COMP 521

Zack Humphries **Homework 2** September 22, 2022

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

```
h = 0.02;                          % sets h
number_of_grids = 4;               % grid sizes for error plot
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):

```
function result = problem_1_f(x)
    result = exp(x)*sin((pi*x)/2);  % Returns f(x) for Problem A
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):

```
function result = problem_1_f_double_prime(x)
    result = -(1/4) * exp(x)*(((pi^2)-4)*sin(pi*x/2)-4*pi*cos(pi*x/2)); % Returns 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):

```
function result = f_double_prime(ui_plus, ui, ui_minus, h)
    result = (ui_plus - (2*ui) + ui_minus)/(h^2);   % centered finite difference 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)

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

calling problem_1(x_interval,h)

```
function [estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
      problem_1(interval,h)                          % Problem A
  interval_length = length(interval);                % saves number of xi's in x_interval
  estimate_interval = zeros(1,length(interval));     % makes empty array for estimate f"(x)
  actual_interval = zeros(1,length(interval));       % makes empty array for actual f"(x)
  for n=1:interval_length
      xi = interval(n);                              % goes through each xi
      ui = problem_1_f(xi);                          % returns f(xi) for f"(xi) estimation
      ui_plus = problem_1_f(xi + h);                 % returns f(xi+h) for f"(xi) estimation
      ui_minus = problem_1_f(xi - h);                % returns f(xi+h) for f"(xi) estimation
      estimate_interval(n) = f_double_prime(ui_plus,ui,ui_minus,h);   % estimate f"(xi)
      actual_interval(n) = problem_1_f_double_prime(xi);              % actual f"(xi)
  end
  estimate_middle = estimate_interval((interval_length+1)/2);         % since x_interval will always have an odd number of values,
  actual_middle = actual_interval((interval_length+1)/2);             % (interval_length+1)/2 will always return middle estimate/actual
end
```

problem_1(x_interval,h)

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

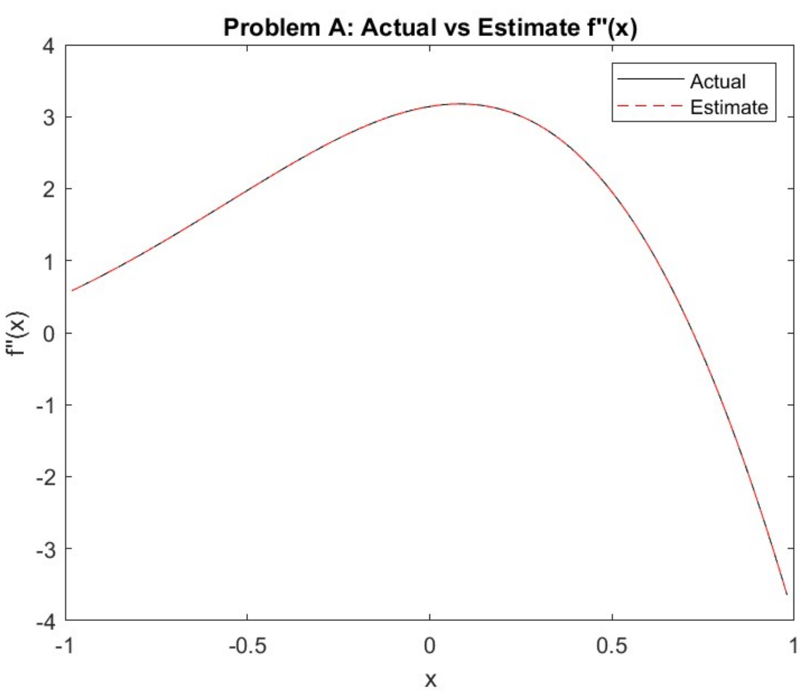

Figure 1: Problem A Actual vs Estimate $f''(x)$

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

```
xlabel("x")
ylabel("f``(x)")
title("Problem A: Actual vs Estimate f``(x)")
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. $[h, \frac{h}{2}, \frac{h}{4}, \frac{h}{8}, \ldots]$)

(b) error_list: the error approximation for each $\frac{h}{2^n}$.

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

