Q2

10 / 10

(a) (4 points) Using the recursive trapezoid rule to compute $\int_0^{2\pi}(\cos(2x)/e^x)dx$. Stop the iteration until the absolute difference between two consecutive computed integrals is not larger than 10^{-5}.

(b) (6 points) Using the adaptive Simpson's method to compute $\int_0^{2\pi}(\cos(2x)/e^x)dx$ by taking $\epsilon = 10^{-5}$ and `level-max`=20. Try to avoid redundant function evaluations.

For both methods, report the number of function evaluations and print the final results and the MATLAB code as well.

Note: The exact integral is $(1 - e^{-2\pi})/5$. You can use this to check if your answer is reasonable.

```
% ASSIGNMENT 6, QUESTION 2
I = (1-exp(-2*pi))/5;
f = @(x) cos(2*x) ./ exp(x);
a = 0;
b = 2*pi;

% Part (a), recursive trapezoid integral approximation
global function_evals_trape
function_evals_trape = 0;

% Call first with n=0, so m=1
I_T_1 = trape(f, a, b, 1);
delta = 1e-5;
r = rec_trape(f, a, b, delta, I_T_1, 2, (b-a)/2);
fprintf(['Approximate integral computed by recursive trapezoid rule is:' ...
    ' %e, with absolute error of %e, and %d function evaluations\n'], ...
    r, abs(I-r), function_evals_trape)

% Part (b), adaptive Simpson's method
c = (b+a)/2;
d = (c+a)/2;
e = (b+c)/2;

global function_evals_simps
% 5 done in the first call to as
function_evals_simps = 5;

numl = as(f,a,b,c,d,e,f(a),f(b),f(c),f(d),f(e),1e-5,1,20);

fprintf("\nApproximate integral computed by adaptive simpson's algorithm " +
...
    "is: %e, with absolute error of %e, and %d function evaluations", ...
    numl, abs(I-numl), function_evals_simps)
```

```
Approximate integral computed by recursive trapezoid rule is: 1.996296e-01,
with absolute error of 3.131625e-06, and 1025 function evaluations

Approximate integral computed by adaptive simpson's algorithm is:
1.996264e-01, with absolute error of 1.437424e-07, and 65 function evaluations
```

Published with MATLAB® R2024b

```
function numl = as(f, a, b, c, d, e, fa, fb, fc, fd, fe, eps, level, ...
                                                              level_max)

% input
%   f: function we are integrating over
%   a: lower limit of integration
%   b: upper limit of integration
%   c: midpoint of a and b
%   d: midpoint of a and c
%   e: midpoint of c and b
%   a: f(a)
%   b: f(b)
%   c: f(c)
%   d: f(d)
%   e: f(e)
%   eps: error tolerance
%   level: depth of recursive function call
%   level_max: maximum depth of recursive function calls

% output
%   numl:  computed integral using the adaptive simpson's method

global function_evals_simps

h = b-a;
I1 = (h*(fa + 4*fc + fb))/6;
level = level + 1;
I2 = (h*(fa + 4*fd + 2*fc + 4*fe + fb))/12;

if level >= level_max
    numl = I2;
else
    if abs(I2-I1) <= 15*eps
        numl = I2 + (1/15)*(I2-I1);
    else
        % left subinterval calculations
        a1 = a;
        b1 = c;
        c1 = d;
        d1 = (c1+a1)/2;
        e1 = (b1+c1)/2;
        l = as(f,a1,b1,c1,d1,e1,fa,fc,fd,f(d1),f(e1),eps/2,level,level_max);

        % right subinterval calculations
        a2 = c;
        b2 = b;
        c2 = e;
        d2 = (c2+a2)/2;
        e2 = (b2+c2)/2;
        r = as(f,a2,b2,c2,d2,e2,fc,fb,fe,f(d2),f(e2),eps/2,level,level_max);

        % note from above that a lot of the function evaluations of the
```

```
        % endpoints can be reused so only actually doing 4 more each time
        function_evals_simps = function_evals_simps + 4;

        numl = l + r;
    end
end
end
```

Published with MATLAB® R2024b

```
function r = rec_trape(f, a, b, delta, prev_r, m, h)

% input
%   f: function we are integrating over
%   a: lower limit of integration
%   b: upper limit of integration
%   delta: tolerance to stop iterations when consecutive computed integrals
%          are close enough
%   prev_r: result of previous call to rec_trape (I_T(2^(n-1)))
%   m: number of subintervals, m = 2^n
%   h: panel width, h = (b-a)/m
%   I: actual computed integral

% output
%   r:  result of approximate integral I_T(2^n) using the trapezoid rule

global function_evals_trape

% m = 2^n so m/2 = 2^(n-1)
x = linspace(a+h, a+(m-1)*h, m/2);

% have to evaluate function at each point in x
function_evals_trape = function_evals_trape + length(x);

r = (prev_r)/2 + h*sum(f(x));

if abs(r - prev_r) <= delta
    % If tolerance met, return computed result for integral
    return;
else
    % If tolerance not met, make recursive call
    r = rec_trape(f, a, b, delta, r, m*2, h/2);
end
end
```

Published with MATLAB® R2024b

```
function I_T = trape(f, a, b, n)

global function_evals_trape

h = (b-a)/n;
x = linspace(a, b, n+1);
fx = f(x);

% have to evaluate function at each point in x
function_evals_trape = function_evals_trape + length(x);

I_T = h*(sum(fx(2:n))+(fx(1)+fx(n+1))/2);

end
```

Published with MATLAB® R2024b