Madrigal Weersink
weersinm
400244666

1. a)

$$\int_a^b f(x)dx = \frac{h}{2}[f(a)+f(b)] + h\sum_{i=1}^{n-1} f(t_i)$$

$$\int_a^b f(x)dx = \sum_{i=1}^{n} \frac{h}{2}[f(t_{i-1})+f(t_i)] - \sum_{i=1}^{n} \frac{f''(\eta_i)}{12}h^3$$

$$\min_{x\epsilon[a,b]} f''(x) \leqslant \frac{1}{n}\sum_{i=1}^{n} f''(\eta_i) \leqslant \max_{x\epsilon[a,b]} f''(x)$$

$$E = -\sum_{i=1}^{n} \frac{f''(\eta_i)h^3}{12} = -\frac{1}{n}\sum_{i=1}^{n} \frac{f''(\eta_i)}{12} \cdot n \cdot h \cdot h^2 = -\frac{f''(\mu)(b-a)h^2}{12} = -\frac{f''(\mu)(b-a)^3}{12n^2}$$

$$E = -\frac{f''(\mu)\left(\frac{\pi}{2}-0\right)^3}{12n^2} = -\frac{f''(\mu)\cdot\frac{\pi^3}{8}}{12n^2} = -\frac{f''(\mu)\cdot\pi^3}{96n^2}$$

b)

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 2\sum_{i=1}^{\frac{n}{2}-1} f(t_{2i}) + 4\sum_{i=1}^{\frac{n}{2}} f(t_{2i-1}) + f(b)\right]$$

$$E = -\frac{f^{(4)}(\zeta)}{180}(b-a)h^4 = -\frac{f^{(4)}(\zeta)(b-a)^5}{180n^4}$$

$$E = -\frac{f^{(4)}(\zeta)\left(\frac{\pi}{2}-0\right)^5}{180n^4} = -\frac{f^{(4)}(\zeta)\cdot\frac{\pi^5}{32}}{180n^4} = -\frac{f^{(4)}(\zeta)\cdot\pi^5}{5760n^4}$$

c)

$$\int_a^b f(x)dx \approx h\sum_{i=1}^{n} f\left(a + \left(i-\frac{1}{2}\right)h\right)$$

$$E = \frac{f''(\xi)}{24}(b-a)h^2 = \frac{f''(\xi)(b-a)^3}{24n^2} = \frac{f''(\xi)\left(\frac{\pi}{2}-0\right)^3}{24n^2} = \frac{f''(\xi)\cdot\frac{\pi^3}{8}}{24n^2} = \frac{f''(\xi)\cdot\pi^3}{192n^2}$$

errors.m

```
function errors(tol)
    f2 = @(x) exp(2*x).*(3*sin(x)+4*cos(x));
    f4 = @(x) exp(2*x).*(-7*sin(x)+24*cos(x));
    f2_max = max(f2(0:pi/2));
    f4_max = max(f4(0:pi/2));

    n_trap = 1;
    n_simp = 1;
    n_mid = 1;

    err_trap = inf;
    err_simp = inf;
    err_mid = inf;

    while (err_trap > tol)
        err_trap = abs(f2_max*pi^3/(96*n_trap^2));
        n_trap = n_trap + 1;
    end

    while (err_simp > tol)
        err_simp = abs(f4_max*pi^5/(5760*n_simp^4));
        n_simp = n_simp + 1;
    end

    while (err_mid > tol)
        err_mid = abs(f2_max*pi^3/(192*n_mid^2));
        n_mid = n_mid + 1;
    end

    fprintf("tol = %e\ntrapezoid\tn = %8.f,\terror = %e\nmidpoint\tn = %8.f,\terror =
%e\nSimpson\t\tn = %8.f,\terror = %e\n\n", tol, n_trap, err_trap, n_mid, err_mid,
n_simp, err_simp);
end
```

Output:

```
tol = 1.000000e-04
trapezoid  n =      336,  error = 9.964259e-05
midpoint   n =      238,  error = 9.954236e-05
Simpson    n =       14,  error = 9.727228e-05

tol = 1.000000e-12
trapezoid  n =  3344009,  error = 1.000000e-12
midpoint   n =  2364572,  error = 9.999998e-13
Simpson    n =     1293,  error = 9.970397e-13
```

2.

