Scientific Computing Homework 1

Michael Nelson

Problem 1

Exercise 1. We consider the example from class for approximating $f'(2)$ for $f(x) = \sin x$ using a finite difference approximation. This time, we will be using

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0-h)}{2h}$$

instead. Show using Taylor's Theorem that you expect the truncation error of this approximation to be $O(h^2)$.

Solution 1. By Taylor's Theorem, there exists a real-valued function ψ defined on a neighborhood of 0 such that $\lim_{h\to 0} \psi(h) = 0$ and such that

$$f(x_0+h) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2}h^2 + \psi(h)h^2 \tag{1}$$

and

$$f(x_0-h) = f(x_0) - f'(x_0)h + \frac{f''(x_0)}{2}h^2 + \psi(-h)h^2 \tag{2}$$

Subtracting (2) from (1) and rearranging terms gives us

$$R(h) = (\psi(h) - \psi(-h))h^2,$$

where $R(h) = f'(x_0) - (f(x_0+h) - f(x_0-h))/2h$ is the truncation error we wish to approximate. We claim that $R(h) = o(|h|^2)$ as $h \to 0$ (which is much stronger than the statement that $R(h) \in O(|h|^2)$ as $h \to 0$). Indeed, let $\varepsilon > 0$. Since $\psi(h) \to 0$ as $h \to 0$, there exists $\delta > 0$ such that $|h| < \delta$ implies $|\psi(h)| < \varepsilon/2$. In particular, we see that $|h| < \delta$ implies

$$\begin{aligned}
|R(h)| &= \left| f'(x_0) - \frac{f(x_0+h) - f(x_0-h)}{2h} \right| \\
&= \left| (\psi(h) - \psi(-h))h^2 \right| \\
&\leq |\psi(h)||h^2| + |\psi(-h)||h^2| \\
&< \frac{\varepsilon}{2}|h|^2 + \frac{\varepsilon}{2}|h|^2 \\
&= \varepsilon |h|^2.
\end{aligned}$$

It follows that $R(h) = o(|h|^2)$ as $h \to 0$.

Problem 2

Exercise 2. Modify the MATLAB example from the lecture to use the approximation in 1) and produce a convergence table (include a third column with the computation of the rate by dividing two consecutive errors, respectively). Finally, show the error plot in log-log scale and include a second line defined by h^2 to confirm the quadratic behavior.

Solution 2. We give the code below

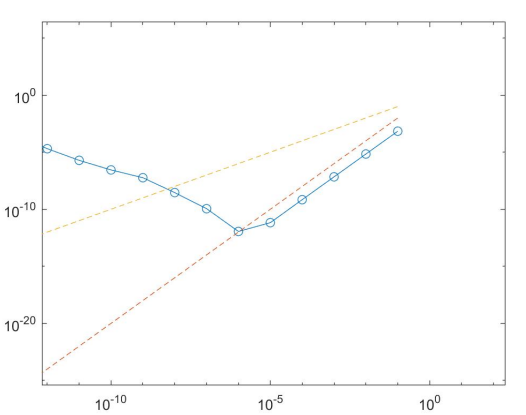
```
format longg

f= @(x) sin(x);
x0 = 2;
ref = cos(2);
hs = 10.^(-(1:16));
errors = [];
derivatives = [];
errorrates=[];
for h = hs
        derivative = (f(x0+h)-f(x0-h))/(2*h);
        error = abs(ref - derivative);
        derivatives = [derivatives derivative];
        errors = [errors error];
end
for i=1:15
        errorrate = errors(i+1)/errors(i)
        disp([i errorrate]);
        errorrates = [errorrates errorrate];
end
for i=1:15
        disp([i derivatives(i) errors(i) errorrates(i)]);
end
hs2 = 10.^(-(2:2:32));
loglog(hs,errors,'o-',hs,hs2, '--',hs,hs,'--')
```

The convergence table which MATLAB outputs is given below:

1	−0.415453605192704	0.000693231354438606	0.0100049512877535
2	−0.416139900801 21	6.935745932301 62e−06	0.0100000526090672
3	−0.416146767189318	6.935782420613 99e−08	0.0099886712788655
4	−0.416146835854336	6.928060902744 45e−10	0.0096537914992886 9
5	−0.416146836540454	6.688205544946 87e−12	0.170022575611699
6	−0.416146836546005	1.137145932972 32e−12	98.6324139614352
7	−0.416146836434983	1.121594483954 88e−10	25.7465336293019
8	−0.416146833659425	2.887717009958 38e−09	20.1455160404598
9	−0.416146894721692	5.817454934442 52e−08	4.81686850052599
10	−0.416147116766297	2.802191542694 57e−07	6.92396242539031
11	−0.416144896320247	1.940226895036 37e−06	10.4442597148631
12	−0.416167100780 74	2.026423359752 23e−05	18.1755615502931
13	−0.415778522722121	0.000368313825021283	0.507169904036325
14	−0.416333634234434	0.000186797687291296	147.586291501278
15	−0.388578058618805	0.0275687779283377	15.0948597587051

The plot which MATLAB outputs is given below:



Problem 3

Exercise 3. On paper, compute the binary representation of the decimal number 0.1 and show that the representation is infinite (that you need an infinite number of binary digits to express 0.1 exactly). Are there binary numbers with a finite number of digits that only have an infinite representation in decimal? Why or why not?

