HW 8

Objective: Numerical differentiation, symbolic processing, and verifying rate of convergence (aka order of approximation, order of convergence).

0. We discussed in class several methods for systematically performing numerical differentiation in arbitrary order of convergence. Here is a brief summary of what we covered. In the following, we always identify \( x \) with \( x_0 \), which is where we wish to compute numerically the derivative of a function \( f \).

a. The finite difference approximation of the \( i \)-th order derivative of \( f \) is of the form

\[
\frac{d^i f(x)}{dx^i} = \frac{1}{h^i} \sum_{j=0}^{m} a_j f(x_j) + O(h^k)
\]

In this formula

(i) \( \{x_j \mid j = 0, 1, \ldots, m\} \) is a bunch of distinct points where \( f \) is sampled. The derivative of \( f \) is sought at \( x = x_0 \).

(ii) \( h \) is the spacing between the points \( \{x_j\} \) if they are equispaced, or some sort of average spacing of the \( m + 1 \) points. For example, the \( m + 1 \) Chebyshev points in \([-1, 1]\) are not equispaced, We could define \( h \) as \( 2/(m + 1) \).

(iii) We always have to require that \( m \geq i \). Then the order of approximation \( k \) will be \( m - i \) or one order higher if there is symmetry to cancel out the leading term in the error.

(iv) The Taylor series is the tool unifying all the approaches we address here.

b. We first consider finite difference approximation of the first order derivative \( f'(x) \) with \( i = 1 \). There are three methods to do it: forward, backward, and central differences. The sampling points \( \{x_j\} \) could be equispaced, or arbitrarily distributed.

c. Let’s consider equispaced points \( \{x_j \mid j = 0, 1, \ldots, m\} \). Given \( h > 0 \), let \( E \) be the forward shift operator so that \( E f(x) = f(x + h) \). Then the forward, backward, and central difference operators \( \Delta, \nabla, \) and \( \delta \) can be defined in terms of \( E \)

\[
\Delta = E - I, \quad \nabla = I - E^{-1}, \quad \delta = E^{1/2} - E^{-1/2}
\]

Remark 0.1 Each of the three operators is proportional to \( h \) and thus small. For example, \( |\Delta f(x)| = O(h) \), or for short \( \Delta = O(h) \), for any given smooth function \( f \). Similarly, \( \nabla = O(h) \) and \( \delta = O(h) \).

d. Forward difference approximation of the derivative. The Taylor series for \( f(x+h) \) about \( x \) implies that

\[
E = e^{hD}, \quad \text{or} \quad I + \Delta = e^{hD}
\]

with \( D \) the differentiation operator. Thus \( D \) can be expressed as polynomial (of infinite degree) of \( \Delta \)

\[
hD = \log(I + \Delta) = \Delta - \Delta^2/2 + \Delta^3/3 - \cdots
\]

An order \( k \) approximation to \( D \) is obtained by truncation after the \( k \)-th term

\[
hD = \Delta - \Delta^2/2 + \Delta^3/3 - \cdots + (-1)^{k-1} \Delta^k/k + O(h^{k+1})
\]
namely

\[ f'(x) = \frac{1}{h} \sum_{j=1}^{k} \frac{(-1)^{j-1}}{j} \Delta_j f(x) + O(h^k) \]  

(6)

Now we can convert it to the form (1) where \( m = k \), \( x_j = x + jh \), \( j = 0, 1, \ldots, m \). Namely, we can figure out from (1) the linear combination coefficients \( a_j \), \( j = 0, 1, \ldots, m \). This is easily achieved by (i) Our exertion in algebra, or more reliably (ii) Matlab’s symbolic processing without much thinking on our part - let live and think.

% k-order forward difference approximation to \( D^i \),

```
syms x; k=5; i=3; % take i-th derivative
n=k+i; % compute first n terms in Taylor for log
u=log(1+x); % u for hD, x for Delta
v=taylor(u^i,n); % v for (hD)^i, retain first m terms
w=subs(v,x-1); % w for (hD)^i, x is now for E
w=expand(w); % same w expanded in monomials of E
coeff=sym2poly(w); % obtain the coefficients \( a_j \) of monomial \( E^j \)
aj=fliplr(coeff); % flip it and call it aj
```

% print off the coefficients

```
fprintf('f(x+%1.0fh) %26.16e\n', [0:n-1]; aj)
```

