1 Introduction

When integrating functions of one or more variables, analytic solutions are desired because they offer a complete description of solution behavior. For most situations of engineering interest, however, the governing equations defy all attempts to obtain a closed solution. In such instances, numerical integration provides a tool for exploring the solution space.

We solve the following problems to better understand the tools of numerical integration. As will be described in the methods section, our tools include the trapezoidal method, Simpson's method, and Richardson extrapolation.

1.1 Problem A

We wish to show that the equality

$$\int_0^{2\pi} \frac{dx}{1+a\cos x} = \frac{2\pi}{\sqrt{1-a^2}}, \qquad a^2 < 1 \tag{1}$$

holds for $a = \frac{1}{2}$. This is a relatively simple finite integral of a continuous function with no singularities.

1.2 Problem B

We wish to confirm the equalities

$$\text{(i)} \quad \int_0^{\infty} \frac{\sin^4 x}{x^4}\, dx = \frac{\pi}{3}, \qquad \text{and} \qquad \text{(ii)} \quad \int_0^{\infty} \frac{\sin x}{\sqrt{x}}\, dx = \sqrt{\frac{\pi}{2}}. \tag{2}$$

Note that both integrands are indeterminate at $x = 0$ and otherwise periodic, and that both integrals extend indefinitely. Particular care will need to be paid to singularities and periodicity.

1.3 Problem C

Moving to functions of two variables, we wish to estimate

$$I = \int_1^3 \int_1^2 f(x,y)\, dx\, dy \tag{3}$$

for the integrands

$$\text{(i)} \quad f(x,y) = xy(1+x), \qquad \text{and} \qquad \text{(ii)} \quad f(x,y) = x^2y^3(1+x). \tag{4}$$

As in the first problem, these expressions involve well-behaved integrands and finite bounds. Application of our chosen numerical method over a two-dimensional domain will be the crux of this problem.

2 Methodology

Versions of the trapezoidal method, Simpson's method, and Richardson extrapolation are all implemented in MATLAB. The following problems employ a subset of these methods in their solution.

The value of π used for calculations is always obtained through the equality $\pi = 4\arctan(1)$.

2.1 *Problem A*

To solve this problem, we use Simpson's method with $N_1 = \{10, 20\}$ and $N_2 = \{100, 200\}$ equally-spaced parts. That is, we split the interval $x \in [0, 2\pi]$ with equally spaced points $x_i, i \in [0, N]$ such that $x_0 = 0$ and $x_N = 2\pi$. In all cases, we calculate the relative error as

$$e_r = \frac{|f_e - f_c|}{f_e}, \tag{5}$$

where f_e is the exact value of the integral and f_c is the computed value. We then apply Richardson extrapolation to the sets N_1 and N_2.

3 Results

4 Discussion

5 References

Appendix: MATLAB Code

The following code listings generate all figures presented in this homework assignment.

Listing 1: Homework_1_Driver.m

```
function [] = Homework_1_Driver()
%%%
% HOMEWORK 1 DRIVER
%
% Run this file to calculate output for Homework 1.
%%%

Set_Default_Plot_Properties();

Blanks();

Problem('A');
Problem_A();
Blanks();

Problem('B');
Problem_B();
Blanks();

Problem('C');
Problem_C();
Blanks();

end

function [] = Problem( name )
    fprintf('************************************************************\n');
    fprintf('**********>>> Problem %s <<<**********\n', name);
    fprintf('************************************************************\n');
end

function [] = Blanks()
    fprintf('\n\n');
end
```

Listing 2: RelErr.m

```
function [ re ] = RelErr( fe, fc )
```

```
    re = abs(fe - fc) ./ fe;
end
```

