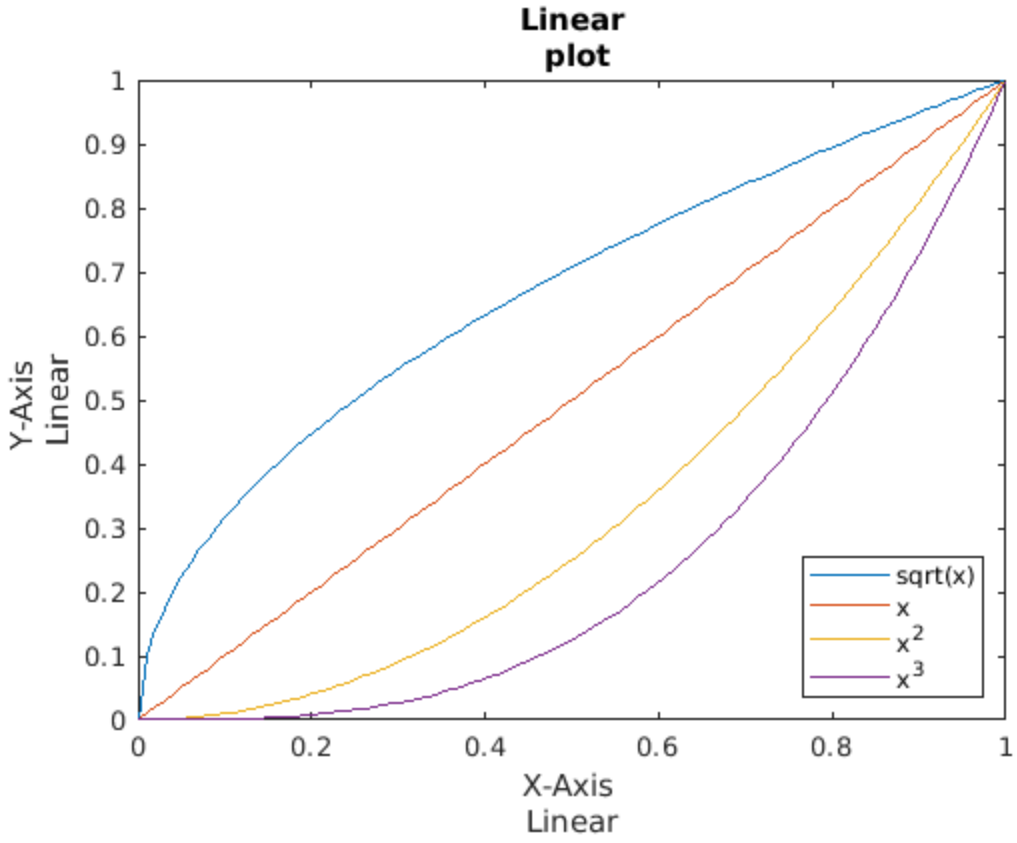
```
% Anthony Galczak - agalczak@unm.edu - WGalczak@gmail.com
% CS375 - HW1

clc % Clears command window
close all % Closes all figures
format compact % Makes disp() not output new lines

% Question 1
fprintf("------\nOutput\n------\nQuestion 1\n");

% 1a.) It sets index 1 to 0 and every 3rd index after elements 1,4,7
% are 0. i.e. z now equals [0 40 70 0 20 30 0 60]
z = [10 40 70 90 20 30 50 60];
z(1:3:7) = zeros(1,3);
fprintf("z = ");
disp(z);

% 1b.) It sets elements 3 4 1 to the empty set.
% Removes elements 1, 3, 4. i.e. z now equals [40 20 30 50 60]
z = [10 40 70 90 20 30 50 60];
z([3 4 1]) = [];
fprintf("z = ");
disp(z);

% Question 2
fprintf("\nQuestion 2\n");

% 2a.)
% (i) linspace(1,25,7) makes an array of numbers from 1 to 25 with 7
% elements.
% (ii) linspace(-11,1,13) makes an array of numbers from -11 to 1
% with 13 elements.
t=1:4:25;
t_linspace = linspace(1,25,7);
fprintf("t and t_linspace equal? %d\n", isequal(t,t_linspace));

x=-11:1;
x_linspace = linspace(-11,1,13);
fprintf("x and x_linspace equal? %d\n", isequal(x,x_linspace));

% 2b.)