% check order of convergence: is it a k-th order formula?

```
f=inline('exp(t)');
t=2.209264; DiF_exact=f(t);
h2=[0.1, 0.05]; % two h values to check rate of convergence
for s=1:2,
    h=h2(s); t_sample=transpose([t:h:t+(n-1)*h]);
    f_sample=f(t_sample); DiF_numer(s)=(aj*f_sample)/h^i;
    rel_err(s)=abs((DiF_exact-DiF_numer(s))/DiF_exact);
end
rate_conv = log2(rel_err(1)/rel_err(2));
fprintf('rate of convergence: %4.2f\n', rate_conv)
```

% look at the two approximations

```
fprintf('DiF_exact =%26.16e\n', DiF_exact)
fprintf('DiF_numer(1)=%26.16e\n', DiF_numer(1))
fprintf('DiF_numer(2)=%26.16e\n', DiF_numer(2))
```

**Remark 0.2** Please observe that some of the Matlab symbolic functions may be what they call “fragile”, that is, buggy. My version of Matlab won’t do the following correctly

```
syms x;
taylor(x^5,6)
taylor(x^6,7)
```

The correct print out should be \( x^5 \) and \( x^6 \), but my Matlab won’t give me that. I trust that you’d not think this is time to blame the computer; if the situation is not great, so much care must be exercised to cope with it, which we exert ourselves to do anyway when computing. Actually there should not be any exertion if only we go step by step, literally, not two steps at a time, in programming. In order to go fast, most of us need to go slow and organized first, unless we’re Gauss or Euler or von Neumann.
e. Backward difference approximation of the derivative. Almost verbatim, we express $hD$ in terms of $\nabla$ and obtain the analog to (4)

$$hD = \log(I - \nabla) = \nabla + \nabla^2 / 2 + \nabla^3 / 3 + \cdots$$

(7)

from which we develop the backward difference approximation to the first derivative. A little alteration - a few sign changes - here and there in the Matlab code will do the actual job.

f. Finite difference approximations of higher order derivatives. This is very simple: choose $i$ in the above Matlab code to be whatever value you want; $i=4$ will give us fourth order derivative. The rate of convergence will still be $k$ as before since we truncate the Taylor series after $k+i$ terms.

g. Central difference approximation of the derivative. Similarly, the Taylor expansions

$$E^{\pm 1/2} = e^{\pm \nfrac{1}{2}hD}$$

(8)

give rise to the relation

$$\delta = 2 \sinh(\nfrac{1}{2}hD)$$

(9)

whose inversion yields

$$hD = 2 \text{arcsinh}(\delta/2)$$

(10)

Replace $\log(1 + x)$ in the Matlab code by $2 \text{asinh}(x/2)$, in addition to a few minor changes of the code, and we’re done with the central difference approximation.

Remark 0.3 The central difference approximation to odd order derivatives is not really useful since we don’t have $f(x)$ sampled at the half grid points. It is only useful to compute even order derivatives $f''(x), f^{(4)}(x)$, and so on, with $i=2,4$ and so forth.

h. Stability. Taking derivatives of a function is not a stable procedure, and should be done with care. The condition number of taking the $i$-th derivative of a function $f(x)$ is proportional to $1/h^i$

$$\kappa \approx \frac{f(x)}{h^i f^{(i)}(x)}$$

(11)

on an equispaced grid. It says that for example taking the ninth derivative numerically at $h = 0.01$ is most likely to be meaningless because $1/h^i$ will be over $10^{16}$. In many applications, we need to take derivatives numerically. When they are used in a legitimate way, the stability issue will be naturally suppressed by intervening factors in the formulas in which they appear. A typical example is to compute derivatives numerically for the Euler-Maclaurin summation formula where $d^i f(x)/dx^i$, which is obtained numerically, will be multiplied by $h^i$, and then summed up in the formula; see the last homework problem for details.

i. Additional stability issues. The forward and backward finite difference approximations to derivatives have their additional instabilities to the already unstable process of taking derivatives. The coefficients $a_j$ of (1) become increasingly large for $n$ greater than 7 where $n=k+i$ is the number Taylor terms we retain, or $n+1$ is the number of sampling points employed in the finite difference scheme. Try to run the Matlab code with $k=16; i=4$ and look the coefficients $a_j$. In contrast, the coefficients $a_j$ for the central difference are quite small, giving rise to no additional instability.

