Problem Set 1

Congyue Cui ccui@princeton.edu

1 Error in (symmetric) rounding vs. chopping

$x = 1.b_2 \dots b_p b_{p+1} \dots \cdot 2^e$

If $b_{p+1} = 0$:

$\mathrm{rd}(x) = 1.b_2 \dots b_p \cdot 2^e$

$$\left|\frac{x - \mathrm{rd}(x)}{x}\right| = \left|\frac{\underbrace{0.000 \dots 0}_{p \text{ zeros}} b_{p+1} \dots \cdot 2^e}{1.b_2 \dots b_p b_{p+1} \dots \cdot 2^e}\right|$$
$$\leq \left|\frac{\underbrace{0.000 \dots 0}_{p+1 \text{ zeros}} \dots}{1.000 \dots 0 \dots}\right|$$
$$\leq \underbrace{0.000 \dots 0}_{p+1 \text{ zeros}} 111 \dots$$
$$= \underbrace{0.000 \dots 0}_{p \text{ zeros}} 1$$
$$= 2^{-p}$$

If $n_{p+1} = 1$:

$\mathrm{rd}(x) = \left(1.b_2 \dots b_p + 2^{-p+1}\right) \cdot 2^e$

$$\left|\frac{x - \mathrm{rd}(x)}{x}\right| = \left|\frac{\underbrace{0.000 \dots 0}_{p \text{ zeros}} 1 \dots - \underbrace{0.000 \dots 0}_{p \text{ zeros}} 1 \cdot 2}{1.b_2 \dots b_p b_{p+1} \dots \cdot 2^e}\right|$$
$$\leq \left|\frac{\underbrace{0.000 \dots 0}_{p \text{ zeros}} 1 - \underbrace{0.000 \dots 0}_{p \text{ zeros}} 1 \cdot 2}{1.000 \dots 1 \dots}\right|$$
$$< \underbrace{0.000 \dots 0}_{p \text{ zeros}} 1$$
$$= 2^{-p}$$

2 An accurate implementation of e^x

(a) 244.71
(b) Convergence: $k = 17$ Relative error: 7.3839×10^{-5}
(c) Convergence: $k = 16$ Relative error: 7.3839×10^{-5}
(d) Method ii converges most quickly and has the lowest error.
Not as quick or accurate compared to positive case.
- i. Convergence: $k = 25$ Relative error: 6.1288×10^{-2}
- ii. Convergence: $k = 19$ Relative error: 2.1232×10^{-2}
- iii. Not convergent due to comparison of large number
$e^x = 122.35 - 122.35 = 0$
- iv. Not convergent due to comparison of large number
$e^x = 122.34 - 122.34 = 0$

(e) $e^{-5.5} = 1/e^{5.5}$, sum from right to left
Convergence: $k = 16$ Relative error: 6.6419×10^{-5}

3 Error propagation in exponentiation

(a) Repeated multiplication

$$\begin{aligned} \mathrm{fl}\,(x^n) &= \mathrm{fl}(x^{n-1})x(1+\varepsilon) \\ &= \mathrm{fl}(x)^{n-2}x^2(1+\varepsilon)^2 \\ &= x^n(1+(n-1)\varepsilon) \end{aligned}$$

Log-exponential

$$\mathrm{fl}\big(\ln(x)\big) = \ln(x)(1+\varepsilon)$$

$$\mathrm{fl}\big(n\ln(x)\big) = n\ln(x)(1+\varepsilon)^2$$

$$\begin{aligned} \mathrm{fl}\big(e^{n\ln(x)}\big) &= e^{n\ln(x)}(1+\varepsilon)e^{2n\ln(x)\varepsilon} \\ &= x^n(1+\varepsilon+2n\ln(x)\varepsilon) \end{aligned}$$

Use repeated multiplication when $n\big(1-2\ln(x)\big) < 2$
Use log-exponential when $n\big(1-2\ln(x)\big) > 2$

(b) $\varepsilon\big(e^{a\ln(x)}\big) = \Delta\big(a\ln(x)\big)$
- i. $\Delta\big(a\ln(x)\big) = a\varepsilon_a\ln(x)$ substantial when either a or x is large.
- ii. $\Delta\big(a\ln(x)\big) = a\varepsilon_x$ substantial when a is large.