% (i) 10:0.4:-8 makes an array of numbers from -10 to -8 with
% 0.4 increments.
% (ii) 0:0.25:1 makes an array of numbers from 0 to 1 with
% 0.25 increments.
v=linspace(-10,-8,6);
v_colon = -10:0.4:-8;
fprintf("v and v_colon equal? %d\n", isequal(v,v_colon));

r=linspace(0,1,5);
r_colon = 0:0.25:1;
fprintf("r and r_colon equal? %d\n", isequal(r,r_colon));
```

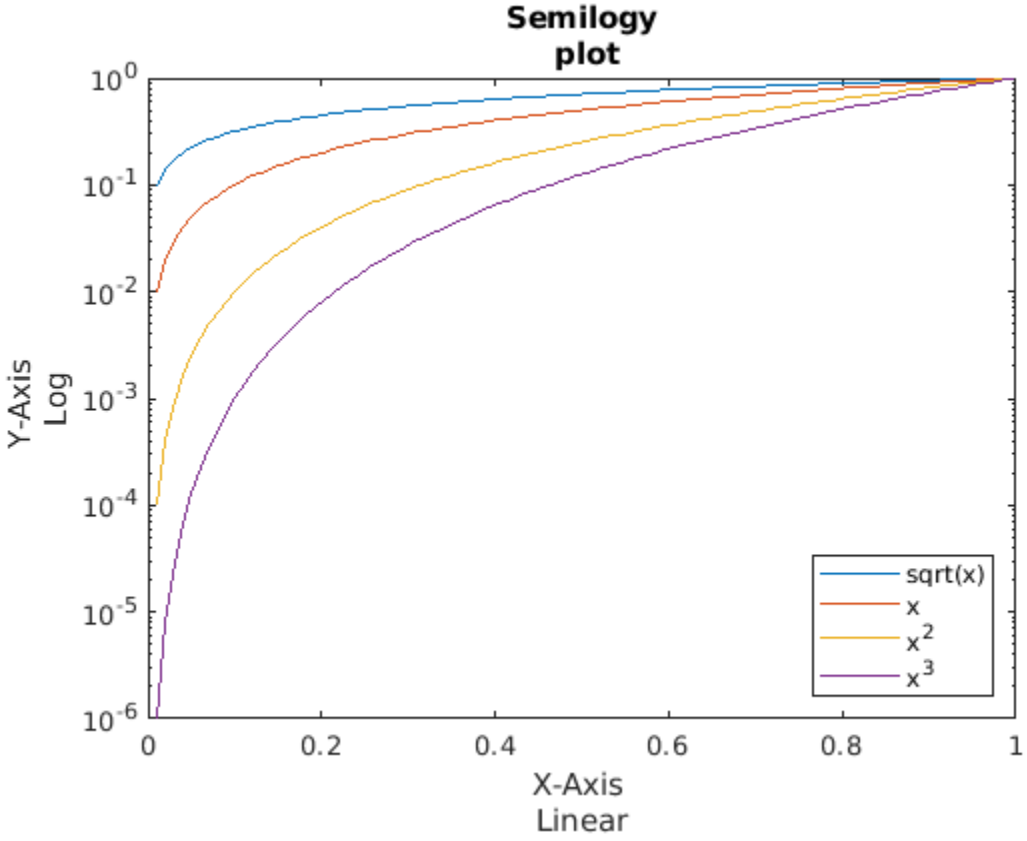
```
% Question 3
fprintf("\nQuestion 3\n");
t = 0:0.1:1;
y = sin(pi * t);

% 3a.) Sum of the elements of t
sum_tk = sum(t);
fprintf("Sum of tk: %f\n", sum_tk);

% 3b.) The vector multiplication of t and y is the sum of tkyk.
sum_tk_yk = sum(t*transpose(y));
fprintf("Sum of tk*yk: %f\n", sum_tk_yk);

% 3c.) sqrt() applied to a vector will apply the function to each
 element
% in the array.
sum_sqrt_tk = sum(sqrt(t));
fprintf("Sum of sqrt(tk): %f\n", sum_sqrt_tk);

% Question 4
nums = linspace(0,1);

% Making a list of the plot methods we want to use.