j. Taking arbitrary order derivative on an arbitrary grid. Finally, let’s look at a systematic method to compute derivatives of arbitrary order, not just even order ones, from arbitrarily spaced sampling points $\{x_j \mid j = 0, 1, 2, \ldots, m\}$, and with arbitrary rate of convergence. Of course the rate of convergence is always an integer. Denote by $h$ some average spacing of the $m+1$ points. We want to compute derivatives of $f$ at $x = x_0$ which in practice
should be placed as near the center of \( \{ x_j \} \) as possible, so we can always assume that 
\[
\max_j (x_0, x_j) = O(h).
\]
We again reply on the Taylor series for \( f(x_j) \) about \( x = x_0 \)
\[
f(x_j) = \left[ \sum_{i=0}^{n} \frac{(x_j - x)^i}{i!} D^i \right] f(x) + O(h^{n+1})
\] (12)
In other words,
\[
E(x_j-x)/h f(x) = \left[ \sum_{i=0}^{n} \frac{(x_j - x)^i}{i!} D^i \right] f(x) + O(h^{n+1})
\] (13)
or equivalently
\[
E(x_j-x)/h = \frac{1}{h^i i!} (hD)^i + O(h^{n+1})
\] (14)
Therefore,
\[
E(x_j-x)/h = \sum_{i=0}^{n} \frac{(x_j - x)^i}{h^i i!} y_i, \quad j = 1, 2, \ldots, m
\] (15)
could be interpreted as a system of \( m \) equations for \( n \) unknowns \( y_i = (hD)^i, \quad i = 1, 2, \ldots, n \) whose least-squares solution would yield approximations to \( D^i \). As we said, we must have \( m \geq n \) in order to determine the first \( n \) derivatives with any accuracy: The number of sampling points must be greater than the highest order derivatives to be approximated. Thus the least-squares problem is an over determined system for which we already know the level of residual, \( O(h^{n+1}) \). So roughly speaking without taking symmetry into account, the rate of convergence will be \( n + 1 - i \) for approximating the \( i \)-th order derivative \( D^i \); see Remark 0.4.

Let \( Ay = b \) be our system of \( m \) equations. Obviously, \( A \) is \( m \)-by-\( n \) with
\[
A_{ji} = \frac{(x_j - x)^i}{h^i i!}, \quad 1 \leq j \leq m, \quad 1 \leq i \leq n
\] (16)
and
\[
b_j = E(x_j-x)/h - I, \quad 1 \leq j \leq m
\] (17)
where \( I \) is the identity operator coming from the \( i = 0 \) term in the summation. Needless to say, this is not a system of linear equations in the original sense where \( b_j \) would be a number, \( y_i \) also a number. If we introduce more carefully the entire apparatus of rings and groups, and what not, into this whole concoction of operators, this linear system will make perfect sense; moreover, all the mathematicians of the Soviet State will be utterly satisfied, and will be singing and dancing on the streets of Moscow. But we cannot do that; we are serious people. At any rate, here is what we get if we solve the system for \( y \)
\[
D^i = \frac{1}{h^i} \sum_{j=1}^{m} A_{ij}^+ b_j, \quad 1 \leq i \leq n
\] (18)
where \( A_{ij}^+ \) is the \((i, j)\) entry of the pseudo inverse \( A^+ \). Since we cannot make the residual \( Ay - b \) vanish - it is always of the order \( h^{n+1}, \) (18) ought to be regarded as an approximation, as opposed to an equality. Its true meaning should be
\[
\frac{d^i f(x)}{dx^i} \approx \frac{1}{h^i} \sum_{j=1}^{m} A_{ij}^+ [f(x_j) - f(x)], \quad 1 \leq i \leq n
\] (19)
This is a bona fide finite difference approximation to the \( i \)-th derivative, almost in the form of (1) with
\[
x = x_0, \quad a_0 = - \sum_{j=1}^{m} A_{ij}^+, \quad a_j = A_{ij}^+, \quad 1 \leq j \leq m
\] (20)
We begin with the Euler-Maclaurin summation formula.

Remark 0.4 The order of approximation, or the rate of convergence, of (19) is generally $n+1-i$. But it is one order higher if there are more symmetry in the sampling points $\{x_j\}$ and in differentiation: when both $m$ and $n-i$ are even and when $\{x_j\}$ are symmetrically distributed about $x = x_0$.

