Name: Robert Haubrich and Muhideen Ogunlowo .

Title: Integration with Adaptivity and Difficult Endpoints

Date: 17th of November, 2023

Introduction: This is a MATLAB live script that integrates two functions where each has an endpoint singularity at the left end. This live script uses adaptive integration using MATLAB's built-in function integral as well as the textbook function intadapt and the fucntion which implements Simpson's rule for numerical integration, simp2.

Methods Description:

simp2: Implements Simpson's rule for numerical integration of a function f over an interval `[a, b]` using `n` subintervals.

intadapt: Performs adaptive integration on a function `f` over an interval `[a, b]` with a specified error tolerance `tol`. It uses recursive bisection and error estimation to adaptively refine the integration and returns the integral approximation and the vector of nodes used .

`part2to4(a, b)` and `part5(a, b)`: These functions define two integrands `f1` and `f2` and perform numerical integration using the `integral`, `intadapt`, and `simp2` methods. They compare the results and errors of these methods, and output tables and plots to visualize the comparisons. The functions differ in their focus, with `part2to4` emphasizing error analysis and `part5` focusing on approximation comparisons.

The exact integral of $f_1(x) = \frac{1}{x}\sin(x^2) + 2x\log(x)\cos(x^2)$ is given as $F_1(x) = \log(x)\sin(x^2)$

To verify that the given integral is correct, we need to take the differential of $F_1(x)$ which will give us $f_1(x)$ by backward proof (rhs)

$F'(x) = \sin(x^2) * \frac{d}{\mathrm{dx}}(\log(x)) + \log(x) * \frac{d}{\mathrm{dx}}(\sin(x^2))$ (Using the product rule)

$\frac{d}{\mathrm{dx}}(\log(x)) = \frac{1}{x}$

$\frac{d}{\mathrm{dx}}(\sin(x^2)) = 2x * \cos(x^2)$

Replacing $\frac{d}{\mathrm{dx}}$ with $\frac{1}{x}$ and $\frac{d}{\mathrm{dx}}(\sin(x^2))$ with $2x * \cos(x^2)$

We get :

$F'(x) = \sin(x^2) * \frac{1}{x} + \log(x) * 2x * \cos(x^2)$

$F'(x) = f(x),$

$f_1(x) = \frac{1}{x}\sin(x^2) + 2x\log(x)\cos(x^2)$

Since the differential of the integral $F_1(x) = \log(x)\sin(x^2)$ is equal to $f_1(x) = \frac{1}{x}\sin(x^2) + 2x\log(x)\cos(x^2)$, it holds true

The exact integral of $f_2(x) = -\frac{1}{2x^{\frac{3}{2}}}\sin(x^2) + 2\sqrt{x} * \cos(x^2)$ is given as $F_2(x) = \frac{1}{\sqrt{x}}\sin(x^2)$

To verify that the given integral is correct, we need to take the differential of $F_1(x)$ which will give us $f_1(x)$ by backward proof (rhs)

$F'(x) = \sin(x^2) * \frac{d}{\mathrm{dx}}\left(\frac{1}{\sqrt{x}}\right) + \frac{1}{\sqrt{x}} * \frac{d}{\mathrm{dx}}(\sin(x^2))$ (Using the product rule)

$\frac{d}{\mathrm{dx}}\left(\frac{1}{\sqrt{x}}\right) = -\frac{1}{2x^{\frac{3}{2}}}$

$\frac{d}{\mathrm{dx}}(\sin(x^2)) = 2x * \cos(x^2)$

Replacing $\frac{d}{\mathrm{dx}}$ with $-\frac{1}{2x^{\frac{3}{2}}}$ and $\frac{d}{\mathrm{dx}}(\sin(x^2))$ with $2x * \cos(x^2)$

We get :

$F'(x) = \sin(x^2) * -\frac{1}{2x^{\frac{3}{2}}} + 2x * \cos(x^2) * \frac{1}{\sqrt{x}}$

