# Homework 1

ASEN 5417: Numerical Methods

Ryan Skinner

Due 2015/09/08

## 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, periodicity, and limits of integration.

### 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

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

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 $\pi$ used for calculations is always obtained through the equality $\pi = 4\,\mathtt{arctan}(1)$.

### *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 $N_2$, to see if it improves our estimate $f_c$.

### *2.2 Problem B*

Recall that three topics must be addressed in this problem:

1. Singular integrand at $x = 0$
2. Periodicity
3. Indefinite limit of integration

To address the singularities at $x = 0$, we Taylor-expand $\sin x$ about $x = 0$, to arrive at $\sin x \approx x + \mathcal{O}(x^2)$. Keeping only first-order terms in the limit as $x \to 0$, we substitute $\sin x \approx x$ into the integrands in (2). Thus the respective integrands are equal to 1 and 0 as $x \to 0$. By using these limiting values when evaluating the integrands at $x = 0$, singularities are addressed.

Periodicity is coupled with choosing a finite limit for the indefinite integral. Both are addressed as follows.

For (2)(i), periodicity is not very important, since we know $\sin^4 x$ is bounded by $[0, 1]$, and the integrand drops off in magnitude with $x^4$. Nonetheless, we truncate the indefinite integral at $x = 8\pi$, where the last period has a maximum at $x = 7.5\pi$ of $\sim 3.2 \times 10^{-6}$ (nearly 6 orders of magnitude lower than the first maximum at $x = \pi/2$).

For (1)(ii), we note that $\sin x$ has a range of $[-1, 1]$ over each period. It is thus advantageous to numerical accuracy to truncate our integration at some multiple of $2\pi$. If the similar-magnitude positive and negative parts of each period are both accounted for, any "incomplete" period at the end of our interval could sway the resulting integral disproportionately. After some experimentation, the maximum bound on integration was set to $600\pi$. The bound is so far out because the integrand decays very slowly, proportional to $x^{1/2}$.

### *2.3 Problem C*

For these integrals, we perform integration once in the $x$-direction and once in the $y$-direction. We use $h = dx = \frac{1}{2}$, and $k = dy = 1$. This means we have nine grid points in our domain, and use Simpson's method on one slab of width $2h$ and $2k$ in each direction, respectively. Since Simpson's method assigns the weights $[1, 4, 1]$ to successive points in the slab containing $[x_i, x_{i+1}, x_{i+2}]$, the weights of the two-dimensional points are simply

$$S_{ij} = \begin{bmatrix} 1 \\ 4 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 1 \\ 4 & 16 & 4 \\ 1 & 4 & 1 \end{bmatrix}. \tag{6}$$

Assigning the weight $S_{ij}$ to the value of $f(x_i, y_j)$ within the slab should do the trick. That is, for each slab (consisting of two intervals in each dimension), the contribution of that two-dimensional slab to the integral is

$$T_{\text{slab}} = \sum_{i=1}^{3} \sum_{j=1}^{3} S_{ij} f(x_i, y_j), \tag{7}$$

where the points indexed by $i$ and $j$ lie within the slab in question. This strategy is applied to the two integrals in this problem.

## 3 Results

### 3.1 *Problem A*

Results for Simpson's method and Richardson extrapolation are shown in Table 1.

| Method | Result | Error |
|---|---|---|
| Exact | 7.25519745693687134747e+00 | 0 |
| Trapezoidal ($n = 10$) | 7.26190582552281771456e+00 | 9.2377e-04 |
| Trapezoidal ($n = 100$) | 7.25519745693687756471e+00 | 8.5693e-16 |
| Trapezoidal ($n = 20$) | 7.25518822952264841319e+00 | 1.2718e-06 |
| Trapezoidal ($n = 200$) | 7.25519745693686868293e+00 | 3.6725e-16 |
| Richardson ($n = \{10, 20\}$) | 7.25474038978930391153e+00 | 6.3002e-05 |
| Richardson ($n = \{100, 200\}$) | 7.25519745693686779475e+00 | 4.8967e-16 |

**Table 1:** Problem A computed using various methods of integration.

### 3.2 *Problem B*

Results for Problem B are presented in Figures 1 and 2.

