Problems

1) (a) Write a code to approximate
$$\int_{-5}^{5} \frac{1}{1+x^2}\,\mathrm{d}x$$
using composite trapezoidal rule and composite Simpson's rule.

Code is attached at the end of the document.

(b) To can compute the number of intervals to use for the trapezoidal rule to get an error less than 10^{-4} we first need to maximize the second derivative of our integrand over the interval from -5 to 5:
$$M = \max_{x\in[-5,5]} \left| \frac{\mathrm{d}^2}{\mathrm{d}x^2} \frac{1}{1+x^2} \right| = \max_{x\in[-5,5]} \left| \frac{2(3x^2-1)}{(x^2+1)^3} \right| = 2.$$
Next, we can obtain n as
$$\frac{(b-a)^3 M}{12n^2} = \frac{10^3 2}{12n^2} < 10^{-4} \implies n > 1290.99.$$
So, we will take $n_T = 1291$.
Similarly, we can find n for Simpson's rule by, first, maximizing the fourth derivative of our integrand as
$$M = \max_{x\in[-5,5]} \left| \frac{\mathrm{d}^4}{\mathrm{d}x^4} \frac{1}{1+x^2} \right| = \max_{x\in[-5,5]} \left| \frac{24(5x^4-10x^2+1)}{(x^2+1)^5} \right| = 24.$$
Then, we can compute n from the error bound as
$$\frac{(b-a)^5 M}{180n^4} = \frac{10^5 24}{180n^4} < 10^4 \implies n > 107.457.$$
So, take $n_S = 108$.

(c) Running my code gives me the following results:

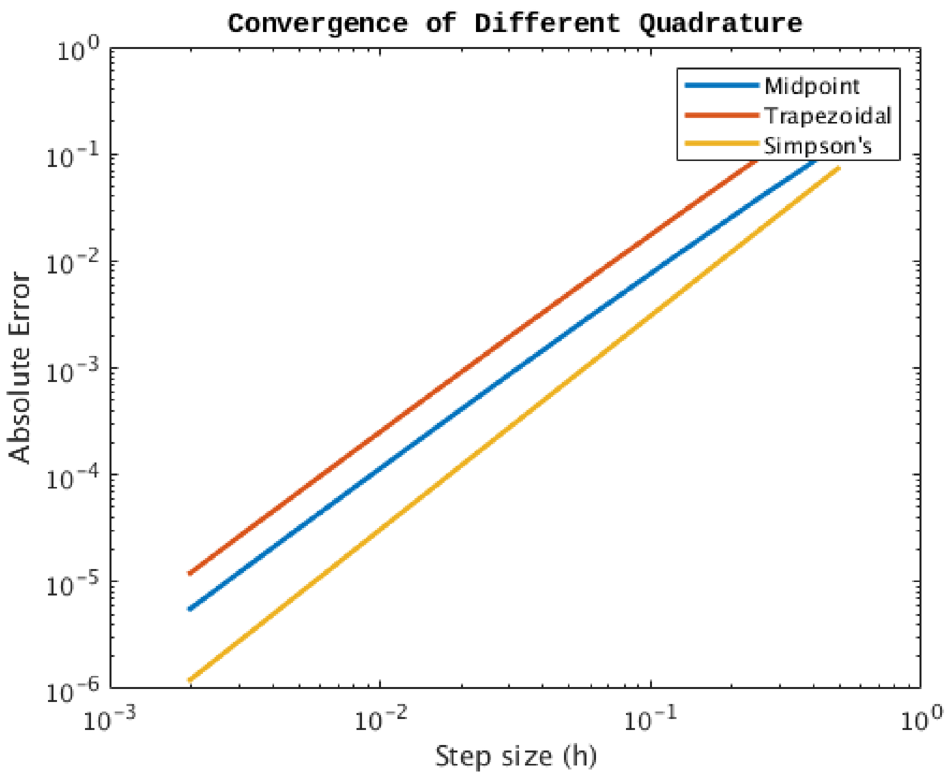
Method	Runtime (s)	# Func. Evals.	Result
Trapezoidal	$6.70 \cdot 10^{-5}$	1292	2.7468014
Simpson's	$3.80 \cdot 10^{-5}$	109	2.7478015
Quad $\varepsilon = 10^{-4}$	$1.79 \cdot 10^{-4}$	41	2.7467951
Quad $\varepsilon = 10^{-6}$	$2.08 \cdot 10^{-4}$	81	2.7468013

From the table, we can see that in every case, `quad(...)` uses less function evaluations than both composite trapezoidal and composite Simpson's. The lower number function evaluations can be attributed to `quad(...)` using an adaptive Simpson's rule which will, hopefully, optimize the nodes that we need function evaluations at to get a desired accuracy. As a result of the adaptive nodes used in `quad(...)`, we should also expect a longer runtime to find the optimal nodes instead of just using equilength intervals like trapezoidal and regular Simpson's. In fact, we do see the longer runtimes in the table. So even though `quad(...)` uses less function evaluations, it still takes longer to find the optimal nodes.

Another interesting observation comes with the accuracy of trapezoidal and Simpson's rule. Even though we computed the required number of intervals to use to obtain an error of 10^{-4}

both trapezoidal and Simpon's achieved a good few more digits accuracy than the required error tolerance. This better convergence is due to the error bounds being just that, bounds on the error. In other words, the worst our error could be with our chosen number of intervals is 10^{-4} but in practice we will do a little better.

2) Using my code, I was able to obtain the convergence plot below



Just as expected from our computed error bounds in class, we see that both midpoint and trapezoidal rule converge at the same rate with the error of the midpoint rule being slightly better. Furthermore, we can see that Simpson's rule has a higher convergence rate than both midpoint and trapezoidal rule which is right in line with the error bounds we found in class.