$F'(x) = -\frac{1}{2x^{\frac{3}{2}}} * \sin(x^2) + 2\sqrt{x} * \cos(x^2)$

$F'(x) = f(x)$

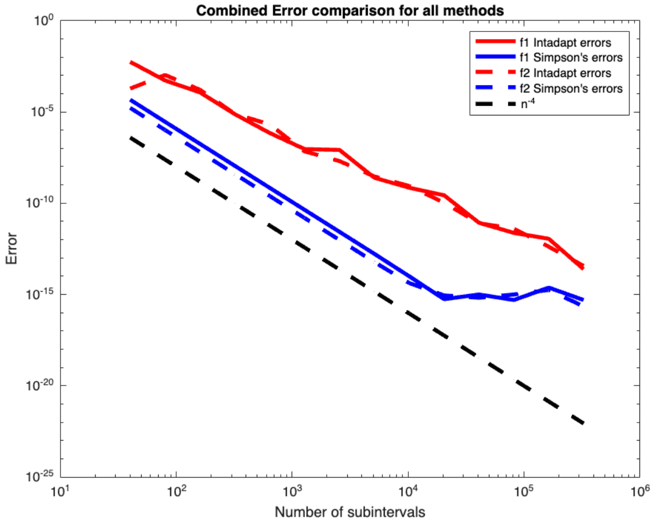
$f_2(x) = -\frac{1}{2x^{\frac{3}{2}}}\sin(x^2) + 2\sqrt{x} * \cos(x^2)$

Since the differential of the integral $F_2(x) = \frac{1}{\sqrt{x}}\sin(x^2)$ is equal to $f_2(x) = -\frac{1}{2x^{\frac{3}{2}}}\sin(x^2) + 2\sqrt{x} * \cos(x^2)$, it holds true

```
part2to4(1, 3);
```

tols	f1 integral err.	f1 intadapt err.	f1 intadapt nodes	n_s	f1 Simpson's err.
0.01	4.996e-16	0.0054496	13	40	4.6073e-05
0.001	4.996e-16	0.00052347	21	80	2.8458e-06
0.0001	4.996e-16	0.00011609	29	160	1.7734e-07
1e-05	4.996e-16	7.4644e-06	45	320	1.1076e-08

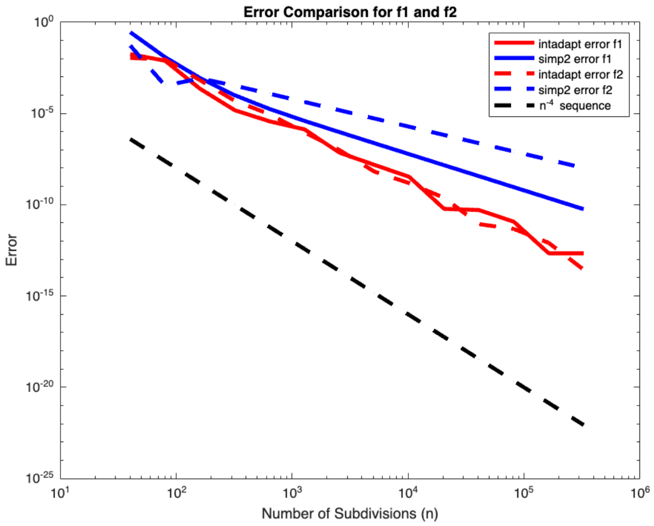
1e-06	4.996e-16	6.945e-07	81	640	6.921e-10
1e-07	4.996e-16	9.1497e-08	121	1280	4.3254e-11
1e-08	4.996e-16	8.2301e-08	201	2560	2.7029e-12
1e-09	4.996e-16	2.4684e-09	325	5120	1.6831e-13
1e-10	4.996e-16	7.0085e-10	525	10240	9.9365e-15
1e-11	4.996e-16	2.6981e-10	809	20480	5.5511e-16
1e-12	4.996e-16	8.4499e-12	1357	40960	9.992e-16
1e-13	4.996e-16	2.3586e-12	2081	81920	4.996e-16



