CSCD37: Assignment #2

1. Recall the full-Newton algorithm for solving the nonlinear system $F(x) = 0; F, x \in \mathbb{R}^n$:

```
Generate an initial approximation \hat{x}_0 for k=0,1\dots until convergence compute -F(\hat{x}_k) compute \frac{\partial F(\hat{x}_k)}{\partial x} solve \frac{\partial F(\hat{x}_k)}{\partial x}\Delta_k = -F(\hat{x}_k) update \hat{x}_{k+1} = \hat{x}_k + \Delta_k end for
```

- (a) This algorithm is computationally expensive. Give a detailed analysis of the cost using standard big-oh notation. You may use results from CSCC37; e.e., the cost (measuring flops) of the LU-factorization of an $n \times n$ matrix is $(1/3)n^3 + \mathcal{O}(n^2)$.
 - The following analysis makes the assumption that each component of F can be evaluated in approx 1 flop, and that taking the derivative also takes around 1 flop. For each iteration of the loop, first evaluating the function F at a point $x \in \mathbb{R}^n$ will take n operations. The evaluation of the n derivatives will also take n, so evaluating n derivatives at n points will take n^2 time. Solving the linear system of the derivative matrix against the original function will take around $(1/3)n^3 + \mathcal{O}(n^2)$ as was seen in CSCC37. Finally updating a vector with an addition of 2n length vectors will simply be n additions. Overall, the entire loop iteration will take $n^2 + n + (1/3)n^3 + \mathcal{O}(n^2) + n = (1/3)n^3 + \mathcal{O}(n^2) = \mathcal{O}(n^3)$ steps per iteration. This cannot be generalized outside of iterations, as convergence is not always guaranteed and k may run on infinitely.
- (b) We briefly discussed in lecture the "quasi-Newton algorithm" for solvling nonlinear systems. Modify the algorithm above to take advantage of the optimizations we discussed. You do not need to implement the modifications ... pseudo-code will suffice. Be careful to discuss both flop optimizations and convergence issues ("X-test" and "F-test" tolerance, maximum number of iterations, condition of Jacobian, how long to hold Jacobian fixed, etc.).

- 2. In lecture we derived the divided-difference (Newton) form of the interpolating polynomial for the simple interpolation problem. This question will investigate how the Newton polynomial can be used for osculatory interpolation.
 - (a) Prove that

$$\lim_{\substack{x_{i+j} \to x_i \\ 1 \le j \le k}} y[x_{i+k}, x_{i+k-1}, \dots, x_i] = \frac{y^{(k)}(x_i)}{k!}$$

Provided $y \in \mathcal{C}^k$.

Proof. Examine first, the error formula given in lecture, E(x) = y(x) - p(x) for any polynomial interpolant, then using the Newton polynomial as p(x), it produces:

$$E(x) = y(x) - y[x_i] + (x - x_i)y[x_{i+1}, x_i] + \dots + (x - x_i)(x - x_{i+1}) + \dots + (x - x_{i+k-1})y[x_{i+k}, \dots, x_i]$$

Knowing that E(x) has n+1 distinct roots (at the interpolation constraints), by Rolle's theorem, gives that the kth derivative of E(x) has at least 1 zero $E(\psi)$ where $\psi \in \text{span}\{x_i, \ldots, x_{i+k}\}$. Looking at the Newton polynomial, the kth derivative will only have the leading coefficient times k! since it is a degree k polynomial. This coefficient, from the definition can be seen as $y[x_{i+k}, \ldots, x_i]$

$$\Rightarrow \frac{d^k}{dx^k} E(x) = y^{(k)}(x) - y[x_{i+k}, \dots, x_i]k!$$

$$\frac{d^k}{dx^k} E(\psi) = y^{(k)}(\psi) - y[x_{i+k}, \dots, x_i]k!$$

$$0 = y^{(k)}(\psi) - y[x_{i+k}, \dots, x_i]k!$$

$$y[x_{i+k}, \dots, x_i]k! = y^{(k)}(\psi)$$

$$y[x_{i+k}, \dots, x_i] = \frac{y^{(k)}(\psi)}{k!}$$

$$\Rightarrow \lim_{\substack{x_{i+j} \to x_i \\ 1 \le j \le k}} y[x_{i+k}, \dots, x_i] = \lim_{\substack{x_{i+j} \to x_i \\ 1 \le j \le k}} \frac{y^{(k)}(\psi)}{k!}$$

$$\Rightarrow \lim_{\substack{x_{i+j} \to x_i \\ 1 \le j \le k}} y[x_{i+k}, \dots, x_i] = \frac{y^{(k)}(x_i)}{k!}$$

The last line is since if $\forall x_{i+j}, |x_{i+j} - x_i| < \varepsilon \implies \operatorname{span}\{x_i, \dots, x_{i+k}\} \subset (x_i - \varepsilon, x_i + \varepsilon) \implies \psi \to x_i$.

