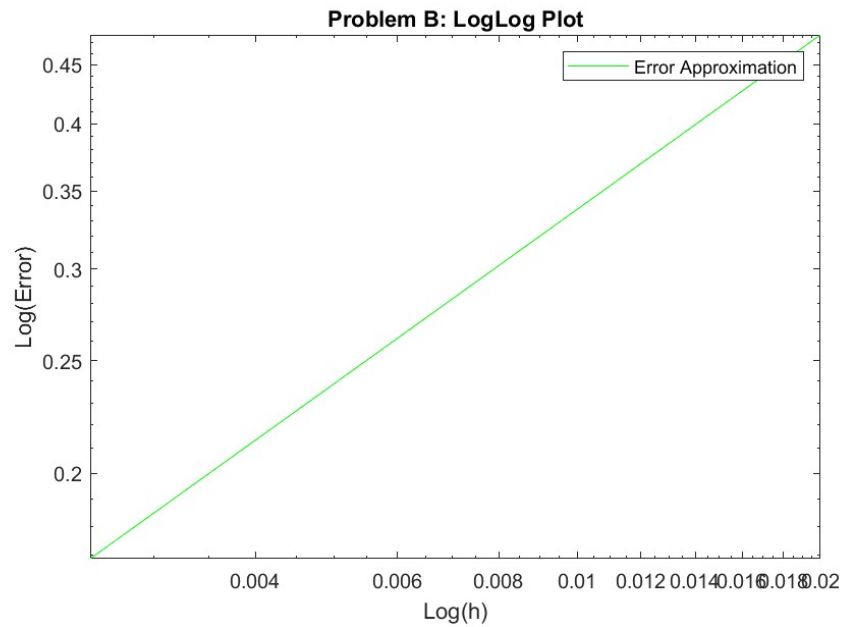


Figure 4: Problem B LogLog Plot

```

1 problem_number = 2; % 2 for Problem B
2 [h_interval, error_list] = grid_reduction(h,number_of_grids, problem_number);
3
4 % Plotting Problem B: LogLog Plot
5 figure(4)
6 loglog(h_interval, error_list, 'g-')
7 legend("Error Approximation")
8 xlabel("Log(h)")
9 ylabel("Log(Error)")
10 title("Problem B: LogLog Plot")
11 hold off

```

Plotting Problem B LogLog Plot

```

1 %% Zack Humphries
2 % COMP 521
3 % HW2
4
5 clc;          % clear command window
6 clear;        % removes all saved variables
7 close all;    % close any open windows
8
9 %%
10 h = 0.02;      % sets h
11 number_of_grids = 4; % grid sizes for error plot
12 x_interval = -1+h : h : 1-h; % sets interval [-1 to 1] for xi with h gap
13
14 %% Problem A
15 [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
16     problem_1(x_interval, h); % returns estimate vs actual and middle estimate vs actual
17     for reference
18
19 % Plotting Problem A: Actual vs Estimate f''(x)
20 plot(x_interval, actual_interval, 'k-')
21 hold on;
22 plot(x_interval, estimate_interval, 'r-')
23 legend("Actual", "Estimate")
24 xlabel("x")
25 ylabel("f''(x)")
26 title("Problem A: Actual vs Estimate f''(x)")
27 hold off
28
29 % Returns all h intervals (ex. [h, h/2, h/4, h/8,...]) and the error
30 % approximation for each h/(2^n). Takes in problem_number (1 for A and 2
31 % for B)
32 problem_number = 1; % 1 for Problem A
33 [h_interval, error_list] = grid_reduction(h, number_of_grids, problem_number);
34
35 % Plotting Problem A: LogLog Plot
36 figure(2)
37 loglog(h_interval, error_list, 'r-')
38 legend("Error Approximation")
39 xlabel("Log(h)")
40 ylabel("Log(Error)")
41 title("Problem A: LogLog Plot")
42 hold off
43
44 %% Problem B
45 [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
46     problem_2(x_interval, h);
47
48 % Plotting Problem B: Actual vs Estimate f''(x)
49 figure(3)
50 plot(x_interval, actual_interval, 'k-')
51 hold on;
52 plot(x_interval, estimate_interval, 'g-')
53 legend("Actual", "Estimate")
54 xlabel("x")
55 ylabel("f''(x)")
56 title("Problem B: Actual vs Estimate f''(x)")
57 hold off
58
59 problem_number = 2; % 2 for Problem B
60 [h_interval, error_list] = grid_reduction(h, number_of_grids, problem_number);
61
62 % Plotting Problem B: LogLog Plot
63 figure(4)
64 loglog(h_interval, error_list, 'g-')
65 legend("Error Approximation")
66 xlabel("Log(h)")
67 ylabel("Log(Error)")
68 title("Problem B: LogLog Plot")
69 hold off