The methods are working properly, intadapt and simp2 errors are have a similar slope in relation to the number of intervals

```
part6(eps, 2*pi);
```

tols	n_s	f1 Integral Error	f1 Intadapt Error	f1 Simpson Error	f1 Intadapt Nodes
0.01	40	2.3578e-11	0.017502	0.27831	57
0.001	80	2.3578e-11	0.0075911	0.011897	85
0.0001	160	2.3578e-11	0.00022363	0.00086061	149
1e-05	320	2.3578e-11	1.4777e-05	9.7464e-05	249
1e-06	640	2.3553e-11	3.6407e-06	1.7213e-05	413
1e-07	1280	1.4511e-12	1.3221e-06	3.8592e-06	645
1e-08	2560	1.4539e-12	6.9954e-08	9.3711e-07	1097
1e-09	5120	7.8604e-14	1.5181e-08	2.3255e-07	1729
1e-10	10240	9.3925e-14	3.3295e-09	5.8029e-08	2593
1e-11	20480	7.3275e-15	5.8858e-11	1.45e-08	4429
1e-12	40960	2.2204e-15	5.0784e-11	3.6247e-09	6957
1e-13	81920	3.7748e-15	1.1539e-11	9.0615e-10	10413



Simpson's rule performs best with smooth, well-behaved functions. If the function being integrated is not smooth (e.g., it has discontinuities like, $f_1(x)$ and $f_2(x)$, Simpson's rule might not converge well. Functions with singularities or undefined points in the interval of integration are particularly problematic.

```
%Question 5, This evaluates all functionsat b value of 2pi and a value of 0
f1 = @(x) x.^(-1) .* sin(x.^2) + 2 .* x .* log(x) .* cos(x.^2);
f2 = @(x) -1 ./ (2.*x.^(3/2)) .* sin(x.^2) + 2 .* x.^(0.5) .* cos(x.^2);

F1 = @(x) log(x) .* sin(x.^2);
F2 = @(x) x.^(-0.5) .* sin(x.^2);
```

```
% Define tolerances and number of subintervals
tols = 10.^(-2:-1:-15)';
n_s = 10*2.^(2:15)';
n_inv4 = n_s.^-4;

% Initialize arrays for results and errors
integrals = zeros(length(n_s), 2); % For integral of f1 and f2
integerr = zeros(length(n_s), 2);
intadapts = zeros(length(n_s), 4); % For intadapt of f1 and f2
intaderr = zeros(length(n_s), 4);
simpsons = zeros(length(n_s), 2); % For simp2 of f1 and f2
simp2err = zeros(length(n_s), 2);


for i = 2:1:15
    tol = 10.^(-1*i);
    n = 10*2.^(i);
    integrals(i-1,1) = integral(f1, 0, 2*pi, 'AbsTol', tol, 'RelTol', tol);
    integrals(i-1,2) = integral(f2, 0, 2*pi, 'AbsTol', tol, 'RelTol', tol);
end
Table_of_integral_values = integrals
```

```
Table_of_integral_values = 14×2
    1.7981    0.3903
    1.7981    0.3903
    1.7981    0.3903
    1.7981    0.3903
    1.7981    0.3903
    1.7981    0.3903
    1.7981    0.3903
    1.7981    0.3903
    1.7981    0.3903
    1.7981    0.3903
```

```
%Intadapt function for (a,b)= (0,2pi)
for i = 2:1:15
    tol = 10.^(-1*i);
    n = 10*2.^(i);
    [intadapts(i-1,1), t] = intadapt(f1, 0, 2*pi, tol);
    intadapts(i-1,2) = length(t);
    [intadapts(i-1,3), t] = intadapt(f2, 0, 2*pi, tol);
    intadapts(i-1,4) = length(t);
end
```

```
Out of memory. The likely cause is an infinite recursion within the program.

Error in proj2>intadapt/do_integral (line 84)
        xl = (a+m)/2;  fl = f(xl);
```

As intadapt tries to evaluate the limit as a=0, the integrand has a singularity at a=0. The function value becomes undefined or tends to infinity as x approaches zero. This can lead to difficulties in accurately estimating the integral, as the adaptive algorithm may struggle to handle the infinite or undefined values at the singularity.

```
%Simp2 function for (a,b)= (0,2pi)
for i = 2:1:15
    tol = 10.^(-1*i);
    n = 10*2.^(i);

    simpsons(i-1,1) = simp2(f1, 0, 2*pi, n);
    simpsons(i-1,2) = simp2(f2, 0, 2*pi, n);
end
Table_of_Simpson_values= simpsons
```

```
Table_of_Simpson_values = 14×2
   NaN    NaN
   NaN    NaN
   NaN    NaN
   NaN    NaN
   NaN    NaN
   NaN    NaN
   NaN    NaN
   NaN    NaN
   NaN    NaN
   NaN    NaN
```