4 Conditioning

(a) $(\mathrm{cond}f)(x) = \left|\frac{xe^{-x}}{1-e^{-x}}\right|$

$g(x) = xe^{-x} - (1 - e^{-x})$

$g'(x) = -xe^{-x} < 0 \quad x \in [0,1]$

$g(x) \le g(0) = 0 \quad x \in [0,1]$

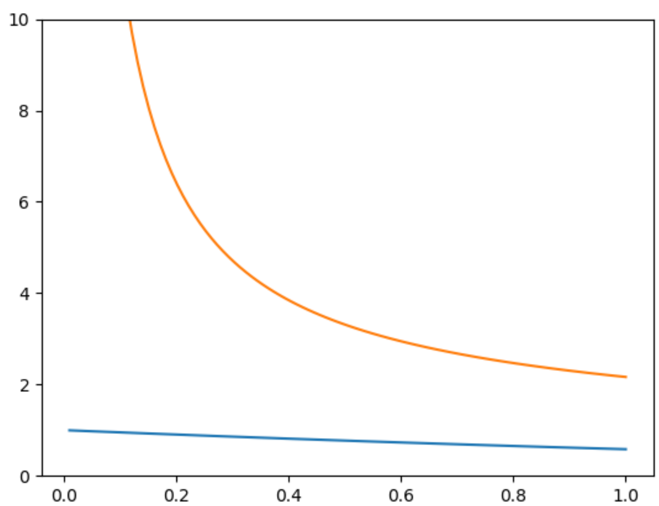
$0 < xe^{-x} \le 1 - e^{-x} \quad x \in [0,1]$

$(\mathrm{cond}f)(x) \le 1 \quad x \in [0,1]$

(b) $(x_A - x)f'(x) = \Delta f(x)$

$(\mathrm{cond}A)(x) = \left|\frac{x_A - x}{x}\right| = \left|\frac{(1-e^{-x})+e^{-x}(1+x)}{xe^{-x}}\right| = \left|\frac{1+xe^{-x}}{xe^{-x}}\right| > 1 \quad x \in [0,1]$

(c) $f(x)$ is infinitely close to zero when x is near zero, while the absolute error of $A(x)$ cannot be infinitely close to zero due to machine rounded inaccuracy.



(d) $\left|\frac{x_A - x}{x}\right| = 2^n$

1 bit: $\left|\frac{1+x_{\min}e^{-x_{\min}}}{xe^{-x_{\min}}}\right| = 2$ no solution

2 bit: $\left|\frac{1+x_{\min}e^{-x_{\min}}}{xe^{-x_{\min}}}\right| = 4 \quad x_{\min} \approx 0.619$

3 bit: $\left|\frac{1+x_{\min}e^{-x_{\min}}}{xe^{-x_{\min}}}\right| = 8 \quad x_{\min} \approx 0.169$

4 bit: $\left|\frac{1+x_{\min}e^{-x_{\min}}}{xe^{-x_{\min}}}\right| = 16 \quad x_{\min} \approx 0.072$

(e) $\Delta f = (1 + xe^{-x})\varepsilon$
4 bit: $\Delta f = 2.333\varepsilon$
8 bit: $\Delta f = 2.143\varepsilon$
16 bit: $\Delta f = 2.067\varepsilon$