```

calling grid_reduction(h, number_of_grids, problem_number)

```
function [h_interval, error_list] = grid_reduction(h,number_of_grids, problem_number)
  h_interval = zeros(1,number_of_grids);      % sets empty array for future h/# values
  error_list = zeros(1,number_of_grids);      % sets empty array for error approximations
  h_div = h;                                  % initializes first h = h
  for n=1:number_of_grids
      h_interval(n) = h_div;                  % replaces 0 in empty array with actual h/# value
      interval = -1+h_div : h_div : 1-h_div;  % makes fresh xi interval with h/# as gap
      if problem_number == 1      % If problem A, returns estimate vs actual for each xi based on h/# gap AND estimate and actual middle value for loglog graph
          [estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
              problem_1(interval,h_div);
      elseif problem_number == 2  % If problem B...
          [estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
              problem_2(interval,h_div);
      end
      error = (abs(estimate_middle-actual_middle))^(1.0/number_of_grids); % error approximation
      error_list(n) = error;                  % replaces 0 in empty error_list array with error approximation
      h_div = h/(2^n);                        % sets new h value (h/(2^grid_size))
  end
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...

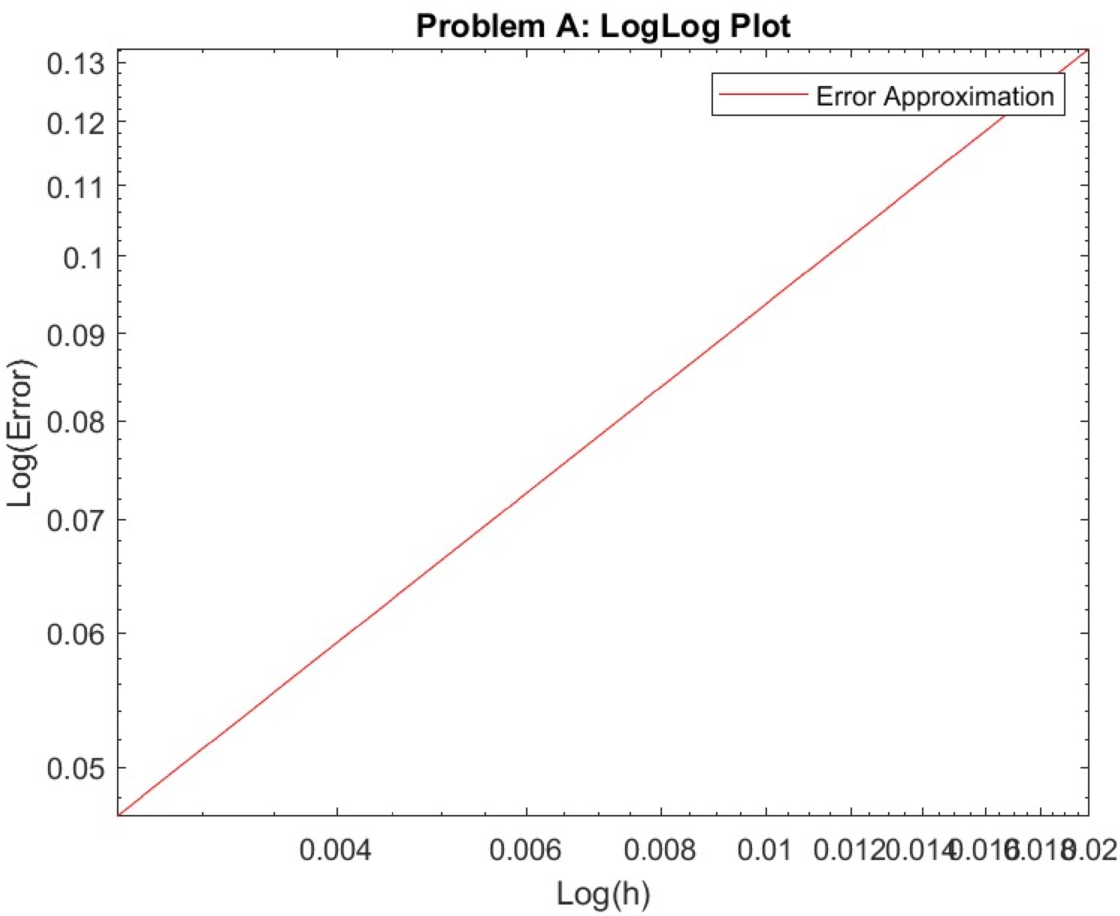

Figure 2: Problem A LogLog Plot