Simpson's rule assumes that the function is well-behaved and continuous over the interval of integration, including the endpoints. A singularity violates this assumption, leading to inaccurate or undefined results.

```
table(Table_of_integral_values,Table_of_Simpson_values) %Table of simpson values and Integral values at (a,b) = (0,2pi)
```

ans = 14×2 table

	Table_of_integral_values	Table_of_Simpson_values

	Table_of_integral_values		Table_of_Simpson_values	
1	1.7981	0.3903	NaN	NaN
2	1.7981	0.3903	NaN	NaN
3	1.7981	0.3903	NaN	NaN
4	1.7981	0.3903	NaN	NaN
5	1.7981	0.3903	NaN	NaN
6	1.7981	0.3903	NaN	NaN
7	1.7981	0.3903	NaN	NaN
8	1.7981	0.3903	NaN	NaN
9	1.7981	0.3903	NaN	NaN
10	1.7981	0.3903	NaN	NaN
11	1.7981	0.3903	NaN	NaN
12	1.7981	0.3903	NaN	NaN
13	1.7981	0.3903	NaN	NaN
14	1.7981	0.3903	NaN	NaN

Contributions: Problems 2,4 and 5 were completed by Robert Haubrich, while Problems 1, 3, and 6 were completed by Muhideen Ogunlowo