(f) Compute $g(x) = x + f(x)$
Then $f(x) = g(x) - x$

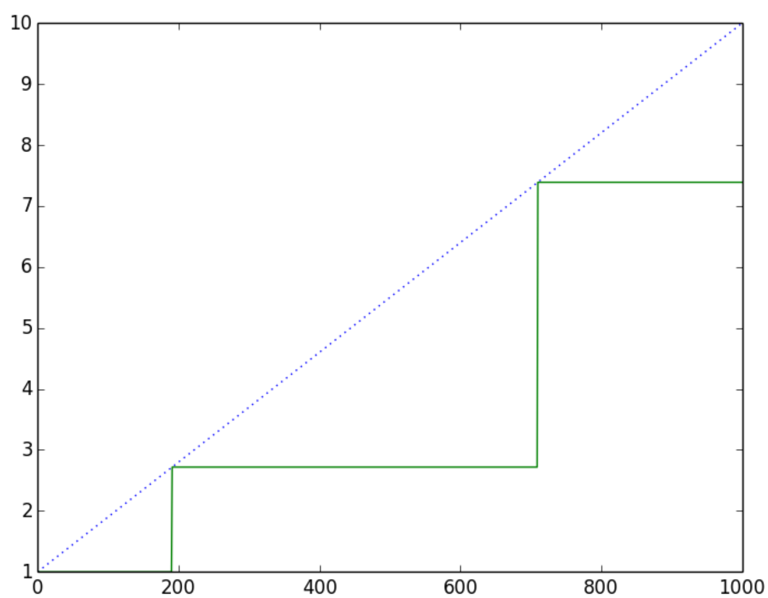
5 Limits in $\mathbb{R}(\boldsymbol{p}, \boldsymbol{q})$

$n_{stop} = 1e13$
$e_{final} = 2.71611003409$

If n is double and has 17 significant digits, the relative error of ε_e is in the order of 1×10^{-4}, which means that only the first 3 digits are accurate.

(Code is in appendix 2)

6 Fun with square roots



Whether we can end up with the starting array depends on the accuracy of $\sigma = (x-1) = x^{1/2^{52}} - 1$.
When $x_i < 1.19$, σ_i is rounded to 0, $x_i' = 1$
When $1.19 < x_i < 1.72$, σ_i is rounded to 2^{-52}, $x_i' = e$
When $x_i > 1.72$, σ_i is rounded to $2 * 2^{-52}$
The double number near 1 can only be represented in the form $1 + n * 2^{-52}$

7 The issue with polynomial roots

(a)

$$\begin{aligned} w(x) = x^{20} - 210x^{19} + 20615x^{18} \\ -1256850x^{17} \\ +53327946x^{16} \\ -1672280820x^{15} \\ +40171771630x^{14} \\ -756111184500x^{13} \\ +11310276995381x^{12} \\ -135585182899530x^{11} \\ +1307535010540395x^{10} \\ -10142299865511450x^{9} \\ +63030812099294896x^{8} \\ -311333643161390640x^{7} \\ +1206647803780373360x^{6} \\ -3599979517947607200x^{5} \\ +8037811822645051866x^{4} \\ -12870931245150988800x^{3} \\ +13803759753640704000x^{2} \\ -8752948036761600000x \\ +2432902008176640000 \end{aligned}$$

(b) Converge to 19.999981106294786
(Using scipy.optimize.newton)

(c) $\delta = 10^{-8} \quad root = 9.585$
$\delta = 10^{-6} \quad root = 7.753$
$\delta = 10^{-4} \quad root = 5.969$
$\delta = 10^{-2} \quad root = 5.470$

(d) $root_{16} = 8.917$
$root_{17} = 8.007$

(e)

i. $p(\Omega_k, a_l) = 0$

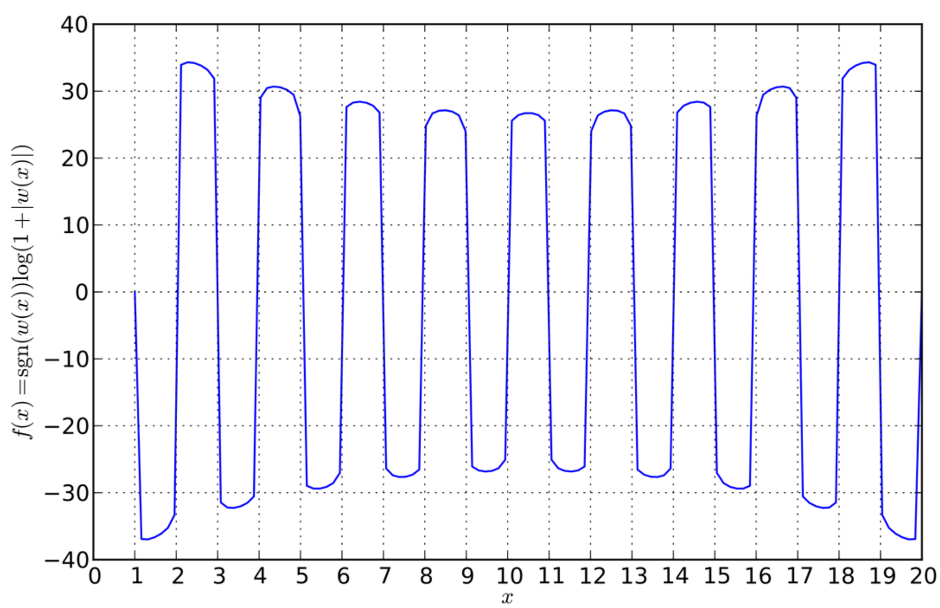
$p'(\Omega_k)\delta\Omega + p'(a_l)\delta a_l = 0$