```
% Plotting Problem A: LogLog Plot
figure(2)
loglog(h_interval, error_list, 'r-')
legend("Error Approximation")
xlabel("Log(h)")
ylabel("Log(Error)")
title("Problem A: LogLog Plot")
hold off

```

Plotting Problem A LogLog Plot

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

Described in the function, problem_2_f(x):

```
function result = problem_2_f(x)
    result = 2*(cos(pi*x)^2)-1;  % Returns f(x) for Problem B
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):

```
function result = problem_2_f_double_prime(x)
    result = 4*(pi^2)*((sin(pi*x)^2)-(cos(pi*x)^2)); % Returns actual f"(x) for Problem B
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).

```
function [estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
        problem_2(interval,h)                          % Problem B
    interval_length = length(interval);                % ...
    estimate_interval = zeros(1,length(interval));
    actual_interval = zeros(1,length(interval));
    for n=1:interval_length
        xi = interval(n);
        ui = problem_2_f(xi);
        ui_plus = problem_2_f(xi + h);
        ui_minus = problem_2_f(xi - h);
        estimate_interval(n) = f_double_prime(ui_plus,ui,ui_minus,h);
        actual_interval(n) = problem_2_f_double_prime(xi);
    end
    estimate_middle = estimate_interval((interval_length+1)/2);
    actual_middle = actual_interval((interval_length+1)/2);
end
```

problem_2_f_double_prime(x)

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

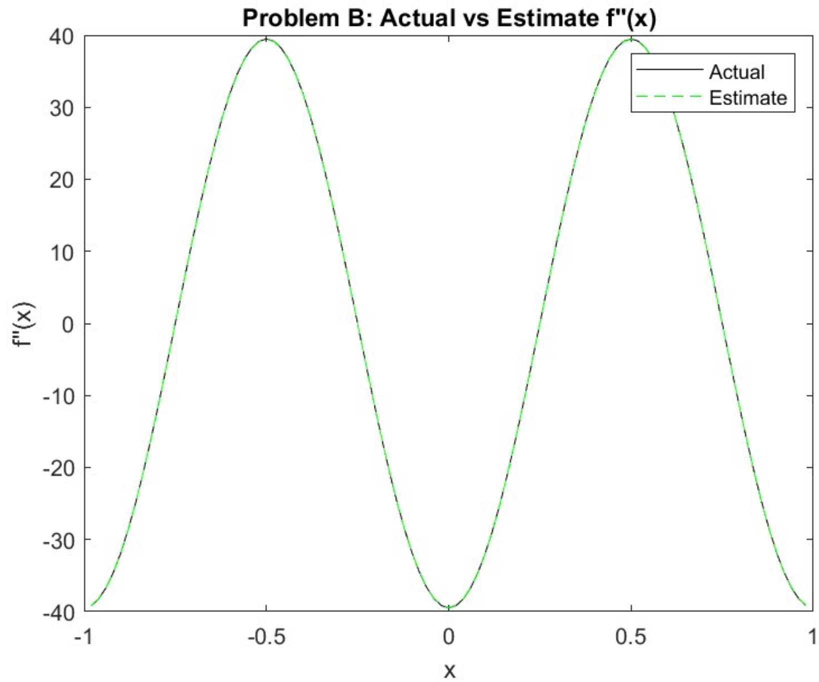

Figure 3: Problem B Actual vs Estimate $f''(x)$