(b) The result in (a) tells us that divided differences can be replaced with derivatives as data points coincide. Using this result, construct a divided difference table to find the coefficients of the Newton polynomial of degree 6 or less that satisfies the following interpolation conditions:

$$p(-1) = 4$$
 $p(0) = 7$ $p(1) = 28$ $p(2) = 247$ $p'(0) = 6$ $p'(1) = 56$ $p''(1) = 140$

(c) Use the Method of Undetermined Coefficients (i.e., as discussed in lecture, construct and solve an appropriate Vandermonde system) to find the coefficients of the monomial-basis polynomial of degree 6 or less that satisfies the interpolation conditions specified in (b). You may use *MatLab* for this question if you wish. Verify that you have obtained the same polynomial as in (b).

 $\iff p(x) = x^6 + 2x^5 + 3x^4 + 4x^3 + 5x^2 + 6x + 7$

```
Editor - C:\Users\Keegan\school\cscd37\a2q2c.m
  a7q1b.m × approx.m × a2q2c.m × +
       size = 7;
 2 -
       vandermonde = zeros(size);
 3 -
      coeffl = 0:(size-1);
 4 -
       coeff2 = [0,0,2,3*2,4*3,5*4,6*5];
      powers0 = 0:(size-1);
 5 -
 6 -
      powers1 = [0,0:(size-2)];
7 -
      powers2 = [0,0,0:(size-3)];
 8 -
       x = [-1, 0, 1, 2];
9 -
      vandermonde(1,:) = x(1).^powers0;
10 -
      vandermonde(2,:) = x(2).^powers0;
11 -
      vandermonde(3,:) = (x(2).^(powers1)).*coeff1;
12 -
      vandermonde(4,:) = x(3).^powers0;
13 -
      vandermonde(5,:) = (x(3).^(powers1)).*coeff1;
14 -
      vandermonde(6,:) = (x(3).^(powers2)).*coeff2;
15 -
       vandermonde(7,:) = x(4).^powers0;
      y = [4 7 6 28 56 140 247];
16 -
17 -
       soln = vandermonde\v';
Command Window
  Trial>> soln
  soln =
      7.0000
      6.0000
      5.0000
      4.0000
      3.0000
```

2.0000 1.0000

 f_{x} Trial>>

3. Consider the function $y \in \mathcal{C}^{n+1}$ and the polynomial $p \in \mathcal{P}_n$ which satisfies

$$p^{(j)}(x_i) = y^{(j)}(x_i); \ j = 0, \dots, j_i; \ i = 0, \dots, k; \ \sum_{i=0}^{k} (j_i + 1) = n + 1;$$

with all of the x_i distinct. The error in this polynomial interpolant is given by

$$E(x) = y(x) - p(x) = \frac{y^{(n+1)}(\xi)}{(n+1)!} (x - x_0)^{j_0+1} (x - x_1)^{j_1+1} \cdots (x - x_k)^{j_k+1}$$
(1)

where $\xi \in \text{span}\{x_0, \dots, x_k, x\} = [\min\{x_0, \dots, x_k, x\}, \max\{x_0, \dots, x_k, x\}] \text{ provided } y \in \mathcal{C}^{n+1} \text{ on span}\{x_0, \dots, x_k, x\}$.

This is a fundemental formula in Numerical Approximation. The following is an outline of a possible derivation of (1). In this question you will expand this outline by proving certain key statements.

If $x = x_i$, i = 0, ..., k, then y(x) - p(x) = 0 since p interpolates y at these k+1 points. Also, $E(x_i) = 0$ in (1) since $(x_i - x_i)^{j_1+1} = 0$. Therefore, (1) holds when $x = x_i$.

Now assume $x \neq x_i$ for any i = 0, ..., k and consider x fixed. Let F(t) = y(t) - p(t) - CW(t) where C = [y(x) - p(x)]/W(x) is a constant and

$$W(t) = (t - x_0)^{j_0 + 1} (t - x_1)^{j_1 + 1} \cdots (t - x_k)^{j_k + 1}$$
(2)

is a polynomial of degree n+1. Clearly F(x)=0, and also

$$F^{(j)}(x_i) = 0; \ j = 0, \dots, j_i; \ i = 0, \dots, k.$$
 (3)