Remark 0.5 The fact that the rate of convergence is $n+1-i$, instead of $m+1-i$, indicates that we usually just need to set $m = n$, $m$ or $m+1$ being the number of sampling points, $n$ or $n+1$ being the number of Taylor terms. Using more sampling points might have other advantages but it won’t land us a higher order of convergence. Conversely, if we are given $m+1$ sampling points, the incentive of higher rate of convergence push us to match $n$ with $m$: $n = m$.

Remark 0.6 It is tempting to expect the following, yet to be verified or disapproved. Let $m = n$ so $A$ is square. If $m$ and $i$ are both even, and if the sampling points $\{x_j\}$ are equispaced with $x = x_0$ in the center of the distribution, the scheme (20) is identical to the one using central finite difference approximation of the $i$-th derivative with the same $m+1$ sampling points. Thus, the method (19) is particularly useful for computing (i) odd order derivatives, (ii) even order derivatives with $m$ odd (iii) any order derivative with arbitrarily distributed sampling points.

1. Construct the 4-th order backward difference approximation to the second order derivative. Provide $a_j$. Must check and show rate of convergence with the test function $f(x) = e^x$ at $x = 1$ with $h = 0.1$ and $h = 0.07$.

2. Design second, fourth, and 16-th order schemes for central difference approximation to the second order derivatives. Provide $a_j$ for each case. Must check and show rate of convergence for the first two cases with the test function $f(x) = e^x$ at $x = 1$ with $h = 0.1$ and $h = 0.06$. Note: checking rate of convergence for the last case is a bit tricky because it converges very fast. It can be done with careful choice of the two $h$ values. You’re encouraged to experiment with it.

3. Given 11 equispaced sampling points with spacing $h$, write a Matlab script to design schemes for computing all possible odd order derivatives at the center point using all the 11 points. Provide $a_j$ for each case. Must check and show rate of convergence for the first three cases with the test function $f(x) = e^x$ at $x = 1$ with $h = 0.1$ and $h = 0.07$.

Note. Do any one or both of the following problems for extra credit. Extra extra credit if do both.

**Extra 1.** Consider 20 Chebyshev points $\{t_j\}$ in $[1, 2]$. Let $f(x) = e^x$. Compute numerically $f''(x)$ at all these points as accurately as you can by using only 7 values of $\{f(t_j)\}$ each time a $f''(t_j)$ is approximated. Note: $\{f(t_j)\}$ is our input, whereas $\{f''(t_j)\}$, or rather, their approximations, is our output. Each time you compute a $f''(t_j)$ for a $j$ between 1 and 20 with a finite difference scheme, the scheme can use just 7 values of $\{f(t_j)\}$. Statement of Work.

a. Show the relative error

$$y_j = \frac{f''(t_j) - \text{its approximation}}{f''(t_j)}$$

for all $\{t_j\}$ in $[1, 2]$ with a semilogy plot.

b. Provide $\|y\|_2$, the 2-norm of the relative error

c. Is there a better, fairer definition for the 2-norm of the relative error, taking into account of the non-equispacing of the Chebyshev points. Compute its value if you can.

**Extra 2.** We begin with the Euler-Maclaurin summation formula.
**Theorem 0.1** Let $a < b$ be a pair of real numbers, $m \geq 1$ be an integer. Let $B_k$ be the Bernoulli numbers

$$B_0 = 1, \quad B_k = -\frac{1}{k+1} \sum_{j=0}^{k-1} \binom{k+1}{j} B_j, \quad \text{which implies} \quad 0 = B_3 = B_5 = B_7 = \ldots \quad (22)$$

and

$$B_2 = \frac{1}{6}, \quad B_4 = -\frac{1}{30}, \quad B_6 = \frac{1}{42}, \quad \ldots, \quad B_{2m} \sim (-1)^{m+1} \frac{2(2m)!}{(2\pi)^{2m}} \quad (23)$$

If $f \in C^{2m+2}[a, b]$, then there exists a real number $\xi$, with $a < \xi < b$, such that

$$\int_a^b f(x)dx = T_n(f) - \sum_{l=1}^{m} \frac{h^{2l}B_{2l}}{(2l)!} (f^{2l-1}(b) - f^{2l-1}(a)) - (b-a)B_{2m+2} \frac{f^{(2m+2)}(\xi)}{(2m+2)!} h^{2m+2} \quad (24)$$

where $T_n$ is the trapezoidal quadrature defined by the formula

$$T_n(f) = h \left[ \sum_{j=1}^{n-1} f(x_j) + \frac{1}{2} f(a) + \frac{1}{2} f(b) \right], \quad x_j = a + jh, \quad h = (b-a)/n \quad (25)$$