plot_functions = {@plot, @semilogy, @loglog};
title_labels = ["Linear", "Semilogy", "Loglog"];
x_labels = ["Linear", "Linear", "Log"];
y_labels = ["Linear", "Log", "Log"];

% Plotting each plot type: plot, semilogy, loglog
for i = 1 : length(plot_functions)
    figure('Name', title_labels(i), 'NumberTitle', 'Off');
    plot_functions{i}(nums, sqrt(nums));
    hold on
    title([title_labels(i) ' plot']);
    xlabel(['X-Axis ' x_labels(i)]);
    ylabel(['Y-Axis ' y_labels(i)]);
    plot_functions{i}(nums, nums);
    plot_functions{i}(nums, nums.^2);
    plot_functions{i}(nums, nums.^3);

    legend('sqrt(x)', 'x', 'x^2', 'x^3', 'Location', 'southeast');
    hold off
end

% Disadvantages and advantages of each plotting method.
% I will be specific to the functions that we are plotting here:
% sqrt(x), x, x^2, x^3

% Linear
% Advantage is that it is the natural format that we learned from
% plotting things in grade school. With a bit of prior knowledge
% you can generally recognize simple functions like x^2, etc.
```

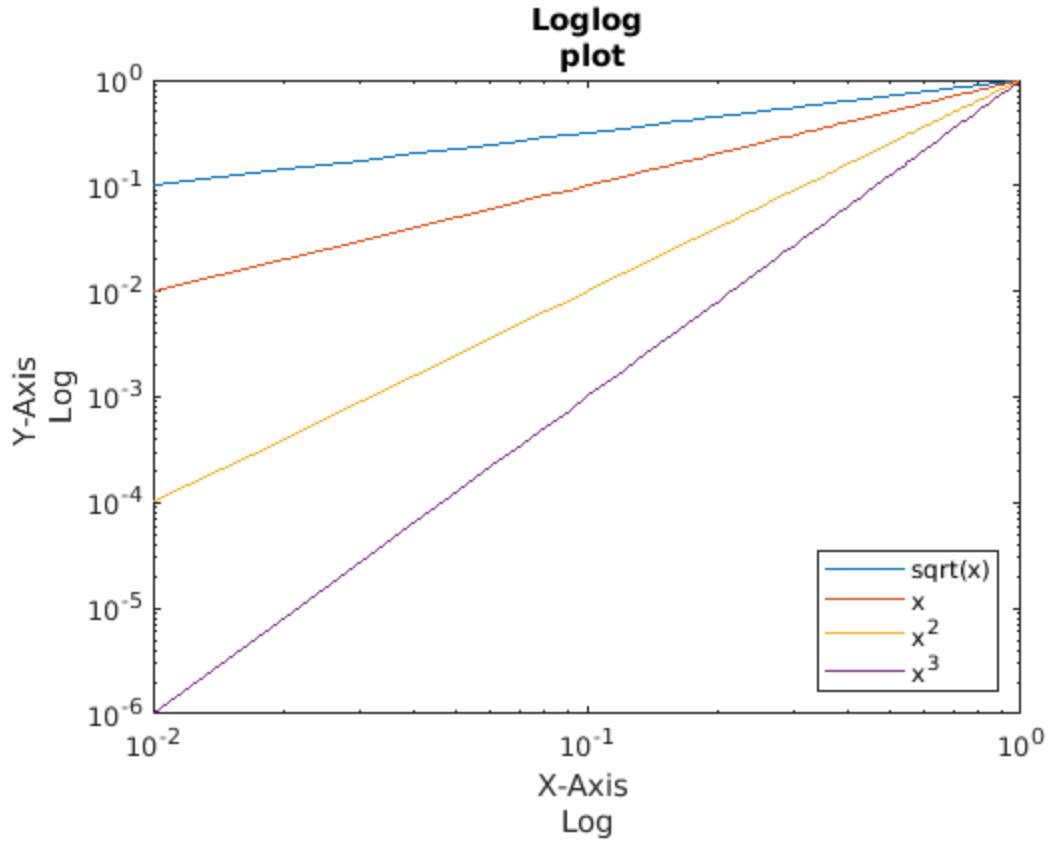
```
% Disadvantage is that it scales poorly with exponentially
% differentiated functions. In other words, telling the difference
% between x^2 and x^3 is kind of difficult, but in a log scale it
% would not be.

