CSC 336 - Fall 2021 - Assignment 1

Songheng Yin, Student Id: 1004762303

1 Question 1.

1.1 1(a)

$$f'(x) = \frac{xe^x - e^x}{x^2}$$

The relative condition number:

$$\begin{aligned}\kappa_f &= |\frac{xf'(x)}{f(x)}| \\ &= |\frac{xe^x - e^x + 1}{x} \frac{x^2}{xe^x - x}| \\ &= |\frac{xe^x - e^x + 1}{e^x - 1}|\end{aligned}$$

Consider $g(x) = e^x - x - 1$

$$\frac{\mathrm{d}g}{\mathrm{d}x} = e^x - 1$$

has root at $x = 0$, so that $g(x)$ attains its minimum when $x = 0$. That is,

$$g(x) = e^x - x - 1 \geq e^0 - 1 = 0$$

Ignore the absolute value symbol first, we have

$$\begin{aligned}\frac{\mathrm{d}\kappa_f}{\mathrm{d}x} &= \frac{e^x(e^x - x - 1)}{(e^x - 1)^2} \\ &\geq 0\end{aligned}$$

Therefore, the function inside the absolute value symbol of κ is non-decreasing.
By L'Hôpital's rule,

$$\begin{aligned}\lim_{x \to 0} \kappa_f(x) &= \lim_{x \to 0} \frac{e^x + xe^x - e^x}{e^x} \\ &= 0 \\ \lim_{x \to -\infty} \kappa_f(x) &= -1 \\ \lim_{x \to \infty} \kappa_f(x) &= \infty\end{aligned}$$

Therefore, the relative condition number keeps increasing and goes large for large posivite x.

1.2 1(b)

By Taylor's Theorem,

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

So

$$\begin{aligned} f(x) &= \frac{xe^x - x}{x^2} \\ &= \frac{e^x - 1}{x} \\ &= \frac{x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots}{x} \\ &= 1 + \frac{x}{2!} + \frac{x^2}{3!} + \dots \end{aligned} \tag{1}$$

We use equation (1), denoted as $g(x)$, to compute the approximation value of f. More specifically, we compute the sum from left to right up to saturation, using the fact that each term added is the previous term times $\frac{x}{k+1}$.
Justification is the same as question 8 from tutorial 2 (ways to compute e^x). That is, since x is positive, $g(x)$ avoids catastrophic cancellation. On account of every step in the calculation is well-conditioned, the algorithm is stable.
The condition number of g is

$$\begin{aligned} \kappa_g &= |\frac{xg'(x)}{g(x)}| \\ &= |\frac{x(\frac{1}{2!} + \frac{2x}{3!} + \dots)}{f(x)}| \\ &= |\frac{xe^x - e^x + 1}{e^x - 1}| \end{aligned}$$

By L'Hôpital's rule,

$$\begin{aligned} \lim_{x \to 0} \kappa_g &= \lim_{x \to 0} \frac{e^x + xe^x - e^x}{e^x} \\ &= 0 \end{aligned}$$

1.3 1(c)

Let $y = -x$, clearly y is a positive number close to 0.

$$\begin{aligned} f(x) &= \frac{e^x - 1}{x} \\ &= (\frac{1}{e^y} - 1)\frac{1}{-y} \\ &= \frac{e^y - 1}{y} \cdot \frac{1}{e^y} \end{aligned}$$

Notice $\frac{e^y-1}{y}$ can be computed in the exactly same way as 1(b), and then by compute $e^y = 1 + y + \frac{y^2}{2!} + \dots$ from left to right up to saturation. Finally, divide the value of $\frac{e^y-1}{y}$ by the value of e^y.
The algorithm is stable because y is positive so that e^y can be computed without the cancellation issue. Also, we avoid any subtraction between two almost equal numbers so that every step is well-conditioned.