$$\mathrm{T}_{kl} = \frac{1}{\varepsilon}\left|\frac{\delta\Omega_k}{\Omega_k}\right| = \frac{1}{\varepsilon}\left|\frac{p'(a_l)\delta a_l}{\Omega_k p'(\Omega_k)}\right| = \left|\frac{(\Omega_k)^{l-1}a_l}{p'(\Omega_k)}\right|$$

$$(\mathrm{cond}\Omega) = \sum_{l=0}^{n-1}\left|\frac{(\Omega_k)^{l-1}a_l}{p'(\Omega_k)}\right|$$

ii. $r = 14 \quad \mathrm{cond}\, r = 5.420065 \times 10^{13}$
$r = 16 \quad \mathrm{cond}\, r = 3.545858 \times 10^{13}$
$r = 17 \quad \mathrm{cond}\, r = 1.813358 \times 10^{13}$
$r = 20 \quad \mathrm{cond}\, r = 1.378071 \times 10^{11}$

iii. Convert $w(x)$ to $\mathrm{sgn}\big(w(x)\big)\ln(1 + |w(x)|)$



(Source: Wikipedia)

8 Recurrence in reverse

(a) $|\Delta y_n| = \frac{1}{(n+1)} |\Delta y_{n+1}|$

$|\Delta y_k| = \dfrac{k!}{N!} |\Delta y_N|$

$(\text{cond } g_k)(y_N) = \left|\dfrac{\Delta y_k}{\Delta y_N}\right| \leq \dfrac{k!}{N!}$

(b) $N! \geq k! \;/\; \varepsilon$

(c) $N! \geq 20! \times 2^{53}$

$N_{\min} = 32$

(d) Using recurrence relation: $y_{20} = 0.12380383076256993$

Using scipy.integrate.quad: $y_{20} = 0.12380383076256998 \pm 1.68e - 11$

(Code is in appendix 4)