Therefore, counting multiplicities, F(t) has at least n+2 zeros in span $\{x_0, \ldots, x_k, x\}$, which implies $F^{(n+1)}(t)$ has at least 1 zero in span $\{x_0, \ldots, x_k, x\}$, or, in other words,

$$F^{(n+1)}(\xi) = 0, \xi \in \text{span}\{x_0, \dots, x_k, x\}.$$
(4)

But

$$F^{(n+1)}(t) = \frac{d^{n+1}}{dt^{n+1}} [y(t) - p(t) - CW(t)] = y^{(n+1)}(t) - (n+1)!C.$$
 (5)

Therefore,

$$F^{(n+1)}(\xi) = 0 \implies y^{(n+1)}(\xi) - (n+1)!C = 0 \implies C = \frac{y^{(n+1)}(\xi)}{(n+1)!}.$$

But

$$C = \frac{y(x) - p(x)}{W(x)} \implies y(x) - p(x) = \frac{y^{(n+1)}(\xi)}{(n+1)!}W(x).$$

Now for the statements you must prove:

(a) Prove that W(t) in (2) is a polynomial of degree n+1.

Proof. Polynomials are closed under exponentiation and basic operations, so it is sufficient to show the highest power to be n+1. For each term $(t-x_i)^{j_i+1}$, the highest power of t is j_i+1 respectively using the Binomial Theorem. This means that the final product will highest power of all of these terms together, so the power will be $\sum_{i=0}^k j_i + 1 = n+1$ by definition.

(b) Prove (3)

Proof. We know that p must approximate y exactly at the interpolation points, so evidently for a point on the jth derivative, $y^{(j)}(x_i) - p^{(j)}(x_i)$ will be 0. For the other term CW(t), taking the derivative only j times will result in the term $(x - x_i)$ to show up in every term generated by the product rule since it has power of $(j_i + 1)$, hence W(t) will also be 0 making $F^{(j)}(x_i) = 0$ at all interpolation points.

(c) Explain how (4) follows from F(t) having at least n+2 zeros in span $\{x_0,\ldots,x_k\}$.

Lemma. Counting multiplicities, if F(t) is C^1 over $span\{x_0, \ldots, x_k\}$ has at least k zeros in $span\{x_0, \ldots, x_k\}$ then F'(t) has at least k-1 zeros in $span\{x_0, \ldots, x_k\}$.

Proof. Suppose the distinct roots of F(t) are ordered as $a_1 < a_2 < \cdots < a_s$ with multiplicities $m(a_i)$ such that $\sum_{i=0}^s m(a_i) = n$. From Rolle's Theorem, we have that there is some $\rho \in (a_i, a_{i+1}), \ 1 \le i < s$ where $F'(\rho) = 0$ immediately giving us s-1 roots in the derivative. Now counting the multiplicities of the previous roots, we get that there are $\sum_{i=0}^s (m(a_i)-1)$ more roots, which summing together gives $\sum_{i=0}^s m(a_i) - s + (s-1) = n-1$ roots, which are all in the span.

From this, it follows immediately that taking n+1 derivatives of a function with n+2 roots will have one root in the same span.

(d) Prove (5).

Proof. Since p is a polynomial of degree n, taking n+1 derivatives will immediately cause it to vanish. On the otherhand, for W(t), we know it is a polynomial of degree n+1 from (a), so taking the (n+1)th derivative gives us the leading coefficient times (n+1)!. Looking back at how W(t) is setup though, since none of the t in each binomial term have any coefficient the resulting expanded polynomials leading coefficient will be 1. This means the (n+1)th derivative of CW(t) is just C(n+1)!.

$$\Rightarrow F^{(n+1)}(t) = \frac{d^{n+1}}{dt^{n+1}} [y(t) - p(t) - CW(t)]$$

$$= \frac{d^{n+1}}{dt^{n+1}} y(t) - \frac{d^{n+1}}{dt^{n+1}} p(t) - \frac{d^{n+1}}{dt^{n+1}} CW(t)$$

$$= y^{(n+1)}(t) - (n+1)!C.$$

4. In lecture we proved that the roots of the Chebyshev polynomial

$$T_k(x) = \cos(k\cos^{-1}(x)), k = 0, 1, \dots$$
 (6)

are the optimal interpolation points on [-1,1].

(a) Prove that (6) is a polynomial of degree k for all $k \geq 0$.

Proof. For the case of k = 0, 1, we have that

$$\cos(k \arccos(x))|_{k=0} = \cos(0) = 1$$
$$\cos(k \arccos(x))|_{k=1} = \cos(\arccos x) = x$$

So these cases hold. Using induction, assume that $T_n(x)$ is a polynomial for n < k, then there are two inductive cases:

First k = 2t is even

$$\cos(k \arccos(x)) = \cos(2(t \arccos(x)))$$

$$= 2(\cos(t \arccos(x)))^2 - 1 \text{ [Double angle identity]}$$

$$= 2(T_t(x))^2 - 1$$

This is again a polynomial since T_t is one by IH, so this case holds.