Code is attached at the end of the document.

Code Used

Problem 1

```
%%
% Composite trapezoidal and Simpson's rule timings
%
% Author: Caleb Jacobs
% Date last modified: 17-Nov-2021

%% Settings
format long

%% Parameters
f = @(x) 1 ./ (1 + x.^2);        % Function to integrate
a = -5;                          % Left endpoint
b =  5;                          % Right endpoint
n1 = 108;                        % Number of intervals for Simpson's
n2 = 1291;                       % Number of intervals for Trapezoidal

%% Compute integrals and timings
fprintf('Trapezoidal runtime\n')
tic
    T = trapz(f, a, b, n2);
toc

fprintf("\nSimpson's runtime\n")
tic
    S = simps(f, a, b, n1);
toc

fprintf('\nMidpoint runtime\n')
tic
    M = mid(f, a, b, n2);
toc

fprintf('\nQuad runtime with error of 10^-4\n')
tic
    [Q1, nQ1] = quad(f, a, b, 1e-4);
toc

fprintf('\nQuad runtime with error of 10^-6\n')
tic
    [Q2, nQ2] = quad(f, a, b, 1e-6);
toc

%% Display quadrature results
fprintf('\nTrapezoidal\tf evals = %d\t%.10f\n', n2 + 1, T)
fprintf("Simpson's\tf evals = %d\t%.10f\n", n1 + 1, S)
```

```
fprintf('Midpoint\tf evals = %d\t%.10f\n', n2, M)
fprintf('Quad 10^-4\tf evals = %d\t%.10f\n', nQ1, Q1)
fprintf('Quad 10^-6\tf evals = %d\t%.10f\n', nQ2, Q2)

%% Function defintions
% Composite trapezoidal rule
function val = trapz(f, a, b, n)
    xi = linspace(a, b, n + 1);          % Compute evaluation points
    h = xi(2) - a;                       % Compute x spacing

    val = h * (f(a) + f(b)) / 2;         % Add endpoint contribution

    val = val + h * sum(f(xi(2 : n)));   % Add interior contribtuion
end

% Composite Simposon's rule
function val = simps(f, a, b, n)
    % Round n up to nearest even number
    if mod(n, 2) == 0
        N = n;
    else
        N = n + 1;
    end

    xi = linspace(a, b, N + 1);              % Compute evaluation points
    h = xi(2) - a;                           % Compute x spacing

    val = f(a) + f(b);                       % Add endpoint contribution

    val = val + 4 * sum(f(xi(2 : 2 : N)));   % Add odd node contribution

    val = val + 2 * sum(f(xi(3 : 2 : N)));   % Add even node contribution

    val = h * val / 3;                       % Scale integral accordingly
end

% Composite midpoint rule
function val = mid(f, a, b, n)
    xTmp = linspace(a, b, n + 1);                 % Compute standard points
    xi   = (xTmp(2 : n + 1) + xTmp(1 : n)) / 2;   % Get midpionts
    h    = xTmp(2) - a;                           % Compute step size

    val = h * sum(f(xi));                         % Compute integral
end
```

Problem 2

```
%%
% Composite trapezoidal and Simpson's rule convergence
%
% Author: Caleb Jacobs
% Date last modified: 17-Nov-2021

%% Settings
format long

%% Parameters
f = @(x) -4 * x .* log(x);       % Function to integrate
a = 0;                           % Left endpoint
b = 1;                           % Right endpoint
N = 2 .^ [1 : 9];                % Create interval array

%% Compute integrals
errs = zeros(length(N), 3);
for i = 1 : length(N)
    n = N(i);
    EM = abs(mid(f, a, b, n) - 1);
    ET = abs(trapz(f, a, b, n) - 1);
    ES = abs(simps(f, a, b, n) - 1);

    errs(i, :) = [EM, ET, ES];
end

figure(1)
loglog((b - a) ./ N, errs, 'LineWidth', 2)
legend('Midpoint', 'Trapezoidal', "Simpson's")
title('Convergence of Different Quadrature')
xlabel('Step size (h)')
ylabel('Absolute Error')

%% Function defintions
% Composite trapezoidal rule
function val = trapz(f, a, b, n)
    xi = linspace(a, b, n + 1);         % Compute evaluation points
    h = xi(2) - a;                      % Compute x spacing

    if ~isnan(f(a))
        val = h * (f(a) + f(b)) / 2;         % Add endpoint contribution
    else
        val = h * f(b) / 2;
    end
    val = val + h * sum(f(xi(2 : n)));  % Add interior contribtuion
end

% Composite Simposon's rule
```

```matlab
function val = simps(f, a, b, n)
    % Round n up to nearest even number
    if mod(n, 2) == 0
        N = n;
    else
        N = n + 1;
    end

    xi = linspace(a, b, N + 1);             % Compute evaluation points
    h = xi(2) - a;                          % Compute x spacing

    if isnan(f(a))
        val = f(b);                         % Add endpoint contribution
    else
        val = f(a) + f(b);
    end

    val = val + 4 * sum(f(xi(2 : 2 : N)));  % Add odd node contribution

    val = val + 2 * sum(f(xi(3 : 2 : N)));  % Add even node contribution

    val = h * val / 3;                      % Scale integral accordingly
end

% Composite midpoint rule
function val = mid(f, a, b, n)
    xTmp = linspace(a, b, n + 1);               % Compute standard points
    xi   = (xTmp(2 : n + 1) + xTmp(1 : n)) / 2; % Get midpionts
    h    = xTmp(2) - a;                         % Compute step size

    val = h * sum(f(xi));                       % Compute integral
end
```