Functions used at the end of file. This includes the simp2 and intadapt functions while storing functions used in answering questions 2,4 and 6

```
function Q = simp2(fun,a,b,n)
%SIMP2  Classic Simpson's rule implementation to
%       approximate integral of f from a to b with
%       n subintervals
%
% Input:  fun - function to integrate
%         a - lower limit of integration
%         b - upper limit of integration
%         n - number of grid spacings (must be even)
%
% Output: Q - approximation of integral
%
h = (b-a)/n;        % grid spacing
t = a+h*(0:n)';     % grid points
f = fun(t);         % function values on grid
S = f(1) + 4*sum(f(2:2:n)) + 2*sum(f(3:2:n-1)) + f(n+1);
Q = S*h/3;          % approximation

end

function [Q,t] = intadapt(f,a,b,tol)
%INTADAPT   Adaptive integration with error estimation.
% Input:
%   f      integrand (function)
%   a,b    interval of integration (scalars)
%   tol    acceptable error
% Output:
%   Q      approximation to integral(f,a,b)
%   t      vector of nodes used

m = (b+a)/2;
[Q,t] = do_integral(a,f(a),b,f(b),m,f(m),tol);

    % Use error estimation and recursive bisection.
    function [Q,t] = do_integral(a,fa,b,fb,m,fm,tol)

        % These are the two new nodes and their f-values.
        xl = (a+m)/2;  fl = f(xl);
        xr = (m+b)/2;  fr = f(xr);
        t = [a;xl;m;xr;b];              % all 5 nodes at this level

        % Compute the trapezoid values iteratively.
        h = (b-a);
        T(1) = h*(fa+fb)/2;
        T(2) = T(1)/2 + (h/2)*fm;
        T(3) = T(2)/2 + (h/4)*(fl+fr);

        S = (4*T(2:3)-T(1:2)) / 3;      % Simpson values
        E = (S(2)-S(1)) / 15;           % error estimate

        if abs(E) < tol*(1+abs(S(2)))   % acceptable error?
            Q = S(2);                   % yes--done
        else
            % Error is too large--bisect and recurse.
            [QL,tL] = do_integral(a,fa,m,fm,xl,fl,tol);
            [QR,tR] = do_integral(m,fm,b,fb,xr,fr,tol);
            Q = QL + QR;
            t = [tL;tR(2:end)];         % merge the nodes w/o duplicate
        end
    end
```

```
end

function null = part2to4(a, b)
    % Define the functions
    f1 = @(x) x.^(-1) .* sin(x.^2) + 2 .* x .* log(x) .* cos(x.^2);
    f2 = @(x) -1 ./ (2.*x.^(3/2)) .* sin(x.^2) + 2 .* x.^(0.5) .* cos(x.^2);
    % antiderivatives
    F1 = @(x) log(x) .* sin(x.^2);
    F2 = @(x) x.^(-0.5) .* sin(x.^2);

    % Define tolerances and number of subintervals
    tols = 10.^(-2:-1:-15)';
    n_s = 10*2.^(2:15)';
    n_inv4 = n_s.^-4;

    % Initialize arrays for results and errors
    integrals = zeros(length(n_s), 2); % For integral of f1 and f2
    integerr = zeros(length(n_s), 2);
    intadapts = zeros(length(n_s), 4); % For intadapt of f1 and f2
    intaderr = zeros(length(n_s), 4);
    simpsons = zeros(length(n_s), 2); % For simp2 of f1 and f2
    simp2err = zeros(length(n_s), 2);

    % Loop over tolerances for integral and intadapt
    for i = 2:1:15
        tol = 10.^(-1*i);
        integrals(i-1,1) = integral(f1, a, b, 'AbsTol', tol, 'RelTol', tol);
        integrals(i-1,2) = integral(f2, a, b, 'AbsTol', tol, 'RelTol', tol);
    end

    if a == 0
        a = eps;
    end
    if b == 0
        b = eps;
    end

    for i = 2:1:15
        tol = 10.^(-1*i);
        n = 10*2.^(i);

        % integral for f1 and f2 with b = 2pi, a = 0
        integerr(i-1,1) = abs(integrals(i-1,1)-(F1(b)-F1(a)));
        integerr(i-1,2) = abs(integrals(i-1,2)-(F2(b)-F2(a)));

        % intadapt for f1 and f2 with b = 2pi, a = eps
        [intadapts(i-1,1), t] = intadapt(f1, a, b, tol);
        intadapts(i-1,2) = length(t);
        [intadapts(i-1,3), t] = intadapt(f2, a, b, tol);
        intadapts(i-1,4) = length(t);
        intaderr(i-1,1) = abs(intadapts(i-1,1)-(F1(b)-F1(a)));
        intaderr(i-1,2) = abs(intadapts(i-1,3)-(F2(b)-F2(a)));

        simpsons(i-1,1) = simp2(f1, a, b, n);
        simpsons(i-1,2) = simp2(f2, a, b, n);
        simp2err(i-1,1) = abs(simpsons(i-1,1)-(F1(b)-F1(a)));
        simp2err(i-1,2) = abs(simpsons(i-1,2)-(F2(b)-F2(a)));
    end

    % Tabulate errors and evaluations for f1
    T_f1 = table(tols, integerr(:,1), intaderr(:,1), intadapts(:,2), n_s, simp2err(:,1));
    T_f1 = renamevars(T_f1,["Var2", "Var3", "Var4", "Var6"],["f1 integral err.", "f1 intadapt err.", "f1 intadapt nodes",
    disp(T_f1);

    % Tabulate errors and evaluations for f2