$$t = \int_{v(t_0)}^{v(t)} \frac{m}{R(u)} du,\ R(v) = -v\sqrt{v}$$

$$t = \int_{10}^{5} \frac{10}{-u\sqrt{u}} du$$

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 2\sum_{i=1}^{\frac{n}{2}-1} f(t_{2i}) + 4\sum_{i=1}^{\frac{n}{2}} f(t_{2i-1}) + f(b) \right]$$

$$with\ n\ =\ 2^{10} = 1024,$$

$$t \approx \frac{\frac{5-10}{1024}}{3}\left[\frac{10}{-10\sqrt{10}} + 2\sum_{i=1}^{\frac{n}{2}-1} \frac{10}{-u_{2i}\sqrt{u_{2i}}} + 4\sum_{i=1}^{\frac{n}{2}} \frac{10}{-u_{2i-1}\sqrt{u_{2i-1}}} + \frac{10}{-5\sqrt{5}} \right]$$

Using the following Matlab code:

```
a = 10;
b = 5;
n = 2^10;
h = (b-a)/n; % set up values
f = @(x) 10./(-x.*sqrt(x)); % set up function

i1 = 1:n/2;
i2 = 1:n/2 - 1;
t_2i = a + (2.*i2).*h;
t_2i_1 = a +(2.*i1 -1).*h;

Q = (h/3)*(f(a) + 2*sum(f(t_2i)) + 4*sum(f(t_2i_1)) + f(b)); % Simpson's approximation
q = integral(f, 10, 5); % actual value
fprintf("approximation: %1.16f\texact: %1.16f\nabsolute error: %e\t\trelative error:
%e\n", Q, q, abs(Q-q), abs(Q-q)/q);
```

We get the output:

```
approximation: 2.6197165896626844  exact: 2.6197165896624002
absolute error: 2.842171e-13    relative error: 1.084915e-13
```

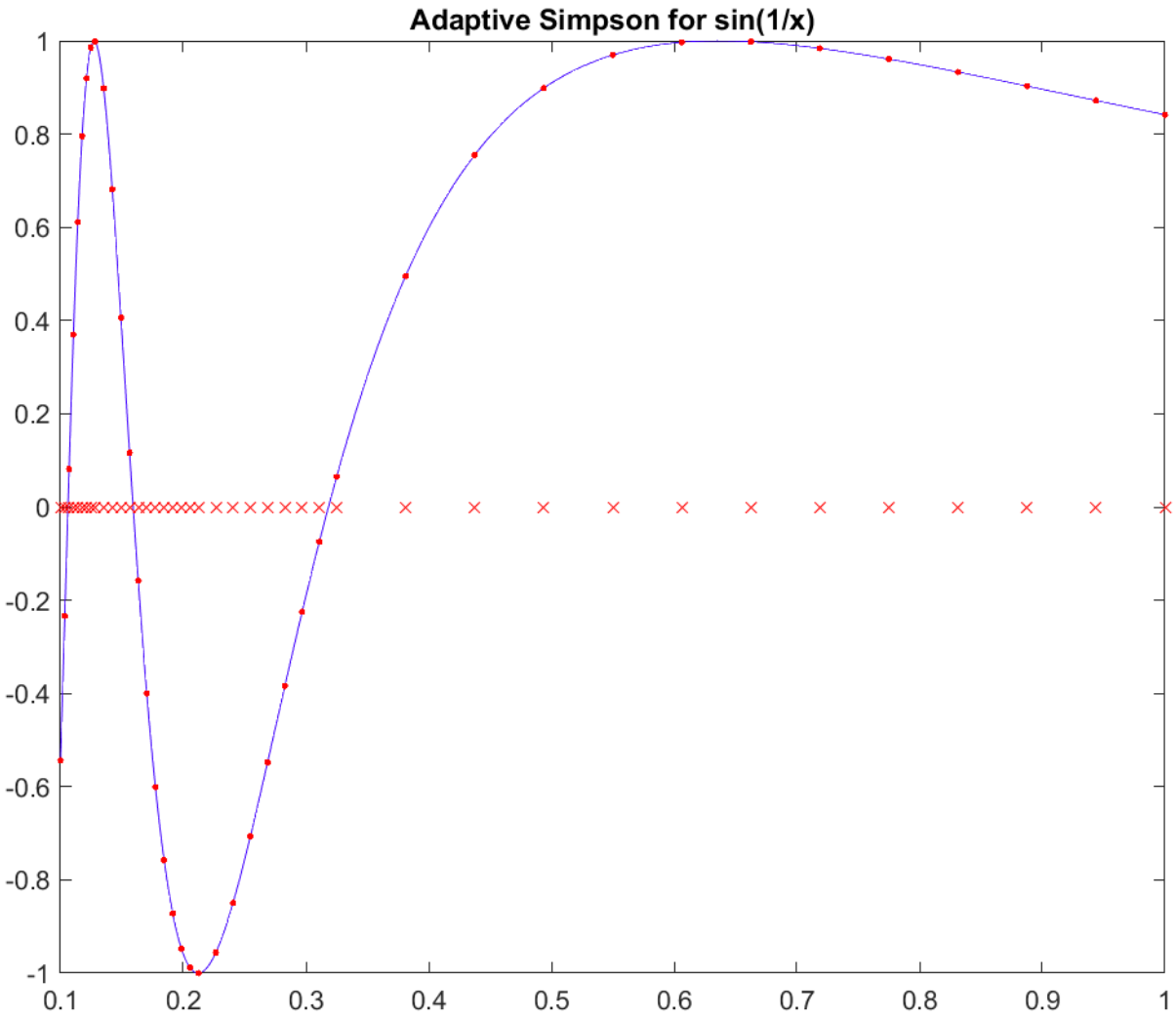
This approximation uses Simpson's method. We get a very small error due to the high value of n as well as the high accuracy of Simpson's method, at least compared to the trapezoid and midpont methods.