```
% Plotting Problem B: Actual vs Estimate f''(x)
figure(3)
plot(x_interval, actual_interval, 'k-')
hold on;
plot(x_interval, estimate_interval, 'g--')
legend("Actual", "Estimate")
xlabel("x")
ylabel("f``(x)")
title("Problem B: Actual vs Estimate f``(x)")
hold off
```

Plotting Problem A

The LogLog plot for Problem B looks like . . .

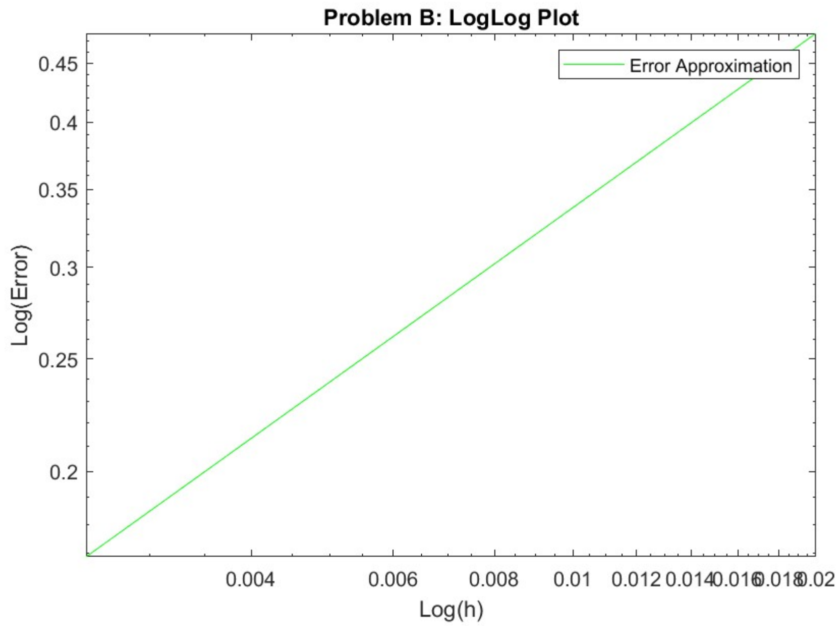

Figure 4: Problem B LogLog Plot

```
problem_number = 2; % 2 for Problem B
[h_interval, error_list] = grid_reduction(h,number_of_grids, problem_number);

% Plotting Problem B: LogLog Plot
figure(4)
loglog(h_interval, error_list, 'g-')
legend("Error Approximation")
xlabel("Log(h)")
ylabel("Log(Error)")
title("Problem B: LogLog Plot")
hold off
```

Plotting Problem B LogLog Plot

```
%% Zack Humphries
% COMP 521
% HW2

clc;       % clear command window
clear;     % removes all saved variables
close all; % close any open windows

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

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

% Plotting Problem A: Actual vs Estimate f''(x)
plot(x_interval, actual_interval, 'k-')
hold on;
plot(x_interval, estimate_interval, 'r--')
legend("Actual", "Estimate")
xlabel("x")
ylabel("f``(x)")
title("Problem A: Actual vs Estimate f``(x)")
hold off


% Returns all h intervals (ex. [h, h/2, h/4, h/8,...]) and the error
% approximation for each h/(2^n). Takes in problem_number (1 for A and 2
% for B)
problem_number = 1; % 1 for Problem A
[h_interval, error_list] = grid_reduction(h,number_of_grids, problem_number);

% Plotting Problem A: LogLog Plot
figure(2)
loglog(h_interval, error_list, 'r-')
legend("Error Approximation")
xlabel("Log(h)")
ylabel("Log(Error)")
title("Problem A: LogLog Plot")
hold off

%% Problem B
[estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
    problem_2(x_interval,h);

% Plotting Problem B: Actual vs Estimate f''(x)
figure(3)
plot(x_interval, actual_interval, 'k-')
hold on;
plot(x_interval, estimate_interval, 'g--')
legend("Actual", "Estimate")
xlabel("x")
ylabel("f``(x)")
title("Problem B: Actual vs Estimate f``(x)")
hold off