    T_f2 = table(tols, integerr(:,2), intaderr(:,2), intadapts(:,4), n_s, simp2err(:,2));
    T_f2 = renamevars(T_f2,["Var2", "Var3", "Var4", "Var6"],["f2 integral err.", "f2 intadapt err.", "f2 intadapt nodes",
    disp(T_f2);

    figure;

    % Plot errors for f1
    loglog(n_s, intaderr(:,1), 'r', 'linewidth', 3); hold on;
    loglog(n_s, simp2err(:,1), 'b', 'linewidth', 3);

    % Plot errors for f2
    loglog(n_s, intaderr(:,2), 'r--', 'linewidth', 3);
    loglog(n_s, simp2err(:,2), 'b--', 'linewidth', 3);

    % Plot n^{-4} line
    loglog(n_s, n_inv4, 'k--', 'linewidth', 3);

    % Add legend
    legend('f1 Intadapt errors', "f1 Simpson's errors", 'f2 Intadapt errors', "f2 Simpson's errors", 'n^{-4}');
```

```
    % Add labels and title
    xlabel('Number of subintervals');
    ylabel('Error');
    title('Combined Error comparison for all methods');

    % Release the hold
    hold off;
end

function null = part6(a, b)
    % Define the functions
    f1 = @(x) x.^(-1) .* sin(x.^2) + 2 .* x .* log(x) .* cos(x.^2);
    f2 = @(x) -1 ./ (2.*x.^(3/2)) .* sin(x.^2) + 2 .* x.^(0.5) .* cos(x.^2);

    % Define tolerances and number of subintervals
    tols = 10.^(-2:-1:-15)';
    n_s = 10*2.^(2:15)';

    % Initialize arrays for results, errors, and function evaluations
    integrals = zeros(length(n_s), 2); % For integral of f1 and f2
    intadapts = zeros(length(n_s), 4); % For intadapt of f1 and f2
    simpsons = zeros(length(n_s), 2); % For simp2 of f1 and f2
    errors = zeros(length(n_s), 6); % For errors in each method

    % Compute the reference values with the smallest tolerance
    ref_val_f1 = integral(f1, a, b, 'AbsTol', min(tols), 'RelTol', min(tols));
    ref_val_f2 = integral(f2, a, b, 'AbsTol', min(tols), 'RelTol', min(tols));

    % Loop over tolerances for integral and intadapt
    for i = 2:1:15
        tol = 10.^(-1*i);
        n = 10*2.^(i);
        % Calculate approximations
        %for integral, the fucntion will take an a value of 0 and b value of
        %2pi
        integrals(i-1,1) = integral(f1, 0, 2*pi, 'AbsTol', tol, 'RelTol', tol);
        integrals(i-1,2) = integral(f2, 0, 2*pi, 'AbsTol', tol, 'RelTol', tol);
        %intadapt and simp2 evaluations
        [intadapts(i-1,1), t] = intadapt(f1, a, b, tol);
        intadapts(i-1,2) = length(t);
        [intadapts(i-1,3), t] = intadapt(f2, a, b, tol);
        intadapts(i-1,4) = length(t);
        simpsons(i-1,1) = simp2(f1, a, b, n);
        simpsons(i-1,2) = simp2(f2, a, b, n);

        % Calculate errors
        errors(i-1,1) = abs(integrals(i-1,1) - ref_val_f1); % Error for integral f1
        errors(i-1,2) = abs(intadapts(i-1,1) - ref_val_f1); % Error for intadapt f1
        errors(i-1,3) = abs(simpsons(i-1,1) - ref_val_f1); % Error for simp2 f1
        errors(i-1,4) = abs(integrals(i-1,2) - ref_val_f2); % Error for integral f2
        errors(i-1,5) = abs(intadapts(i-1,3) - ref_val_f2); % Error for intadapt f2
        errors(i-1,6) = abs(simpsons(i-1,2) - ref_val_f2); % Error for simp2 f2
    end

  % Create table with errors and intadapt nodes for f1
     T_f1 = table(tols, n_s, errors(:,1), errors(:,2), errors(:,3), intadapts(:,2));
     T_f1 = renamevars(T_f1, ["Var3", "Var4", "Var5", "Var6"], ...
                  ["f1 Integral Error", "f1 Intadapt Error", "f1 Simpson Error", "f1 Intadapt Nodes"]);
     disp(T_f1);

  % Create table with errors and intadapt nodes for f2
     T_f2 = table(tols, n_s, errors(:,4), errors(:,5), errors(:,6), intadapts(:,4));
     T_f2 = renamevars(T_f2, ["Var3", "Var4", "Var5", "Var6"], ...
                  ["f2 Integral Error", "f2 Intadapt Error", "f2 Simpson Error", "f2 Intadapt Nodes"]);
     disp(T_f2);

    % Compute n^(-4) sequence
    n4_sequence = (n_s .^ (-4));

    % Combined plot for errors of intadapt and simp2 for both f1 and f2
    figure;
    loglog(n_s, errors(:,2), 'r', ... % intadapt error for f1
       n_s, errors(:,3), 'b', ... % simp2 error for f1
       n_s, errors(:,5), 'r--', ... % intadapt error for f2
       n_s, errors(:,6), 'b--', ... % simp2 error for f2
       n_s, n4_sequence, 'k--', 'linewidth',3);     % n^(-4) sequence
    xlabel('Number of Subdivisions (n)');
    ylabel('Error');
    title('Error Comparison for f1 and f2');
    legend('intadapt error f1', 'simp2 error f1', 'intadapt error f2', 'simp2 error f2', 'n^{-4} sequence', 'Location', 'n
    grid off;
end
```