Now for its application to numerical integration of $f$ with the trapezoidal rule $T_n$,

$$\int_a^b f(x)dx = T_n(f) + \text{error} \quad (26)$$

we observe that the order of $T_n$ is only 2: error is proportional to $h^2$ because its leading term is

$$\frac{h^2 B_2}{2} (f'(b) - f'(a)) \quad (27)$$

But if $f$ can also be sampled outside the interval $[a, b]$, we can raise the order by numerically computing the odd order derivatives, at $a$ and $b$, which occur in the Euler-Maclaurin formula, and adding them to $T_n(f)$ as a part of the quadrature. For example, we can obtain $f'(a)$, $f'(b)$ numerically with second order accuracy

$$f'_{\text{numer}}(a) =: \frac{f(a+h) - f(a-h)}{2h}, \quad \text{so} \quad f'(a) = f'_{\text{numer}}(a) + O(h^2) \quad (28)$$

Then letting $m = 1$ in Euler-Maclaurin we obtain

$$\int_a^b f(x)dx = T_n(f) - \frac{h^2 B_2}{2} \{ f'_{\text{numer}}(b) - f'_{\text{numer}}(a) \} + O(h^4). \quad (29)$$

Substituting $f'(a)$ and $f'(b)$ with their finite difference approximation we have

$$\int_a^b f(x)dx = T_n(f) - \frac{h^2 B_2}{2} \{ f'_{\text{numer}}(b) - f'_{\text{numer}}(a) + O(h^2) \} + O(h^4) \quad (30)$$

which gives us a modified trapezoidal quadrature

$$\int_a^b f(x)dx \approx T_n(f) - \frac{h^2 B_2}{2} \{ f'_{\text{numer}}(b) - f'_{\text{numer}}(a) \} \quad (31)$$

with the approximation error proportional to $h^4$. This is indeed a quadrature, for it computes the integral from all existing equispaced points in $[a, b]$, and two additional points outside $[a, b]$: $a - h$ and $b + h$. 
Remark 0.7 This modified trapezoidal quadrature is also known as corrected trapezoidal rules. It is not necessary to sample $f$ outside $[a, b]$ to have a 4-th order quadrature. If we use forward difference approximation for $f'(a)$, and backward for $f'(b)$, we will only require the existing equispaced points in $[a, b]$. But this approach will become increasingly unstable when we use third, fifth, and so on, derivatives in the Euler-Maclaurin formula to do the correction. So we want to use central difference to do it, to approximate higher order derivatives. That’s why we need to sample $f$ outside $[a, b]$, as well as inside.

More generally, suppose $f$ is sampled at the equispaced points

$$x_j = a + jh, \quad j = -5, \ldots, 0, 1, \ldots, n, n + 1, \ldots, n + 5$$

namely, we employ 5 extra equispaced points to the left of $a$ and right of $b$. We want to compute the integral from $f$ samples at these points, with a higher order of convergence, 11-th order to be precise. We plan to do this by numerically computing not only the first derivative at $a$ and $b$, but also the third, fifth, seventh, and ninth derivatives at $a$ and $b$, and using them to do the correction, just as what we did in (31) with the first derivative. We expect to obtain a quadrature of the form

$$\int_a^b f(x)\,dx = h \sum_{j=-5}^{n+5} w_j f(x_j) + \text{error}$$

and if all goes well, we expect the error is proportional to $h^{11}$, a 11-th order scheme for numerical integration.

Statement of Work.

a. Using the 11 points $x_j = a + jh, j = -5 : 5$ to approximate all the five odd order derivatives at $x = a$ to determine the first 11 weights $\{w_j | j = -5 : 5\}$.

b. Do the same at $x = b$, or better yet, obtain $\{w_j | j = n - 5 : n + 5\}$ from $\{w_j | j = -5 : 5\}$ trivially.

c. What do you expect the order of convergence for these approximations to be. Why do we expect to have an 11-th order quadrature.

d. Check the order of convergence for the test function $f(x) = e^x$ with $a = 0$, $b = 5$, and two values of $h$ of your own choice.

Remark 0.8 It is easy to see that all other $w_j$ is one: $w_j = 1, j = 6 : n - 6$. I believe that $w_0 = w_n = 1/2$. 