Second k = 2t + 1 is odd

$$\cos(k \arccos(x)) = \cos(2(t \arccos(x)) + \arccos(x))$$

$$\stackrel{\text{angle sum}}{=} \cos(2t \arccos(x))(\cos(\arccos(x))) - \sin(2t \arccos(x))(\sin(\arccos(x)))$$

$$= xT_{2t}(x) - \sin(2t \arccos(x))(\sin(\arccos(x)))$$

$$\stackrel{\text{double angle}}{=} xT_{2k}(x) - 2\cos(t \arccos(x))\sin(t \arccos(x))(\sin(\arccos(x)))$$

$$= xT_{2t}(x) - 2T_{t}(x)\sin(t \arccos(x))(\sin(\arccos(x)))$$

$$\stackrel{\text{angle prod}}{=} xT_{2k}(x) - T_{t}(x)(\cos((t-1)\arccos(x)) - \cos((t+1)\arccos(x)))$$

$$= xT_{2t}(x) - T_{t}(x)(T_{t-1} - T_{t+1})$$

Which is again a polynomial all the lower degrees of T are polynomials by IH, so both cases hold.

(b) Derive the leading coefficient of (6) (i.e., the coefficient of x^k). The coefficient is 2^{k-1} for $k \ge 1$ and 1 for k = 0.

Proof. Again, using induction, the case of 1 and 0 are trivial, as $T_0(x) = 1$ and $T_1(x) = x$. For the inductive case, consider again even and odd, and assume that it holds for n < k. For k = 2t even, the leading coefficient $\mathcal{LC}(T_k)$:

$$\mathcal{LC}(T_k) = 2\mathcal{LC}(T_t)^2$$
 From the recurrence in (a)
= $2(2^{t-1})^2 = 2(2^{2t-2}) = 2^{k-1}$ By IH

For k = 2t + 1 odd,

$$\mathcal{LC}(T_k) = \mathcal{LC}(T_{k-1}) + \mathcal{LC}(T_t)\mathcal{LC}(T_{t+1})$$
 From the other recurrence in (a)

$$= 2^{k-2} + (2^{t-1})(2^t)$$
 From IH

$$= 2^{k-2} + 2^{2t-1}$$

$$= 2^{k-2} + 2^{k-2} = 2^{k-1}$$

(c) Derive the roots of (6) (i.e., the so-called Chebyshev points). The roots for a polynomial of degree k are $x_i = \cos(\frac{(2i+1)\pi}{2k})$ where $i = 0, \dots k-1$.

Proof. Given the definition for T_k , plugging in x_i gives

$$\cos(k \arccos(x_i)) = \cos(k \arccos(\cos\left(\frac{(2i+1)\pi}{2k}\right)))$$

$$= \cos\left(k\frac{(2i+1)\pi}{2k}\right)$$

$$= \cos\left(\frac{(2i+1)\pi}{2}\right)$$

$$= 0 \text{ When } i \in \mathbb{N}$$

(d) Complete the proof showing why the Chebyshev points are optimal.

Since the Chebyshev polynomial is infact a polynomial, with coefficient 2^{k-1} , we can write it out as $2^{k-1}(x-x_1)(x-x_2)\cdots(x-x_k)$, where $x_i, 1 \leq i \leq k$ are the roots of x_i . It can also be converted to a monic polynomial by dividing by 2^{k-1} . Note that it is also bound between -1 and 1 because of the cos definition of T_k To show that they are optimal, we prove the following:

Theorem. If W(x) is a monic polynomial of degree $\leq k$, then $\max_{x} |W(x)| \geq \max_{x} \left| \frac{T_k(x)}{2^{k-1}} \right| = \frac{1}{2^{k-1}}$

Proof. Suppose $\max_x |W(x)| < \max_x \left| \frac{T_k(x)}{2^{k-1}} \right|$, then in particular $|W(y_i)| < \left| \frac{T_k(y_i)}{2^{k-1}} \right| = \frac{1}{2^{k-1}}$ where $y_i = \cos\left(\frac{i\pi}{k}\right)$

- 5. Consider the definite integral $I(f) = \int_{-1}^{1} f(x) dx$.
 - (a) Construct the interpolatory quadrature rule for this integral based on nodes $-1, -\frac{1}{2}, \frac{1}{2}, 1$.
 - (b) What is the precision of the quadrature rule derived in (a)? Justify your answer.
 - (c) Find as good an error bound as you can for your quadrature rule, assuming f is as smooth as required for your analysis.