2 Question 2.

2.1 2(a)

$$\begin{aligned}\text{Truncation Error} &= \left| -\frac{h}{2} f''(\xi) \right| \\ &\leq \frac{M_2 h}{2}\end{aligned}$$

for some ξ near x, given the definition of M_2.

$$\begin{aligned}\text{Computation Error} &= \text{Truncation Error} + \text{Rounding Error} \\ &\leq \frac{M_2 h}{2} + 5\frac{\epsilon}{h} \qquad (2)\end{aligned}$$

Let the derivative of equation (2) to h equals to 0,

$$\begin{aligned}\frac{M_2}{2} - \frac{5\epsilon}{h^2} &= 0 \\ h^* &= \sqrt{\frac{10\epsilon}{M_2}}\end{aligned}$$

So the bound for the total computation error in g_a is minimized when $h = \sqrt{\frac{10\epsilon}{M_2}}$.

2.2 2(b)

Consider the following Taylor expansions

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \dots \qquad (3)$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{6}h^3 + \dots \qquad (4)$$

Hence

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{f'''(x)}{6}h^2 + \dots$$

which gives the approximation

$$f'(x) \approx g_b(x; h; f) \equiv \frac{f(x+h) - f(x-h)}{2h}$$

Similar as (a),

$$\begin{aligned}\text{Truncation Error} &= |\frac{f'''(x)}{6}h^2| \\ &\leq \frac{M_3 h^2}{6} \\ \text{Computation Error} &= \frac{M_3 h^2}{6} + \frac{5\epsilon}{2h}\end{aligned}$$

Set the derivative to 0,

$$\begin{aligned}\frac{M_3 h}{3} - \frac{5\epsilon}{2h^2} &= 0 \\ h^* &= \sqrt[3]{\frac{15\epsilon}{2M_3}}\end{aligned}$$

The bound for the total computation error reaches its minimum when $h = \sqrt[3]{\frac{15\epsilon}{2M_3}}$.

2.3 2(c)

Add equation (3) and (4) gives

$$\begin{aligned}f(x+h) + f(x-h) &= 2[f(x) + \frac{f''(x)}{2}h^2 + \frac{f''''(x)}{24}h^4 + \dots] \\ f''(x) &\approx \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} \\ &\equiv g_c(x; h; f)\end{aligned}$$

Similar as (b),

$$\begin{aligned}\text{Truncation Error} &= |\frac{f''''(x)}{12}h^2| \\ &\leq \frac{M_4 h^2}{12} \\ \text{Computation Error} &= \frac{M_4 h^2}{12} + \frac{6\epsilon}{h^2}\end{aligned}$$

Set its derivative to h to 0,

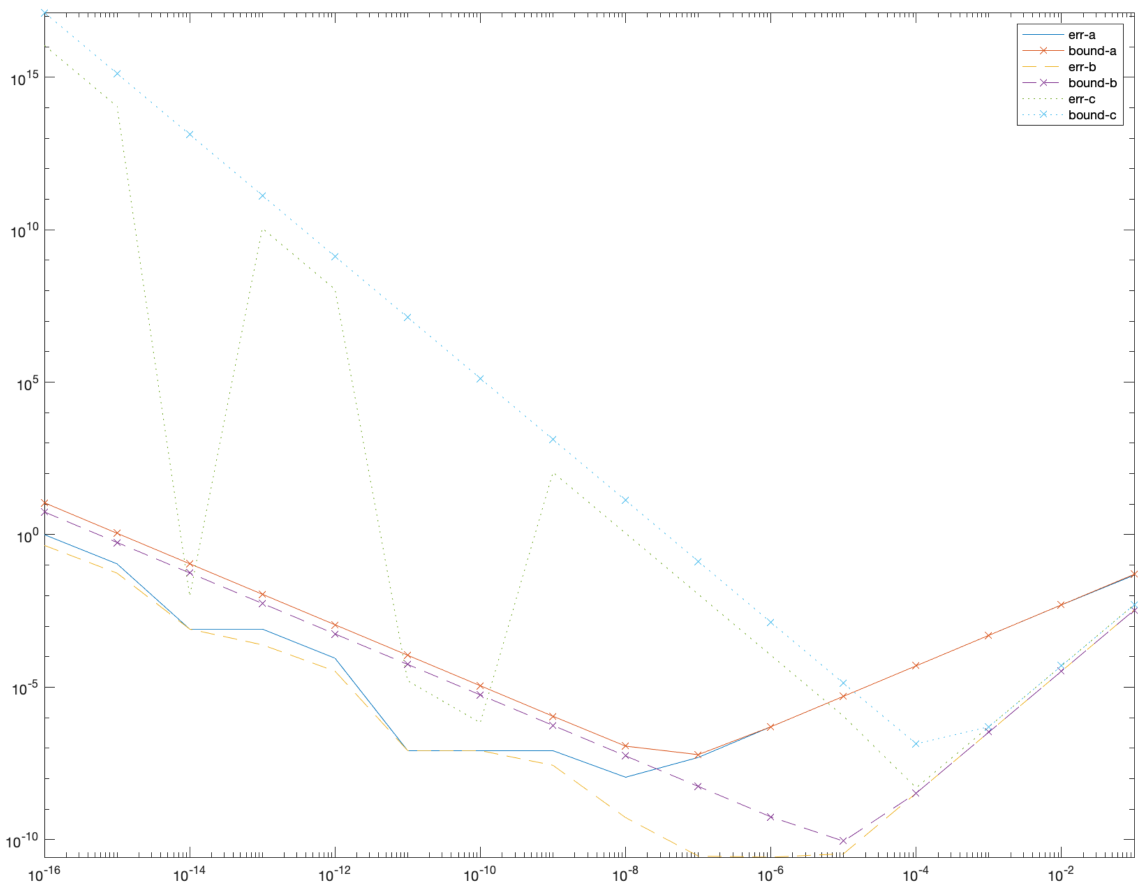
$$\begin{aligned}\frac{M_4 h}{6} - \frac{12\epsilon}{h^3} &= 0 \\ h^* &= \sqrt[4]{\frac{72\epsilon}{M_4}}\end{aligned}$$

