quick answers to the midterm problems

1. T/F.

The natural cubic spline interpolant to a linear polynomial is always that polynomial since it is (1) piecewise cubic and continuous with first and second derivative continuous, and (2) has a zero second derivative at the endpointss. The natural cubic spline interpolant to a polynomial of higher degree than linear is not itself unless it happens to have a vanishing second derivative at the two endpoints (for a quadratic polynomial, this would mean that it is actual a constant). Finally, the complete cubic spline interpolant to any cubic polynomial is that polynomial.

The error term of neither the Midpoint, nor the Trapezoidal, nor Simpson's rule involves the third derivative.

A `Matlab` statement like the following `rand(2,3); A = ans(2*ones(3,1),[3:-1:1]);` generates a matrix with `length(2*ones(3,1))` rows and `length([3:-1:1])` columns regardless of what `ans` might be at that point, provided only that `ans` has as many rows and columns as are required here (in this example, `ans` must have at least 2 rows and 3 columns).

The divided difference $f[x_1, \ldots, x_k]$ equals $f^{(k-1)}(\eta)/(k-1)!$ for some η in the smallest interval containing all the x_i, hence is positive if that derivative is positive on that smallest interval (note, it's not the kth derivative nor any other but the $(k-1)$st derivative of f that matters here).

2. Rewriting a script like the following:

```
for i=1:n
   for j=1:m
      a(i,j) = i*j + 2;
   end
end
```

involves noticing that a matrix with `n` rows and `m` columns is being constructed according to the recipe `[1:n]'*(1:m)+2`, and this can also (slightly less efficiently) be written `repmat((1:n)',1,m).*repmat(1:m,n,1)+2`.

3. Constructing a Newton form of the cubic Hermite interpolant p to some function at the two points a and b involves nothing more than setting up the divided difference table for that function for the points $(x_1, x_2, x_3, x_4) := (a, a, b, b)$, and reading off the requisite coefficients, as is done explicitly in the notes, as is the evaluation of it by Nested Multiplication.

4. Considering the approximation to some function f on the interval $[0 \,..\, 1]$ by the piecewise linear function L with breakpoints $x_i := i/N$, $i = 0{:}N$, that interpolates to f at x_i, $i = 0{:}N$, seemed to be 'French' to many students (as one student put it). It had been meant to be an easy problem, given that it was solved explicitly in the book and in the notes on piecewise linear interpolation (to which I refer you for the solution). As it turned out, most people didn't have a clue. Writing down the error formula for polynomial interpolation at those $N + 1$ points meant to be completely ignoring what was written in the problem (which speaks explicitly of 'piecewise linear' interpolation).

5. The quadrature rule problem also was meant to be easy. The order of a quadrature rule is the largest integer r such that the rule is exact for all polynomials of degree $< r$. Equivalently, it is the order of the derivative that occurs in its error term. Hence it is the smallest r such that the quadrature rule fails to integrate x^r exactly. – For the specific example, the rule handles constants exactly but not $f(x) = x$, hence has order 1.

6. To write down a number as a normalized 2-decimal-digit-floating.# means to write it as $(.b_1b_2)_{10}10^e$, with $b_1 > 0$ and $b_1, b_2 \in \{0, 1, \ldots, 9\}$ and such that $(.b_1b_2)10^6$ equals that number. E.g., for 10,000, this would be $(.10)_{10}10^5$. But, because of IEEE standard and since I only cared that you understood the essential here, I was happy to accept answers like $1 * 10^4$ or $(1.0) * 10^4$ or even $10. * 10^3$ (the essential being that only two significant decimal digits are allowed).

This essential point was enforced in the grading of the calculation which was to be done in the corresponding floatingpoint arithmetic. That meant that the calculation had to be done step by simple step, first carrying out that step in exact arithmetic and then rounding the result to the nearest 2-decimal-digit fl.#. E.g., $\sqrt{10,000 + 1} + 100$ involves the steps $(10,000 + 1) \rightarrow 10,000$, its squareroot is 100, etc.

But, before the fl.# calculation, you were supposed to reformulate the expression so as to avoid the dreaded catastrophic cancellation (when two numbers, each near 100, are subtracted from each other). The idea was to carry out something close to that subtraction symbolically, i.e., in formula. In this case, this amounted to multiplying and dividing by the same expression with a $+$ rather than a $-$, carrying out the subtraction in the numerator symbolically, leaving an expression whose floating-point evaluation involved no catastrophic cancellation.