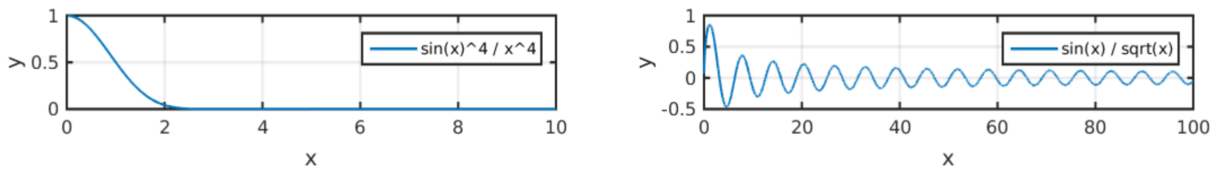

**Figure 1:** Integrands for Problem B, parts (i) and (ii). Note the difference in how quickly the oscillations decay.

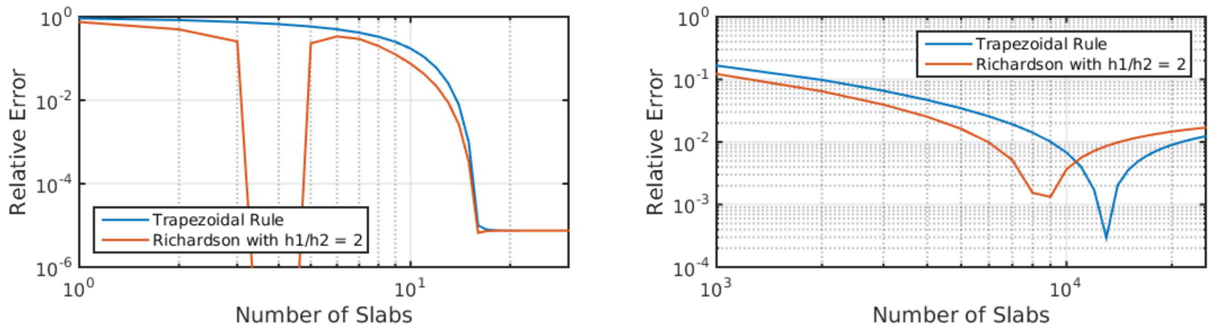

**Figure 2:** Relative errors for Problem B, parts (i) and (ii), respectively.

### *3.3 Problem C*

Results for Simpson's method applied to a two-dimensional integral are shown in Table 2.

| Part | Exact | Calculated | Error |
|---|---|---|---|
| (i) | 46/3 | 15.333333333333332 | 1.15849359091320679264e-16 |
| (ii) | 365/3 | 1.216666666666667e+02 | 1.16801545604400043792e-16 |

**Table 2:** Volume integration results for Problem C. Exact values are calculated by hand.

## 4 Discussion

### *4.1 Problem A*

In problem A, we see that Simpson's method performs well on the given integrand with as few as $n = 10$ slabs used. Doubling the number of slabs reduces the error in both cases, for $n = 10$ and $n = 100$ slabs.

Note that for $n = \{100, 200\}$, the relative error does not change past $\mathcal{O}(10^{-16})$, because machine precision has been reached. Thus, is is no surprise that Richardson extrapolation fails to improve the $n = \{100, 200\}$ estimate of the integral.

It is more surprising that Richardson extrapolation fails to improve upon the $n = 20$ data when combined with the $n = 10$ data. Its relative error is higher than that for $n = 20$, suggesting that Richardson extrapolation is not always guaranteed to work. I posit that this has to do with the concavity of the function we are integrating. It looks like a bell curve, essentially, and it would seem that depending on the number of slabs used, the extrapolation method could overshoot, especially since it has so few slabs in this instance.

### *4.2 Problem B*

In problem B, the integration scheme works perfectly for equation (2)(i). As shown in Figure 2, the relative error of the trapezoidal method drops until the integration limit prevents any further increase in accuracy beyond $\sim 10^{-4}$. Richardson extrapolation outperforms the trapezoidal rule in this case as well. It takes slightly under 30 slabs to reduce the error below $10^{-4}$ in this scenario.