% Semilogy
% Advantage is that on the Y-Axis we have log scale which
% drastically shows the difference between exponents.

% Disadvantage is that we have an entire boxplot where we are
% only using ~25% of the whole plot. This is a symptom of our
% functions that we are plotting. This shows the functions very
% compressed and difficult to read as it approaches 1.

% Loglog
% Advantage is that differences in exponents are perfectly
% differentiable. Since these functions deal with a difference in
% exponents I believe this is the correct plot type to use here.

% Disadvantage is that loglog may be quite unfamiliar to those
% not more "mathematically inclined". Of course, it isn't the plot
% type we learned in grade school.

% Question 5
fprintf("\nQuestion 5\n");

% 5a.) Test case
% This is just the integral of sin(x) from 0 to 2,
% i.e. -cos(2) - (-cos(0))
approx_5a = my_mean(@sin,0,2,100);
fprintf("approx_5a: %f\n", approx_5a);
exact_5a = -cos(2) + cos(0);
fprintf("exact_5a: %f\n", exact_5a);

% 5b.) Test case (Tests for single element and a vector)
approx_5b_single = my_fun(2);
fprintf("approx_5b_single: %f\n", approx_5b_single);
exact_5b_single = 2 * exp(2);
fprintf("exact_5b_single: %f\n", exact_5b_single);

format short g % Outputs the vectors in a shortened format
approx_5b_vector = my_fun([2,4,8]);
fprintf("approx_5b_vector");
disp(approx_5b_vector);
exact_5b_vector = [2 * exp(2), 4 * exp(4), 8 * exp(8)];
fprintf("exact_5b_vector");
disp(exact_5b_vector);

% 5c.) Computing an approximation for x*exp(x) from -1 to 1.
approximations_list = zeros(8,1);
N_list = [10; 20; 40; 80; 160; 320; 640; 1280];
for i = 1:length(N_list)
    approximations_list(i) = my_mean(@my_fun, -1, 1, N_list(i));
end
```

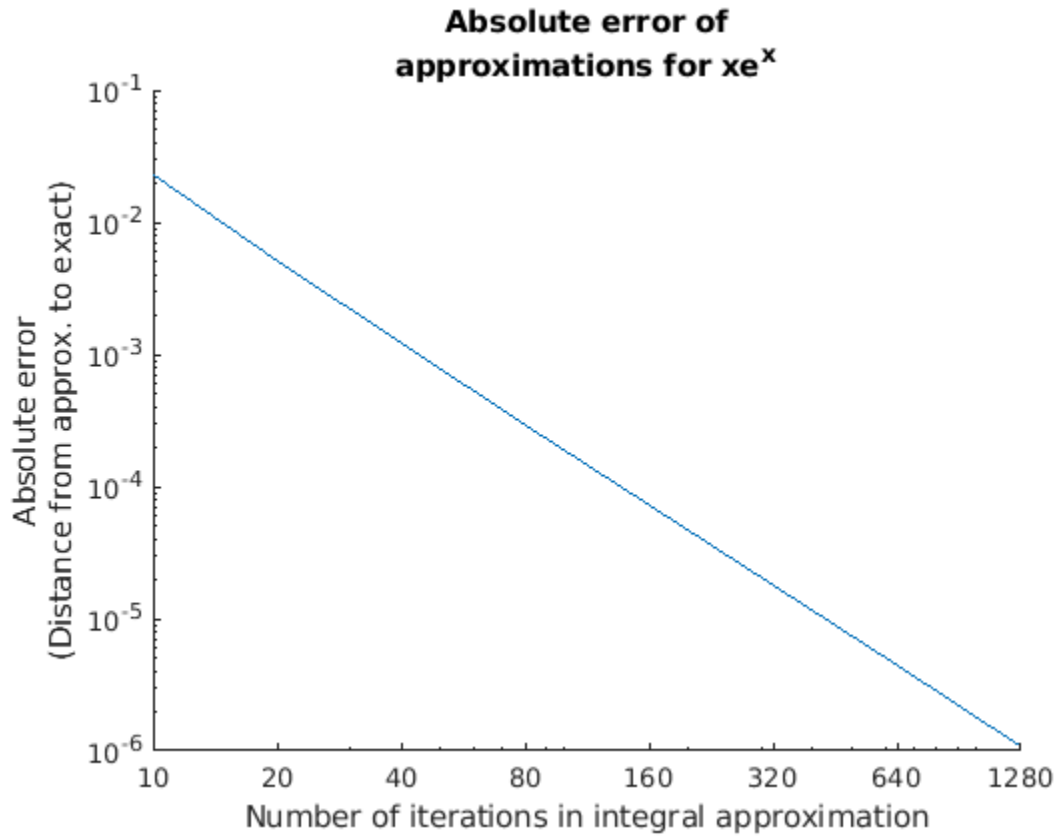
```
abs_errors = abs(approximations_list - (2 / exp(1)));

figure('Name', 'Abs error of approxiations', 'NumberTitle', 'Off');
hold on
title({'Absolute error of';'approximations for xe^x'});
loglog(N_list, abs_errors);
set(gca, 'XScale', 'log', 'YScale', 'log');
xticks(N_list);
xlabel("Number of iterations in integral approximation");
ylabel({'Absolute error'; '(Distance from approx. to exact)'});
hold off

% 5d.) Printing a table of N, approximations and abs. error's.
% There definitely is a pattern apparent here. As N increases,
% the absolute error (difference between approximation and the