Listing 3: Problem_A.m

```
function [] = Problem_A()

    % Define the integrand with a function handle.
    a = 1/2;
    f = @(x) 1 / (1 + a*cos(x));

    % Calculate pi, but name it pii since MATLAB has already defined the variable pi.
    pii = 4 * atan(1);
    fprintf('pi:  %28.20e\n', pii);

    Dashes();

    % Calculate the RHS.
    rhs = 2 * pii / sqrt(1 - a^2);
    fprintf('RHS: %28.20e\n', rhs);

    Dashes();

    % Calculate the LHS with n = {10, 100}. Note: this corresponds to {5, 50} large slabs
    % over which Simpson's method operates (dividing each slab into two intervals).

    v10  = Simpsons(f, [0,2*pii], 5);
    v100 = Simpsons(f, [0,2*pii], 50);

    fprintf('LHS: %28.20e with n=10\n',  v10);
    fprintf('LHS: %28.20e with n=100\n', v100);

    re10  = RelErr(v10,  rhs);
    re100 = RelErr(v100, rhs);
    fprintf('re:  %28.20e with n=10\n',  re10);
    fprintf('re:  %28.20e with n=100\n', re100);

    Dashes();

    % Double the grid points, and re-calculate the LHS.

    v20  = Simpsons(f, [0,2*pii], 10);
    v200 = Simpsons(f, [0,2*pii], 100);

    fprintf('LHS: %28.20e with n=20\n',  v20);
    fprintf('LHS: %28.20e with n=200\n', v200);

    re20  = RelErr(v20,  rhs);
    re200 = RelErr(v200, rhs);
    fprintf('re:  %28.20e with n=20\n',  re20);
    fprintf('re:  %28.20e with n=200\n', re200);

    Dashes();

    % Use Richardson extrapolation on each set.

    ve1 = v20  + (v20  - v10 ) / 15;
    ve2 = v200 + (v200 - v100) / 15;
    fprintf('LHS: %28.20e with n={10, 20}  (Richardson)\n', ve1);
    fprintf('LHS: %28.20e with n={100,200} (Richardson)\n', ve2);

    reR1 = RelErr(ve1, rhs);
    reR2 = RelErr(ve2, rhs);
    fprintf('re:  %28.20e with n={10, 20}  (Richardson)\n', reR1);
    fprintf('re:  %28.20e with n={100,200} (Richardson)\n', reR2);

end
```

Listing 4: Simpsons.m

```
function [ F ] = Simpsons( f, bounds, n_slabs )

    n_pts = 1 + 2 * n_slabs;
    x = linspace(bounds(1), bounds(2), n_pts);
    h = x(2) - x(1);

    y = zeros(1,n_pts);
    for i = 1:n_pts
        y(i) = feval(f, x(i));
    end

    F = 0;
    for i = 1:n_slabs
        % Determine indices involved in Simpson's rule for this slab.
        ii = 1 + 2*(i-1)*ones(1,3) + [0,1,2];
        F = F + y(ii(1)) + 4 * y(ii(2)) + y(ii(3));
    end
    F = F * h/3;

end
```