Appendix

1 Code for Q2

```python
from math import log10, floor, exp

def round5(x):    # reduce to 5 significant figures
    return round(x, 4-int(floor(log10(abs(x)))))

def exp5(x, n, order=0):
    if order == 0:    # left to right
        return exp5_l2r(x, n)

    if order == 1:    # right to left
        return exp5_r2l(x, n)

    if order == 2:    # positive left to right, negative left to right
        return exp5_pl2r(x, n)

    if order == 3:    # positive right to left, negative right to left
        return exp5_pr2l(x, n)

def xon(x, i):    # calculate x^n/n!
    xi = 1.0    # xi: x^n
    ni = 1.0    # ni: n!
    for j in range(1, i + 1):
        xi = round5(xi * x)
        ni = round5(ni * j)

    return round5(xi / ni)

def exp5_l2r(x, n):    # calculate e^x
    x = round5(x)    # ensure x is within 5 significant figures
    ex = 0    # ex: e^x
    for i in range(n + 1):
        ex = round5(ex + xon(x, i))

    return ex

def exp5_r2l(x, n):    # calculate e^x
    x = round5(x)
    ex = 0
    for i in range(n, -1, -1):
        ex = round5(ex + xon(x, i))
```

```
    return ex

def exp5_pl2r(x, n):    # calculate e^x
    x = round5(x)
    pex = 0    # positive sum
    nex = 0    # negative sum

    for i in range(0, n + 1):
        term = xon(x, i)
        if term > 0:
            pex = round5(pex + term)
        else:
            nex = round5(nex + term)

    return round5(pex + nex)

def exp5_pr2l(x, n):    # calculate e^x
    x = round5(x)
    pex = 0    # positive sum
    nex = 0    # negative sum

    for i in range(n, -1, -1):
        term = xon(x, i)
        if term > 0:
            pex = round5(pex + term)
        else:
            nex = round5(nex + term)

    return round5(pex + nex)

def int5(x):    # convert float to 5-digit integer for safe comparison
    if abs(x) < 1e-5:
        return 0

    ndigit = int(floor(log10(abs(x))))

    if ndigit < 4:
        for i in range(4 - ndigit):
            x *= 10

    return int(x)


def equal5(x1, x2):    #compare two floating point numbers
    return int5(x1) == int5(x2)
```

```python
def converge(x, kmax, order=0):    # calculate number of terms to converge
    prev = 1
    for k in range(1, kmax + 1):
        ex = exp5(x, k, order)
        if equal5(ex, prev):
            return k - 1
        else:
            prev = ex

    return kmax

def converge2(x, kmax, order=0):
    prev = 1
    for k in range(1, kmax + 1):
        ex = exp5(x, k, order)
        if equal5(1 / ex, prev):
            return k - 1
        else:
            prev = 1 / ex

    return kmax

def rerr(x, n, order=0):    # calculate relative error
    return abs((exp(x) - exp5(x, n, order)) / exp(x))

def rerr2(x, n, order=0):    # calculate relative error
    return abs((1 / exp(x) - round5(1 / exp5(x, n, order))) * exp(x))


if __name__ == '__main__':
    # a
    print(exp5(5.5, 30))    # 244.71

    # b
    print('%d  %.5g' % (converge(5.5, 30, 0), rerr(5.5, 30, 0)))    # 17  7.3839e-05

    # c
    print('%d  %.5g' % (converge(5.5, 30, 1), rerr(5.5, 30, 1)))    # 16  7.3839e-05

    # d
    print('%d  %.5g' % (converge(-5.5, 30, 0), rerr(-5.5, 30, 0)))    # 25  0.061288
    print('%d  %.5g' % (converge(-5.5, 30, 1), rerr(-5.5, 30, 1)))    # 19  0.021232
    # print('%d  %.5g' % (converge(-5.5, 30, 2), rerr(-5.5, 30, 2)))   # math error
    # print('%d  %.5g' % (converge(-5.5, 30, 3), rerr(-5.5, 30, 3)))    # math error

    # e
```

```
    print('%d  %.5g' % (converge2(5.5, 30, 1), rerr2(5.5, 30, 1)))    # 19  2.12e-02
```

2 Code for Q5

```
from math import floor, log10

def round12(x):    # reduce to 12 significant figures
    return round(x, 11-int(floor(log10(abs(x)))))

def int12(x):  # convert double to 12 digit integer for safe comparison
    return int(round12(x * 1e12))

def equal12(x1, x2):    # compare to doubles
    return int12(x1) == int12(x2)

if __name__ == '__main__':
    prev = 0.1    # previous value of e

    for i in range(10000):
        n = 10 ** i
        e = (1.0 + 1.0 / n) ** n

        if (equal12(e, prev)):
            print('nstop:  %.0e' % (n / 10))
            print('final value: %.11f' % e)
            break

        else:
            print('i = %d    e = %.11f' % (i, e))
            prev = e
```

3 Code for Q6

```
from math import floor, log10

def round12(x):    # reduce to 12 significant figures
    return round(x, 11-int(floor(log10(abs(x)))))

def int12(x):  # convert double to 12 digit integer for safe comparison
    return int(round12(x * 1e12))

def equal12(x1, x2):    # compare to doubles
    return int12(x1) == int12(x2)

if __name__ == '__main__':
```

```python
    prev = 0.1    # previous value of e

    for i in range(10000):
        n = 10 ** i
        e = (1.0 + 1.0 / n) ** n

        if (equal12(e, prev)):
            print('nstop:  %.0e' % (n / 10))
            print('final value: %.11f' % e)
            break

        else:
            print('i = %d    e = %.11f' % (i, e))
            prev = e
```

4 Code for Q6

```python
from math import e, exp
from scipy.integrate import quad

y = 0
for n in range(31, 19, -1):
    y = (e - y) / (n + 1)

print(y)
print(quad(lambda x: x**20 * exp(x), 0, 1)[0])
```