% exact M) decreases.
% It seems to be decreasing at a quadratic rate.
% 10->20 gives .0223 -> .005, change in input is 2x, change in
% output is 4x. So the rate is quadratic.
T = table(N_list,approximations_list, abs_errors);
T.Properties.VariableNames = {'N' 'Approx' 'AbsError'};
disp(T);

% Function declarations. They have to be at the bottom in MATLAB.

% 5a.) my_mean function declaration

% Function called "my_mean" that takes a function fun, a number a,
% another number b (satisfying a<=b), and a positive integer N
function m = my_mean(fun, a, b, N)

    h = (b - a)/(N - 1);

    % Using an inline function to give us x_j at index j
    x_j = inline('a + (j-1) * h', 'a', 'j', 'h');

    % Getting the sum of h*f(x_j) for j = 2:(N-1)
    sum_2_N = 0;
    for i = 2 : (N-1)
        sum_2_N = sum_2_N + (h * fun(x_j(a,i,h)));
    end

    % The approximate integral function
    m = (h/2) * (fun(x_j(a,1,h)) + fun(x_j(a,N,h))) + sum_2_N;
end

% 5b.) my_fun function declaration

% Function called "my_fun" that takes a value x
% (or a vector of x's) and returns x * exp(x).
function f = my_fun(x)
    f = x .* exp(x);
end
```

```
------
Output
------
Question 1
z =      0    40    70     0    20    30     0    60
z =     40    20    30    50    60

Question 2
t and t_linspace equal? 1
x and x_linspace equal? 1
v and v_colon equal? 1
r and r_colon equal? 1

Question 3
Sum of tk: 5.500000
Sum of tk*yk: 3.156876
Sum of sqrt(tk): 7.105093

Question 5
approx_5a: 1.416099
exact_5a: 1.416147
approx_5b_single: 14.778112
exact_5b_single: 14.778112
approx_5b_vector       14.778       218.39        23848
exact_5b_vector       14.778       218.39        23848
     N      Approx      AbsError
    ____    _______    __________
      10     0.7581      0.022338
      20    0.74078     0.0050182
      40    0.73695     0.0011913
      80    0.73605    0.00029036
     160    0.73583    7.1681e-05
     320    0.73578    1.7808e-05
     640    0.73576    4.4381e-06
    1280    0.73576    1.1078e-06
```


Linear
plot
Y-Axis
Linear
X-Axis
Linear
sqrt(x)
x
x2
x3


Semilogy
plot
Y-Axis
Log
X-Axis
Linear
sqrt(x)
x
x2
x3


Loglog
plot
Y-Axis
Log
X-Axis
Log
sqrt(x)
x
x^2
x^3


Absolute error of
approximations for xe^x
Absolute error
(Distance from approx. to exact)
Number of iterations in integral approximation

Published with MATLAB® R2018a