Solution 3. Observe that

$$\begin{aligned}\frac{1}{10} &= \frac{1}{15} + \frac{1}{30} \\ &= \left(\frac{16}{15} - 1\right) + \frac{1}{2}\left(\frac{16}{15} - 1\right) \\ &= \left(\frac{1}{1-2^{-4}} - 1\right) + \frac{1}{2}\left(\frac{1}{1-2^{-4}} - 1\right) \\ &= \sum_{n=1}^{\infty} 2^{-4n} + \sum_{n=1}^{\infty} 2^{-4n-1}\end{aligned}$$

Thus a binary represenation of $1/10$ starts out as

$$0.00011000110001100011\ldots$$

Notice that this is not the only binary representation of $1/10$. Indeed, since

$$\begin{aligned}\frac{1}{10} &= \sum_{i=1}^{\infty} 2^{-4i} + \sum_{i=1}^{\infty} 2^{-4i-1} \\ &= 2^{-4} + \sum_{i=2}^{\infty} 2^{-4i} + \sum_{i=1}^{\infty} 2^{-4i-1} \\ &= \sum_{i=5}^{\infty} 2^{-i} + \sum_{i=2}^{\infty} 2^{-4i} + \sum_{i=1}^{\infty} 2^{-4i-1},\end{aligned}$$

we see (after combining terms) that another binary representation of $1/10$ starts out as

$$0.00010110101101011110\ldots$$

However note that all binary representations of $1/10$ are infinite. Indeed, assume for a contradiction that $1/10$ has a finite binary representation. Then this binary representation must have the form $0.a_1 \cdots a_n$ since $1/10 < 1$ and where we may assume that $a_n \neq 0$. Then

$$\begin{aligned}\frac{1}{10} &= \sum_{i=1}^{n} a_i 2^{-i} \\ &= 2^{-n}\left(\sum_{i=1}^{n} a_i 2^{n-i}\right)\end{aligned}$$

implies

$$10\left(\sum_{i=1}^{n} a_i 2^{n-i}\right) = 2^n, \tag{3}$$

which is a contradiction since 5 divides the left-hand side of (3) but does not divide the right-hand side of (3).

On the other hand, if a number has a finite binary representation, then it necessarily has a finite decimal representation. To see this, suppose $\sum_{i=-m}^{n} a_i 2^i$ is a number which has a finite binary representation. Note that the sum of two finite decimal representations is a finite decimal representation and the product of two finite decimal representations is a finite decimal representation. Thus since $1/2$ has a finite decimal representation ($1/2 = 0.5$), it follows that $2^{-i} = (1/2)^i$ has a finite decimal representation for each $1 \leq i \leq m$. In particular, $\sum_{i=-m}^{n} a_i 2^i$ has a finite decimal representation since each term $a_i 2^i$ in its sum has a finite decimal representation.

Problem 4

Exercise 4. Write a MATLAB function function B=tobinary(n) that converts a positive, whole number n into binary format. Here, B is an array of appropriate length that contains values of 1 and 0. You may use log2(n) once to figure out the largest power of two that fits into n. After that, only use simple arithmetic and comparisons (ignore that there are more efficient routines in MATLAB, that could help you here).

Solution 4. We design an algorithm as follows: let n be a positive integer. Using the fact that $\mathbb{Z}$ is a Euclidean domain with respect to the usual absolute value, we can express n as

$$n = q_1 2 + r_0 \tag{4}$$

where $0 \leq q_1 < n$ and $r_0 \in \{0, 1\}$. If $q_1 < 2$, then from (4) we see that the string $q_1 r_0$ gives a binary representation for n, and so the algorithm terminates at the 0th step with the output given in the form of an array as $[q_1, r_0]$. If $q_1 \geq 2$, then we proceed to the 1st step of the algorithm and express q_1 as

$$q_1 = q_2 2 + r_1$$

where $0 \leq q_2 < q_1$ and $r_1 \in \{0, 1\}$. If $q_2 < 2$, then since

$$\begin{aligned} n &= q_1 2 + r_0 \\ &= (q_2 2 + r_1) 2 + r_0 \\ &= q_2 2^2 + r_1 2 + r_0, \end{aligned}$$

we see that the string $q_2 r_1 r_0$ gives a binary representation for n, and so the algorithm terminates at the 1st step with the output in the form of an array as $[q_2, r_1, r_0]$. If $q_2 \geq 2$, then we proceed to the 2nd step. At the ith step of this algorithm, where $i \geq 1$ and $q_{i-1} \geq 2$, we express q_i as

$$q_i = q_{i+1} 2 + r_i$$

where $0 \leq q_{i+1} < q_i$ and $r_i \in \{0, 1\}$. Since the sequence (q_i) is a strictly decreasing sequence of positive integers, there must exist a $k \in \mathbb{N}$ such that $q_{k+1} < 2$ and $q_i \geq 2$ for all $0 \leq i \leq k$. In this case, using induction, we see that

$$\begin{aligned} n &= q_1 2 + r_0 \\ &= \left(q_{k+1} 2^k + \sum_{i=0}^{k-1} r_{i+1} 2^i \right) 2 + r_0 \\ &= q_{k+1} 2^{k+1} + \sum_{i=0}^{k-1} r_{i+1} 2^{i+1} + r_0 \\ &= q_{k+1} 2^{k+1} + \sum_{i=0}^{k} r_i 2^k. \end{aligned}$$

Thus the string $q_{k+1} r_k \cdots r_0$ gives a binary representation for n, and so the algorithm terminates at the kth step with the output in the form of an array as $[q_{k+1}, r_k, \ldots, r_0]$. The code which performs this algorithm is given below:

```
function B = tobinary(n)

        i = 1;
        q = floor(n/2);
        r = n - 2*q;
        B(1) = r;
        while 2 <= q
                n = q;
                i = i + 1;
                q = floor(n/2);
                r = n - 2*q;
                B(i) = r;
        end
        B(i + 1) = q;
        B = fliplr(B);
end
```