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

% Plotting Problem B: LogLog Plot
figure(4)
loglog(h_interval, error_list, 'g-')
legend("Error Approximation")
xlabel("Log(h)")
ylabel("Log(Error)")
title("Problem B: LogLog Plot")
hold off
```

```matlab

%% Functions used for calculations
function [h_interval, error_list] = grid_reduction(h,number_of_grids, problem_number)
    h_interval = zeros(1,number_of_grids);      % sets empty array for future h/# values
    error_list = zeros(1,number_of_grids);      % sets empty array for error approximations
    h_div = h;                                  % initializes first h = h
    for n=1:number_of_grids
        h_interval(n) = h_div;                  % replaces 0 in empty array with actual h/# value
        interval = -1+h_div : h_div : 1-h_div;  % makes fresh xi interval with h/# as gap
        if problem_number == 1      % If problem A, returns estimate vs actual for each xi based on h/# gap AND estimate and actual middle value for loglog graph
            [estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
                problem_1(interval,h_div);
        elseif problem_number == 2  % If problem B...
            [estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
                problem_2(interval,h_div);
        end
        error = (abs(estimate_middle-actual_middle))^(1.0/number_of_grids); % error approximation
        error_list(n) = error;                  % replaces 0 in empty error_list array with error approximation
        h_div = h/(2^n);                        % sets new h value (h/(2^grid_size))
    end
end

function [estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
        problem_1(interval,h)                           % Problem A
    interval_length = length(interval);                 % saves number of xi's in x_interval
    estimate_interval = zeros(1,length(interval));      % makes empty array for estimate f"(x)
    actual_interval = zeros(1,length(interval));        % makes empty array for actual f"(x)
    for n=1:interval_length
        xi = interval(n);                               % goes through each xi
        ui = problem_1_f(xi);                           % returns f(xi) for f"(xi) estimation
        ui_plus = problem_1_f(xi + h);                  % returns f(xi+h) for f"(xi) estimation
        ui_minus = problem_1_f(xi - h);                 % returns f(xi+h) for f"(xi) estimation
        estimate_interval(n) = f_double_prime(ui_plus,ui,ui_minus,h);   % estimate f"(xi)
        actual_interval(n) = problem_1_f_double_prime(xi);              % actual f"(xi)
    end
    estimate_middle = estimate_interval((interval_length+1)/2);         % since x_interval will always have an odd number of values,
    actual_middle = actual_interval((interval_length+1)/2);             % (interval_length+1)/2 will always return middle estimate/actual
end

function [estimate_interval, actual_interval,estimate_middle,actual_middle] = ...
        problem_2(interval,h)                           % Problem B
    interval_length = length(interval);                 % ...
    estimate_interval = zeros(1,length(interval));
    actual_interval = zeros(1,length(interval));
    for n=1:interval_length
        xi = interval(n);
        ui = problem_2_f(xi);
        ui_plus = problem_2_f(xi + h);
        ui_minus = problem_2_f(xi - h);
        estimate_interval(n) = f_double_prime(ui_plus,ui,ui_minus,h);
        actual_interval(n) = problem_2_f_double_prime(xi);
    end
    estimate_middle = estimate_interval((interval_length+1)/2);
    actual_middle = actual_interval((interval_length+1)/2);
end

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

function result = problem_1_f(x)
    result = exp(x)*sin((pi*x)/2);  % Returns f(x) for Problem A
end
```

```matlab

function result = problem_1_f_double_prime(x)
    result = -(1/4) * exp(x)*(((pi^2)-4)*sin(pi*x/2)-4*pi*cos(pi*x/2)); % Returns actual f"(x) for Problem A
end

function result = problem_2_f(x)
    result = 2*(cos(pi*x)^2)-1;  % Returns f(x) for Problem B
end

function result = problem_2_f_double_prime(x)
    result = 4*(pi^2)*((sin(pi*x)^2)-(cos(pi*x)^2)); % Returns actual f"(x) for Problem B
end
```

Homework 2