The bound for the total computation error reaches its minimum when $h = \sqrt[4]{\frac{72\epsilon}{M_4}}$.

2.4 2(d)

Since $f''(x) = -\frac{1}{x^2}, f'''(x) = -\frac{2}{x^3}, f''''(x) = -\frac{6}{x^4}$.
At point $x = 1$, get the approximated M values by

$$M_2 \approx |f''(1)| = 1$$
$$M_3 \approx |f'''(1)| = 2$$
$$M_4 \approx |f''''(1)| = 6$$



MATLAB source code:

```
% ln'(1) = 1,  ln''(1) = -1
h = 10.^ (-16:-1);
x = ones(1, 16);

g_a = (log(x + h) - log(x)) ./ h;
```

```
g_b = (log(x + h) - log(x - h)) ./ (2 * h);
g_c = (log(x + h) + log(x - h) - 2 * log(x)) ./ (h.^2);
err_a = (1 - g_a) / 1;
err_b = (1 - g_b) / 1;
err_c = (-1 - g_c) / (-1);

epsilon = eps .* ones(1, 16);
m = ones(1, 16);
bound_a = (m .* h) / 2 + 5 * epsilon ./ h;
bound_b = (2 * m) .* (h.^2) / 6 + 5 * epsilon ./ (2 * h);
bound_c = (6 * m) .* (h.^2) / 12 + 6 * epsilon ./ (h.^2);

loglog(h, abs(err_a), '-', h, bound_a, 'x-', ...
    h, abs(err_b), '--', h, bound_b, 'x--', ...
    h, abs(err_c), ':', h, bound_c, 'x:');
legend('err-a', 'bound-a', 'err-b', 'bound-b', 'err-c', 'bound-c');
axis tight;

[~, index_a] = min(abs(err_a));
[~, index_b] = min(abs(err_b));
[~, index_c] = min(abs(err_c));

fprintf('Min Error in case (a): stepsize = %9.2e, error = %9.2e\n', ...
    h(index_a), err_a(index_a));
fprintf('Min Error in case (b): stepsize = %9.2e, error = %9.2e\n', ...
    h(index_b), err_b(index_b));
fprintf('Min Error in case (c): stepsize = %9.2e, error = %9.2e\n', ...
    h(index_c), err_c(index_c));
```

Outputs:

- Min Error in case (a): stepsize $= 1.00e - 08,$ error $= 1.11e - 08$
- Min Error in case (b): stepsize $= 1.00e - 06,$ error $= 2.64e - 11$
- Min Error in case (c): stepsize $= 1.00e - 04,$ error $= -5.00e - 09$

Comments: It can be verified that these stepsizes are the very close to the minima bound points in (a), (b), (c). The maxima error bound occurs when h is smallest or largest, which is consistent as the derivative functions derived in previous questions.
Both (a) and (b) are approximation of f' while algorithm (b) gives more accurate results, this is a proof of $\mathcal{O}(h^2)$ error is better than $\mathcal{O}(h)$ error.