3.

Output of main_simpson.m:

```
composite n = 84  error = 9.398097e-05  adaptive n = 40  error = 1.940143e-05
```

Plot:



The composite method requires roughly twice as many points to have an absolute error within 10^{-4} as the adaptive method. I expected n to be larger for the composite method, and actually expected it to be even larger (compared to the adaptive method) than it is.
Looking at the plot, I can see that at the lower values of x, the function rapidly changes. The adaptive method has a higher concentration of points in this area compared to the right side of the graph, where the slope is gentle. This allows it to use as few points as possible while maintaining a low error. On the other hand, the composite method has evenly spaced points, meaning it needs to maintain a higher concentration of points throughout the graph, even when a lower concentration would suffice.

4.
Algorithm:
To obtain c and k for ch^k, I will first calculate k. First, I will create two n values to allow me to compare two separate results. Using n1 and n2, I will calculate the respective hs, xs (x-value arrays for trapz function), Qs (results of trapz function), and errs (absolute error between Q

and q, which will be calculated using the integral function). Then, I will use the equation $\frac{\log\left(\frac{err2}{err1}\right)}{\log\left(\frac{h2}{h1}\right)}$ to calculate k. I obtained this equation through the following method:

$$err = ch^k$$
$$log(err) = log(c) + k \cdot log(h)$$
$$y = mx + b$$
$$log(err) = y,\ log(h) = x,\ k = m,\ log(c) = b$$
$$m = \frac{y2 - y1}{x2 - x1} \therefore k = \frac{\log(err2) - \log(err1)}{\log(h2) - \log(h1)} = \frac{\log\left(\frac{err2}{err1}\right)}{\log\left(\frac{h2}{h1}\right)}$$

From there, I can just plug k in to $err = ch^k$ and rearrange to get $c = \frac{err}{h^k}$.

error_trapz.m:

```
function [c, k] = error_trapz(f, a, b)
    n1 = 2^10;
    n2 = 2^20; % use 2 n values for comparison
    h1 = (b-a)/n1;
    h2 = (b-a)/n2; % calculating h for each n

    x1 = a:h1:b; % creating the x1 vector of n1 evenly spaced points from a to b
    x2 = a:h2:b; % creating the x2 vector of n2 evenly spaced points from a to b

    q = integral(f, a, b); % using integral function to obtain actual value of q
    Q1 = trapz(x1, f(x1));
    Q2 = trapz(x2, f(x2));  % calculating trapz value for each n
    err1 = abs(Q1 - q);
    err2 = abs(Q2 - q); % calculating error for each n

    k = log(err2/err1)/log(h2/h1); % calculate k by finding the slope when plotted
logarithmically
    c = err1/h1^k; % calculate c by substituting in k
end
```

Output:

```
c=4.156110e+03  k=1.99992
c=6.098277e-02  k=1.99922
```

5. a)