```



```

70
71 %% Functions used for calculations
72 function [h_interval, error_list] = grid_reduction(h, number_of_grids, problem_number)
73     h_interval = zeros(1, number_of_grids); % sets empty array for future h/# values
74     error_list = zeros(1, number_of_grids); % sets empty array for error approximations
75     h_div = h; % initializes first h = h
76     for n=1:number_of_grids
77         h_interval(n) = h_div; % replaces 0 in empty array with actual h/#
78         value
79         interval = -1+h_div : h_div : 1-h_div; % makes fresh xi interval with h/# as gap
80         if problem_number == 1 % If problem A, returns estimate vs actual for each xi
81             based on h/# gap AND estimate and actual middle value for loglog graph
82             [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
83             problem_1(interval, h_div);
84             elseif problem_number == 2 % If problem B...
85                 [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
86                 problem_2(interval, h_div);
87             end
88             error = (abs(estimate_middle - actual_middle))^(1.0/number_of_grids); % error
89             approximation
90             error_list(n) = error; % replaces 0 in empty error_list array with
91             error approximation
92             h_div = h/(2^n); % sets new h value (h/(2^grid-size))
93         end
94     end
95
96 function [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
97     problem_1(interval, h) % Problem A
98     interval_length = length(interval); % saves number of xi's in x_interval
99     estimate_interval = zeros(1, length(interval)); % makes empty array for estimate f"(x)
100     actual_interval = zeros(1, length(interval)); % makes empty array for actual f"(x)
101     for n=1:interval_length
102         xi = interval(n); % goes through each xi
103         ui = problem_1.f(xi); % returns f(xi) for f"(xi) estimation
104         ui_plus = problem_1.f(xi + h); % returns f(xi+h) for f"(xi) estimation
105         ui_minus = problem_1.f(xi - h); % returns f(xi-h) for f"(xi) estimation
106         estimate_interval(n) = f_double_prime(ui_plus, ui, ui_minus, h); % estimate f"(xi)
107         actual_interval(n) = problem_1.f_double_prime(xi); % actual f"(xi)
108     end
109     estimate_middle = estimate_interval((interval_length+1)/2); % since x_interval
110     will always have an odd number of values,
111     actual_middle = actual_interval((interval_length+1)/2); % (interval_length
112     +1)/2 will always return middle estimate/actual
113 end
114
115 function [estimate_interval, actual_interval, estimate_middle, actual_middle] = ...
116     problem_2(interval, h) % Problem B
117     interval_length = length(interval); % ...
118     estimate_interval = zeros(1, length(interval));
119     actual_interval = zeros(1, length(interval));
120     for n=1:interval_length
121         xi = interval(n);
122         ui = problem_2.f(xi);
123         ui_plus = problem_2.f(xi + h);
124         ui_minus = problem_2.f(xi - h);
125         estimate_interval(n) = f_double_prime(ui_plus, ui, ui_minus, h);
126         actual_interval(n) = problem_2.f_double_prime(xi);
127     end
128     estimate_middle = estimate_interval((interval_length+1)/2);
129     actual_middle = actual_interval((interval_length+1)/2);
130 end
131
132 function result = f_double_prime(ui_plus, ui, ui_minus, h)
133     result = (ui_plus - (2*ui) + ui_minus)/(h^2); % centered finite difference
134     approximation of f"(x)
135 end
136
137 function result = problem_1.f(x)
138     result = exp(x)*sin((pi*x)/2); % Returns f(x) for Problem A
139 end

```

```
133
134 function result = problem_1_f_double_prime(x)
135     result = -(1/4) * exp(x) * (((pi^2)-4)*sin(pi*x/2)-4*pi*cos(pi*x/2)); % Returns actual f''(
136     x) for Problem A
137 end
138 function result = problem_2_f(x)
139     result = 2*(cos(pi*x)^2)-1; % Returns f(x) for Problem B
140 end
141
142 function result = problem_2_f_double_prime(x)
143     result = 4*(pi^2)*((sin(pi*x)^2)-(cos(pi*x)^2)); % Returns actual f''(x) for Problem B
144 end
```

Homework 2