Listing 5: Problem_B.m

```
function [] = Problem_B()

    % Calculate pi, but name it pii since MATLAB has already defined the variable pi.
    pii = 4 * atan(1);

    %%%%%%%%%%%%
    % Part (i) %
    %%%%%%%%%%%%

    % Calculate the RHS.
    rhs = pii / 3;
    fprintf('RHS: %28.20e\n', rhs);

    % Define the integrand with a function handle.
    f = @(x) sin(x)^4 / x^4;

    figure();
    x = linspace(0,25);
    y = zeros(0,length(x));
    for i = 1:length(x)
        y(i) = f(x(i));
    end
    plot(x,y);
    xlabel('x');
    ylabel('y');
    legend('sin(x)^4 / x^4');

    min_x   = 0;
    slabspp = 4;
    y0      = 1;

    pers = (1:9)' * 10.^(0:3);
    pers = reshape(pers, 1, numel(pers));
    F1   = zeros(1, length(pers));
    F2   = zeros(1, length(pers));
    Fe   = zeros(1, length(pers));

    for i = 1:length(pers)

        per = pers(i);

        F1(i) = Trapezoidal_Inf(f, min_x, slabspp,   2*pii, per, y0);
        F2(i) = Trapezoidal_Inf(f, min_x, slabspp*2, 2*pii, per, y0);
        Fe(i) = F2(i) + (F2(i) - F1(i)) / 3;
```

```
45
46      end
47
48      re1 = RelErr(F1, rhs);
49      re2 = RelErr(F2, rhs);
50      ree = RelErr(Fe, rhs);
51
52      figure();
53      loglog(pers, re1, pers, re2, pers, ree);
54      xlim([min(pers),max(pers)]);
55      xlabel('Periods Integrated');
56      ylabel('Relative Error');
57      legend('4 Slabs / Period', '8 Slabs / Period', 'Richardson Extrapolation');
58
59      %%%%%%%%%%%%%
60      % Part (ii) %
61      %%%%%%%%%%%%%
62
63      % Calculate the RHS.
64      rhs = sqrt(pii / 2);
65      fprintf('RHS: %28.20e\n', rhs);
66
67      % Define the integrand with a function handle.
68      f = @(x) sin(x) / sqrt(x);
69
70      figure();
71      x = linspace(0,500,5000);
72      y = zeros(0,length(x));
73      for i = 1:length(x)
74          y(i) = f(x(i));
75      end
76      plot(x,y);
77      xlabel('x');
78      ylabel('y');
79      legend('sin(x) / sqrt(x)');
80
81      min_x   = 0;
82      y0      = 1;
83
84      % Do slabs/period = {4, 8}.
85
86      slabspp = 4;
87
88      pers = (1:9)' * 10.^(0:2);
89      pers = reshape(pers, 1, numel(pers));
90      F1   = zeros(1, length(pers));
91      F2   = zeros(1, length(pers));
92      Fe   = zeros(1, length(pers));
93
94      for i = 1:length(pers)
95
96          per = pers(i);
97
98          F1(i) = Trapezoidal_Inf(f, min_x, slabspp,   2*pii, per, y0);
99          F2(i) = Trapezoidal_Inf(f, min_x, slabspp*2, 2*pii, per, y0);
100         Fe(i) = F2(i) + (F2(i) - F1(i)) / 3;
101
102     end
103
104     re1 = RelErr(F1, rhs);
105     re2 = RelErr(F2, rhs);
106     ree = RelErr(Fe, rhs);
107
108     figure();
109     loglog(pers, re1, pers, re2, pers, ree);
110     xlim([min(pers),max(pers)]);
111     xlabel('Periods Integrated');
112     ylabel('Relative Error');
113     legend('4 Slabs / Period', '8 Slabs / Period', 'Richardson Extrapolation');
```

```
114
115     % Do slabs/period = {22, 44}.
116
117     slabspp = 22;
118
119     pers = (1:9)' * 10.^(0:2);
120     pers = reshape(pers, 1, numel(pers));
121     pers = 1:150;
122     F1   = zeros(1, length(pers));
123     F2   = zeros(1, length(pers));
124     Fe   = zeros(1, length(pers));
125
126     for i = 1:length(pers)
127
128         per = pers(i);
129
130         F1(i) = Trapezoidal_Inf(f, min_x, slabspp,   2*pii, per, y0);
131         F2(i) = Trapezoidal_Inf(f, min_x, slabspp*2, 2*pii, per, y0);
132         Fe(i) = F2(i) + (F2(i) - F1(i)) / 3;
133
134     end
135
136     re1 = RelErr(F1, rhs);
137     re2 = RelErr(F2, rhs);
138     ree = RelErr(Fe, rhs);
139
140     figure();
141     loglog(pers, re1, pers, re2, pers, ree);
142     xlim([min(pers),max(pers)]);
143     xlabel('Periods Integrated');
144     ylabel('Relative Error');
145     legend('28 Slabs / Period', '56 Slabs / Period', 'Richardson Extrapolation');
146
147 end
```