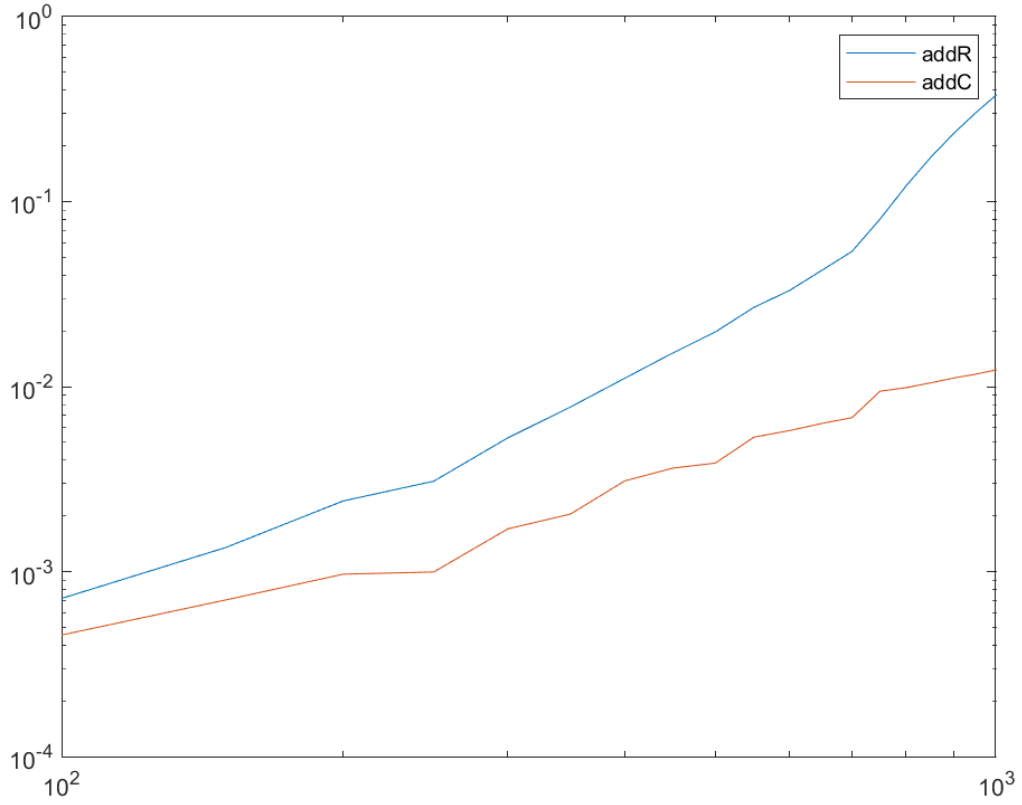
With N = [100:50:1000]:

```
function timeadd()
    N = [100:50:1000];
    [~, i] = size(N);
```

```
    time_R = zeros([1, i]);
    time_C = zeros([1, i]);

    A_mat = zeros([i, 2]);
    R_arr = zeros([i, 1]);
    C_arr = zeros([i, 1]);

    for j = 1:i
        n = N(j);
        disp(n)
        A = randn(n);
        B = randn(n);

        t_r = timef(@addR, A, B);
        t_c = timef(@addC, A, B);
        time_R(1, j) = t_r;
        time_C(1, j) = t_c;

        A_mat(j, 1) = 1;
        A_mat(j, 2) = log(n);
        R_arr(j, 1) = log(t_r);
        C_arr(j, 1) = log(t_c);
    end

    loglog(N, time_R);
    hold on
    loglog(N, time_C);
    legend("addR", "addC")
    hold off

    y_R = A_mat\R_arr;
    y_C = A_mat\C_arr;
    disp(y_R)
    disp(y_C)

    c_row = exp(y_R(1));
    k_row = y_R(2);
    c_col = exp(y_C(1));
    k_col = y_C(2);

    fprintf("c_row = %d\tk_row = %d\nc_col = %d\tk_col = %d\n", c_row, k_row, c_col,
k_col);
end
```

timeadd.m computes:

```
c_row = 9.754912e-10  k_row = 2.766405e+00
c_col = 2.518878e-07  k_col = 1.566363e+00
```



b)

```
c_row = 1.096732e-09   k_row = 2.894349e+00
c_col = 4.908481e-08   k_col = 1.831292e+00
```

My k_row is close to 3, although it should be closer to 2. This may be because Matlab stores matrices in columns, not rows, so adding by row is more work for Matlab. Due to this, the cache memory is insufficient for the operations, artificially inflating the k value.
This is supported by the stark difference between k_row and k_col, despite them adding the exact same matrices. addC.m follows Matlab's natural storage method, and so has a lower k value, whereas addR.m goes against it, adding more work for Matlab.

6. a)

$$F(l) = k(l - l_0) = kl - kl_0 \rightarrow y = mx + b \rightarrow m = k,\ b = -kl_0 = -5.3k$$

$$\begin{bmatrix} 7 & 1 \\ 9.4 & 1 \\ 12.3 & 1 \end{bmatrix} \begin{bmatrix} k \\ -5.3k \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$$

$$A^T A = \begin{bmatrix} \sum_{k=0}^{m} x_k^2 & \sum_{k=0}^{m} x_k \\ \sum_{k=0}^{m} x_k & m+1 \end{bmatrix} = \begin{bmatrix} 288.65 & 28.7 \\ 28.7 & 3 \end{bmatrix}$$