Integration is much more difficult for equation (2)(ii), due to the slow rate of convergence as $x \to \infty$. Even though 300 periods were used for the domain of integration, and up to 20,000 integration points, we are still unable to attain a relative error under $10^{-4}$. Richardson extrapolation again is superior to the trapezoidal rule. This emphasizes the fact that some functions are easier to integrate than others.

### 4.3 *Problem C*

As shown in Table 2, both integrals were calculated exactly using Simpson's rule with a single two-dimensional slab of dimension $2h \times 2k$ (nine grid points). This is likely due to the face that both integrands are polynomial functions of order no higher than 3 in either variable. Since Simpson's method has a global accuracy of $\mathcal{O}(10^4)$, it is able to represent polynomials of degree 3 and lower exactly. Thus, it is able to use the minimum possible number of grid points to exactly calculate the integrals provided.

## 5 References

No external references were used other than the course notes for this assignment.

## Appendix: MATLAB Code

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

**Listing 1:** Homework_1_Driver.m

```matlab
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()
```

```matlab
    fprintf('\n\n');
end
```

**Listing 2:** RelErr.m

```matlab
function [ re ] = RelErr( fe, fc )
    re = abs(fe - fc) ./ fe;
end
```

**Listing 3:** Problem_A.m

```matlab
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.
    % Note: this corresponds to {5, 50} large slabs over which Simpson's method operates
    %   (dividing each slab into two intervals).

    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(0,10,500);
    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;
    max_x   = 8*pi;
    y0      = 1;

    n = 1:30;
    F1   = zeros(1, length(n));
    F2   = zeros(1, length(n));
    Fe   = zeros(1, length(n));

    for i = 1:length(n)

        n_current = n(i);

        F1(i) = Trapezoidal_Inf(f, min_x, max_x, n_current,   y0);
        F2(i) = Trapezoidal_Inf(f, min_x, max_x, n_current*2, y0);

    end

    Fe = F2 + (F2 - F1) / 3;

    re1 = RelErr(F1, rhs);
    re2 = RelErr(F2, rhs);
    ree = RelErr(Fe, rhs);

    figure();
    loglog(n, re1, n, ree);
    xlim([min(n),max(n)]);
    ylim([1e-6, 1]);
    xlabel('Number of Slabs');
    ylabel('Relative Error');
    legend('Trapezoidal Rule', 'Richardson with h1/h2 = 2', 'Location', 'southwest');

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

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

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

    figure();
    x = linspace(0,100,20000);
    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) / sqrt(x)');

    min_x   = 0;
    max_x   = 300*pi;
    y0      = 1;

    n = 1000:1000:25000;
    F1   = zeros(1, length(n));
    F2   = zeros(1, length(n));

    for i = 1:length(n)

        n_current = n(i);
        disp(n_current);

        F1(i) = Trapezoidal_Inf(f, min_x, max_x, n_current,   y0);
        F2(i) = Trapezoidal_Inf(f, min_x, max_x, n_current*2, y0);

    end

    Fe = F2 - (F2 - F1) / 3;

    re1 = RelErr(F1, rhs);
    re2 = RelErr(F2, rhs);
    ree = RelErr(Fe, rhs);

    figure();
    loglog(n, re1, n, ree);
    xlim([min(n),max(n)]);
    xlabel('Number of Slabs');
    ylabel('Relative Error');
    legend('Trapezoidal Rule', 'Richardson with h1/h2 = 2', 'Location', 'northeast');

end
```

**Listing 6:** Trapezoidal_Inf.m

```
function [ F ] = Trapezoidal_Inf( f, min_x, max_x, n, y0 )

%%%
%
% Trapezoidal integration of a function handle f, assuming ...
%   - indefinite right bound, but max_x is intelligently determined by user
%   - zero-value is undefined
%
% If the zero value is specified, y(x==0) will be set to zero_value analytically. This is
%   intended to circumvent divide-by-zero errors at x==0 by passing the analytic value of
%   the limit to this function.
%
%%%

    % Value of the integral.
    F = 0;

    x = linspace(min_x, max_x, n+1);

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

    y = zeros(1, n+1);
    for i = 1:n
        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
        slab = (h/2) * ( y(i) + y(i+1) );
        F = F + slab;
    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) %
    %%%%%%%%%%%%

```

```matlab
    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

```matlab
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

```matlab
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
```