Listing 6: Trapezoidal_Inf.m

```
1  function [ F ] = Trapezoidal_Inf( f, min_x, n_slabpp, T, n_periods, y0 )
2
3  %%%
4  %
5  % Trapezoidal integration of a function handle f, assuming ...
6  %   - indefinite right bound, and
7  %   - function has some period T, and
8  %   - zero-value is undefined
9  %
10 % Note: n_slabpp is number of slabs per period.
11 %
12 % If the most recently calculated period's integral F_T divided by the total integral
13 %   F + F_T,  is below tolerance, integration stops.
14 %
15 % If zero_value is specified, y(x==0) will be set to zero_value analytically. This is
16 %   intended to circumvent divide-by-zero errors at x==0 by passing the analytic value of
17 %   the limit to this function.
18 %
19 %%%
20
21     % Number of points per period for the trapezoidal rule.
22     n_ppp = n_slabpp + 1;
23
24     % Index of the period we are evaluating.
25     period_i = 0;
26
27     % Value of the integral.
28     F = 0;
29
30     while period_i < n_periods
31
```

```
        % This period's contribution to the integral.
        F_T = 0;

        a = min_x +  period_i      * T;
        b = min_x + (period_i + 1) * T;
        x = linspace(a, b, n_ppp);

        h = x(2) - x(1);

        y = zeros(1, n_ppp);
        for i = 1:n_ppp
            if x(i) == 0
                y(i) = y0;
            else
                y(i) = feval(f, x(i));
            end
        end

        % Sum each slab's contribution to this period's integral.
        for i = 1:n_slabpp
            slab = (h/2) * ( y(i) + y(i+1) );
            F_T = F_T + slab;
        end

        % Update total integral.
        F = F + F_T;

        % Move to the next period.
        period_i = period_i + 1;

    end

end
```

Listing 7: Problem_C.m

```
function [] = Problem_C()

    % Calculate pi, but name it pii since MATLAB has already defined the variable pi.
    pii = 4 * atan(1);

    %%%%%%%%%%%%
    % Part (i) %
    %%%%%%%%%%%%

    rhs = 46/3;

    f = @(x,y) x * y * (1+x);

    F = Simpsons_2D(f, [1,2], [1,3], 1, 1);

    re = RelErr(F, rhs);
    fprintf('re:  %28.20e (Part i )\n',  re);

    %%%%%%%%%%%%
    % Part (ii) %
    %%%%%%%%%%%%

    rhs = 365/3;

    f = @(x,y) x^2 * y^3 * (1+x);

    F = Simpsons_2D(f, [1,2], [1,3], 1, 1);

    re = RelErr(F, rhs);
    fprintf('re:  %28.20e (Part ii)\n',  re);

end
```

Listing 8: Simpsons_2D.m

```
function [ F ] = Simpsons_2D( f, bounds_x, bounds_y, n_slabs_x, n_slabs_y )

    n_pts_x = 1 + 2 * n_slabs_x;
    n_pts_y = 1 + 2 * n_slabs_y;
    x = linspace(bounds_x(1), bounds_x(2), n_pts_x);
    y = linspace(bounds_y(1), bounds_y(2), n_pts_y);
    hx = x(2) - x(1);
    hy = y(2) - y(1);

    z = zeros(n_pts_x, n_pts_y);
    for i = 1:n_pts_x
    for j = 1:n_pts_y
        z(i,j) = feval(f, x(i), y(j));
    end
    end

    % Weights to use for each coordinate within the slab.
    simpson_square = [1,  4, 1; ...
                      4, 16, 4; ...
                      1,  4, 1];

    F = 0;
    for i = 1:n_slabs_x
    for j = 1:n_slabs_y
        % Determine indices involved in Simpson's rule for this slab.
        ii = 1 + 2*(i-1);
        jj = 1 + 2*(j-1);
        % Calculate volume of slab.
        slab = sum(sum( simpson_square .* z(ii:ii+2, jj:jj+2) ));
        F = F + slab;
    end
    end
    F = F * (hx * hy / 9);

end
```

Listing 9: Set_Default_Plot_Properties.m

```
function [] = Set_Default_Plot_Properties()

    set(groot, 'DefaultTextInterpreter', 'none', ...
               'DefaultLegendInterpreter', 'none', ...
               'DefaultLineLineWidth', 1.5, ...
               'DefaultAxesLineWidth', 1.5, ...
               'DefaultAxesBox', 'on', ...
               'DefaultAxesXGrid', 'on', ...
               'DefaultAxesYGrid', 'on', ...
               'DefaultAxesZGrid', 'on', ...
               'DefaultAxesGridColor', [0.5,0.5,0.5], ...
               'DefaultAxesFontName', 'Helvetica', ...
               'DefaultAxesFontSize', 12, ...
               'DefaultAxesTitleFontSizeMultiplier', 1.2, ...
               'DefaultAxesLabelFontSizeMultiplier', 1.2 ...
        );

end
```

Listing 10: Dashes.m

```
function [] = Dashes()
    fprintf('------------------------------------------------------------\n');
end
```