3 Question 3.

C language source code:

```
#include <stdio.h>
#include <math.h>

void run_single_precision() {
    float eapprox, n;
    for (int i = 0; i <= 12; i++) {
        n = pow(10, i);
        eapprox = pow((1 + 1 / n), n);
```

```
        printf("%13.0f %13.10f %13.10f %11.3e %14.11f\n",
            n,
            eapprox,
            exp(1),
            (exp(1) - eapprox) / exp(1),
            (1 + 1 / n)
        );
    }
}

void run_double_precision() {
    double eapprox, n;
    for (int i = 0; i <= 12; i++) {
        n = pow(10, i);
        eapprox = pow((1 + 1 / n), n);
        printf("%13.0f %13.10f %13.10f %11.3e %14.11f\n",
            n,
            eapprox,
            exp(1),
            (exp(1) - eapprox) / exp(1),
            (1 + 1 / n)
        );
    }
}

int main() {
    run_single_precision();
    printf("\n");
    run_double_precision();
    return 0;
}
```

Table 1: Output given by Single Precision

i	eapprox	e	Relative Err	$1+\frac{1}{n}$
0	2.0000000000	2.7182818285	2.642e-01	2.00000000000
1	2.5937430859	2.7182818285	4.582e-02	1.10000002384
2	2.7048113346	2.7182818285	4.956e-03	1.00999999046
3	2.7170507908	2.7182818285	4.529e-04	1.00100004673
4	2.7185969353	2.7182818285	-1.159e-04	1.00010001659
5	2.7219622135	2.7182818285	-1.354e-03	1.00001001358
6	2.5952267647	2.7182818285	4.527e-02	1.00000095367
7	3.2939677238	2.7182818285	-2.118e-01	1.00000011921
8	1.0000000000	2.7182818285	6.321e-01	1.00000000000
9	1.0000000000	2.7182818285	6.321e-01	1.00000000000
10	1.0000000000	2.7182818285	6.321e-01	1.00000000000
11	1.0000000000	2.7182818285	6.321e-01	1.00000000000
12	1.0000000000	2.7182818285	6.321e-01	1.00000000000

Table 2: Output given by Double Precision

i	eapprox	e	Relative Err	$1+\frac{1}{n}$
0	2.0000000000	2.7182818285	2.642e-01	2.00000000000
1	2.5937424601	2.7182818285	4.582e-02	1.10000000000
2	2.7048138294	2.7182818285	4.955e-03	1.01000000000
3	2.7169239322	2.7182818285	4.995e-04	1.00100000000
4	2.7181459268	2.7182818285	5.000e-05	1.00010000000
5	2.7182682372	2.7182818285	5.000e-06	1.00001000000
6	2.7182804691	2.7182818285	5.001e-07	1.00000100000
7	2.7182816941	2.7182818285	4.942e-08	1.00000010000
8	2.7182817983	2.7182818285	1.108e-08	1.00000001000
9	2.7182820520	2.7182818285	-8.224e-08	1.00000000100
10	2.7182820532	2.7182818285	-8.269e-08	1.00000000010
11	2.7182820534	2.7182818285	-8.274e-08	1.00000000001
12	2.7185234960	2.7182818285	-8.890e-05	1.00000000000

Comments: When using the double precision, the algorithm gives little relative error, one reason is that the $1+\frac{1}{n}$ is very accurate. However for the single precision, the $1+\frac{1}{n}$ equals to 1 for all $n \geq 8$ so that its power will be equal to 1.0 as well, leading to the failure of the calculation. Even for small n, due to the less accuracy, it has larger relative error than the double precision.