CS/Math 3414 Assignment 5
Solution Sketches

---

1. For the various spacings given in the question, the number of points involved can be computed as $n = (b-a)/h$. Since $f(x) = |x|$, we have

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{0} (-x)dx + \int_{0}^{1} (x)dx = 2\int_{0}^{1} xdx = 1.$$

The lower sums give a lower bound on this integral, so they will always be less than or equal to this value. When $h = 2$, we have only one integration interval. The greatest lowest bound (glb) of $f(x)$ in this interval is 0, and so the lower sum is zero. When $h = 1$, there are two integration intervals but the glb is zero in both, giving the lower sum as zero. Only when $h = 1/2$ do we get a better bound (namely 1/2). The upper bounds are better for this problem as they start with the complete rectangle that encloses $f(x)$ and start shrinking the area. For instance, the upper sum for $h = 2$ is 2. The trapezoidal rule gives values bounded by the lower sum from below and by the upper sum from above. The complete set of answers are given below:

h	2	1	1/2	1/4
Lower sums	0	0	1/2	3/4
Upper sums	2	2	3/2	5/4
Trapezoids	2	1	1	1

2. Since we have $n+1$ equally spaced points, there are $n-1$ subintervals in the integration. The trapezoidal rule uses trapezoids to model the areas in each of these subintervals. If the width of each subinterval is h, then the trapezoidal rule is computing:

$$\begin{aligned}\int_{0}^{1} x^2 dx &= h\sum_{i=1}^{n-1} x_i^2 + \frac{h}{2}(x_0^2 + x_n^2) \\ &= \frac{1}{n^3}\{1^2 + 2^2 + 3^2 + \cdots + (n-1)^2\} + \frac{1}{2n} \\ &= \frac{1}{6n^3}n(n-1)(2n-1) + \frac{1}{2n}\end{aligned}$$

In the above, we have used the fact that $x_0 = 0$, $x_n = 1$, $h = 1/n$, and $x_i = i/n$. As $n \to \infty$, the above sum converges to

$$\lim_{n\to\infty}\left(\frac{1}{3} + \frac{1}{6n^2}\right) = \frac{1}{3}$$

3. There are three significant differences between how MATLAB implements the QUAD function and the Simpson algorithm as described in your book. All of these would have been clear if you took the trouble to look at the code listing for `quad` (and the function it calls, namely `quadstep`). Here are the headlines.

One of the first things that `quad` does is embodied in the statements:

```
% Initialize with three unequal subintervals
h = 0.13579*(b-a);
x = [a a+h a+2*h (a+b)/2 b-2*h b-h b];
```

Many of you didn't read past the comment and mistook this to mean that the algorithm is doing a three-way recursive division, rather than a two-way division. In fact, it is *not* doing a three-way recursive division but is instead dividing up the given integral into three integrals and computing them separately (each by a recursive Simpson's rule). If you scrolled down the code, you would see where this happens. In other words, this is not the place where recursion happens.

The second difference has to do with 'safety' code such as:

```
% Fudge endpoints to avoid infinities
if ~isfinite(y(1))
        ...
end
if ~isfinite(y(7))
        ...
end
```

This helps MATLAB overcome the problems with evaluating integrals such as

$$\int_0^{1/3} \frac{dx}{3x-1}$$

that have end-point singularities. If you look at the code represented above as `...` you will see what MATLAB uses in place for the end-point values.

The third difference can be seen in the `quadstep` function called by `quad`, which is really the recursive Simpson's algorithm. Everything is as it should be except for the lines:

```
...
[Qac,fcnt,warnac] = quadstep(f,a,c,fa,fd,fc,tol,trace,fcnt,hmin,varargin{:});
[Qcb,fcnt,warncb] = quadstep(f,c,b,fc,fe,fb,tol,trace,fcnt,hmin,varargin{:});
...
```

Notice that the same tolerance (`tol`) is used in both recursive calls instead of `tol`/2! This is a significant deviation from what we studied in class and what is described in the book (see page 226).

This is so because MATLAB doesn't want to be *too conservative*. It is frequently the case that one of the recursive calls can get an estimate of its subintegral to a much greater tolerance than the other (can). In such a case, the second recursive call has more leverage in meeting its accuracy requirement. We would thus be too stringent in requesting `tol`/2 from it (since the other subintegral is picking up most of the tab). This technique is called *banking* and effectively allows the second subintegral to get away with a less accurate answer. To implement this effectively, the code should have been written as:

```
...
[Qac,fcnt,warnac] =
              quadstep(f,a,c,fa,fd,fc,tol/2,trace,fcnt,hmin,varargin{:});
% determine an error estimate, say lefte, from this part of the problem
% and use it in the next part.
```

```
[Qcb,fcnt,warncb] =
          quadstep(f,c,b,fc,fe,fb,tol-lefte,trace,fcnt,hmin,varargin{:});
...
```

However, the designers of MATLAB chose not to be so careful and claim that this problem 'rarely surfaces.' It is possible to design an integral where MATLAB will be tricked into providing an answer that does not satisfy the requested error bound. In fact, it is always possible to fool any adaptive integration routine!

4. Both the integrals are ways of computing π. The MATLAB code QUAD will give you approximations to π as 3.1416 (unless you requested greater significance). Please also note the differences between QUAD's implementation of the Simpson's algorithm and the algorithm as described in your book (previous question).

5. We are given that
$$\int_{-1}^{1} f(x)dx = \alpha f(\frac{-1}{2}) + \beta f(0) + \gamma f(\frac{1}{2})$$
holds for polynomials of degree upto 2. By substituting 1, x, and x^2 for $f(x)$, we get the three simultaneous equations:
$$\begin{aligned} \alpha + \beta + \gamma &= 2 \\ -\frac{1}{2}\alpha \quad + \frac{1}{2}\gamma &= 0 \\ \frac{1}{4}\alpha \quad + \frac{1}{4}\gamma &= \frac{2}{3} \end{aligned}$$
This gives $\alpha = \frac{4}{3}$, $\beta = -\frac{2}{3}$, and $\gamma = \frac{4}{3}$.

6. We are given that
$$\int_{-2}^{2} |x| f(x)dx \approx Af(-1) + Bf(0) + Cf(1)$$
is exact for polynomials of degree upto 2. Proceeding as in the previous question, we get:
$$\begin{aligned} A + B + C &= 4 \\ -A \quad + C &= 0 \\ A \quad + C &= 8 \end{aligned}$$
This gives $A = 4$, $B = -4$, and $C = 4$. To see if the rule is exact for polynomials of degree greater than 2, we try substituting x^3, x^4, $\cdots$ in the rule and determine if the equality holds. For x^3, we get $0 = -A + C$ which is true. For x^4, we get $\frac{64}{3} = A + C$ which is not true. Thus, the rule is exact only for polynomials of degree ≤ 3.