$$A^T f = \begin{bmatrix} \sum_{k=0}^{m} x_k y_k \\ \sum_{k=0}^{m} y_k \end{bmatrix} = \begin{bmatrix} 125.4 \\ 12 \end{bmatrix}$$

$$\begin{bmatrix} 288.65 & 28.7 \\ 28.7 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} 0.0710 & -0.6791 \\ -0.6791 & 6.8303 \end{bmatrix}$$

$$\begin{bmatrix} k \\ -5.3k \end{bmatrix} = \begin{bmatrix} 288.65 & 28.7 \\ 28.7 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 125.4 \\ 12 \end{bmatrix} = \begin{bmatrix} 0.0710 & -0.6791 \\ -0.6791 & 6.8303 \end{bmatrix} \begin{bmatrix} 125.4 \\ 12 \end{bmatrix} = \begin{bmatrix} 0.7525 \\ -3.1988 \end{bmatrix}$$

$$\frac{-3.1988}{0.7525} = -4.251$$

b)

$$F(l) = k(l - l_0) = kl - kl_0 \rightarrow y = mx + b \rightarrow m = k,\ b = -kl_0 = -5.3k$$

$$\begin{bmatrix} 8.3 & 1 \\ 11.3 & 1 \\ 14.4 & 1 \\ 15.9 & 1 \end{bmatrix} \begin{bmatrix} k \\ -5.3k \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 8 \\ 10 \end{bmatrix}$$

$$A^T A = \begin{bmatrix} \sum_{k=0}^{m} x_k^2 & \sum_{k=0}^{m} x_k \\ \sum_{k=0}^{m} x_k & m+1 \end{bmatrix} = \begin{bmatrix} 656.75 & 49.9 \\ 49.9 & 4 \end{bmatrix}$$

$$A^T f = \begin{bmatrix} \sum_{k=0}^{m} x_k y_k \\ \sum_{k=0}^{m} y_k \end{bmatrix} = \begin{bmatrix} 355.6 \\ 26 \end{bmatrix}$$

$$\begin{bmatrix} 656.75 & 49.9 \\ 49.9 & 4 \end{bmatrix}^{-1} = \begin{bmatrix} 0.0292 & -0.3643 \\ -0.3643 & 4.7941 \end{bmatrix}$$

$$\begin{bmatrix} k \\ -5.3k \end{bmatrix} = \begin{bmatrix} 656.75 & 49.9 \\ 49.9 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 355.6 \\ 26 \end{bmatrix} = \begin{bmatrix} 0.0292 & -0.3643 \\ -0.3643 & 4.7941 \end{bmatrix} \begin{bmatrix} 355.6 \\ 26 \end{bmatrix} = \begin{bmatrix} 0.9125 \\ -4.8831 \end{bmatrix}$$

$$\frac{-4.8831}{0.9125} = -5.351$$

Based on dividing our a and b values (k and $-kl_0$), I would estimate that the values from b) are a better fit for the whole data. In a), $-l_0$ evaluates to -4.251, and in b), $-l_0$ evaluates to -5.351. We know that l_0 is 5.3, which the answer in b) is much closer to (difference of 0.051 vs. 1.049).

However, I did want to be sure, so I used this Matlab code:

```
f_a = @(x) 0.7525*x - 3.1988;
f_b = @(x) 0.9125*x - 4.8831;
x = [7 9.4 12.3 8.3 11.3 14.4 15.9];
y = [2 4 6 2 5 8 10];

fprintf("values:\n")
disp(f_a(x))
disp(f_b(x))
fprintf("diff:\n")
disp(abs(y-f_a(x)))
disp(abs(y-f_b(x)))

a_avg = mean(abs(y-f_a(x)));
b_avg = mean(abs(y-f_b(x)));
a_max = max(abs(y-f_a(x)));
b_max = max(abs(y-f_b(x)));
fprintf("a err: avg = %f\tmax = %f\n", a_avg, a_max);
fprintf("b err: avg = %f\tmax = %f\n", b_avg, b_max);
```

to get:

```
values:
    2.0687    3.8747    6.0569    3.0470    5.3044    7.6372    8.7659
    1.5044    3.6944    6.3407    2.6907    5.4282    8.2569    9.6257

diff:
    0.0687    0.1253    0.0569    1.0470    0.3044    0.3628    1.2341
    0.4956    0.3056    0.3407    0.6907    0.4282    0.2569    0.3743
a err: avg = 0.457029  max = 1.234050
b err: avg = 0.413129  max = 0.690650
```

As we can see, both the average and maximum error for b) are lower than for a), showing my assumption to be correct. The values from b) fits the whole data set best.