University of Colorado
Boulder

# HW2

Jesse Hettleman

APPM 4600

College of Engineering and Applied Science

Department of Applied Math

September 12, 2024

# 1 Problem 1

This problem is asking for proofs of various "big O" and "little o" relationships numbers approaching both 0 and $\infty$.

The key assumptions in this problem are:

- $f(x) = o(g(x))$ if $\lim_{x\to 0} \frac{f(x)}{g(x)} = 0$
- $f(x) = O(g(x))$ if $\lim_{x\to 0} \frac{f(x)}{g(x)} = L$ exists

**1.a**
Prove $(1+x)^n = 1+nx+o(x)$ as $x \to 0$:

$$(1+x)^n - (1+nx) = o(x) \text{ if } \lim_{x\to 0} \frac{(1+x)^n-(1+nx)}{x} = 0$$

$$\lim_{x\to 0} \frac{(1+x)^n-(1+nx)}{x} = \lim_{x\to 0} \frac{n(1+x)^{n-1}}{1} \text{ via L'Hopital's Rule} \quad (1)$$
$$= n*1-n \quad (2)$$
$$= 0 \quad (3)$$

Thus, $(1+x)^n = 1+nx+o(x)$ as $x \to 0$.

**1.b**
Prove $xsin(\sqrt{x}) = O(x^{\frac{3}{2}})$ as $x \to 0$:

$$xsin(\sqrt{x}) = O(x^{\frac{3}{2}}) \text{ if } \lim_{x\to 0} \frac{xsin(\sqrt{x})}{x^{\frac{3}{2}}} = L \text{ exists.}$$

$$-1 \le sin(\sqrt{x}) \le 1 \quad (4)$$
$$\frac{-x}{x^{\frac{3}{2}}} \le \frac{x}{x^{\frac{3}{2}}} * sin(\sqrt{x}) \le \frac{x}{x^{\frac{3}{2}}} \quad (5)$$

$\lim_{x\to 0} \frac{-x}{x^{\frac{3}{2}}} = \lim_{x\to 0} \frac{x}{x^{\frac{3}{2}}} = 0$ so by Squeeze Theorem:

$$\lim_{x\to 0} \frac{xsin(\sqrt{x})}{x^{\frac{3}{2}}} = 0$$

Thus, $xsin(\sqrt{x}) = O(x^{\frac{3}{2}})$ as $x \to 0$.

**1.c**
Prove $e^{-t} = o(\frac{1}{t^2})$ as $x \to \infty$:

$$e^{-t} = o(\tfrac{1}{t^2}) \text{ if } \lim_{t\to\infty} \frac{e^{-t}}{\frac{1}{t^2}} = 0.$$

$$\lim_{t\to\infty} \frac{e^{-t}}{\frac{1}{t^2}} = \lim_{t\to\infty} \frac{t^2}{e^t} \tag{6}$$
$$= \lim_{t\to\infty} \frac{2t}{e^t} \text{ via L'Hopital's Rule} \tag{7}$$
$$= \lim_{t\to\infty} \frac{2}{e^t} \text{ via L'Hopital's Rule} \tag{8}$$
$$= \frac{2}{\infty} \tag{9}$$
$$= 0 \tag{10}$$

Thus, $e^{-t} = o(\frac{1}{t^2})$ as $x \to \infty$.

**1.d**
Prove $\int_0^\varepsilon e^{-x^2} dx = O(\varepsilon)$ as $\varepsilon \to 0$:

$$\int_0^\varepsilon e^{-x^2} dx = O(\varepsilon) \text{ if } \lim_{\varepsilon\to 0} \frac{\int_0^\varepsilon e^{-x^2} dx}{\varepsilon} = L \text{ exists.}$$

$$\lim_{\varepsilon\to 0} \frac{\int_0^\varepsilon e^{-x^2} dx}{\varepsilon} = \lim_{\varepsilon\to 0} \frac{\frac{d}{d\varepsilon}\int_0^\varepsilon e^{-x^2} dx}{1} \text{ via L'Hopital's Rule} \tag{11}$$
$$= \frac{0}{1} \tag{12}$$
$$= 0 \tag{13}$$

Thus, $\int_0^\varepsilon e^{-x^2} dx = O(\varepsilon)$ as $\varepsilon \to 0$.

# 2 Problem 2

This problem investigates an ill-conditioned matrix and the resulting errors in solutions given small perturbations in the inputs.

**2.a**
We are solving $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, given $\mathbf{b}$. To do solve for $\mathbf{x}$ we can use the following equation: $\mathbf{x} = A^{-1}\mathbf{b}$.

We are given the following information:

$A = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1+10^{-10} & 1-10^{-10} \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\mathbf{s} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, and $A^{-1} = \begin{bmatrix} 1-10^10 & 10^10 \\ 1+10^10 & -10^10 \end{bmatrix}$.

Given a perturbation in $\mathbf{b}$ of $\begin{bmatrix} \Delta b_1 \\ \Delta b_2 \end{bmatrix}$, the equation for $\Delta\mathbf{x}$ is:

$$\Delta \mathbf{x} = A^{-1}(\mathbf{b} + \Delta \mathbf{b}) =$$
$$\begin{bmatrix} (1-10^{10}) * (b_1 + \Delta b_1) + (10^{10}) * (b_2 + \Delta b_2) \\ (1+10^{10}) * (b_1 + \Delta b_1) - (10^{10}) * (b_2 + \Delta b_2) \end{bmatrix}$$

**2.b**

Using python code found in 5, I determined the condition number of $A$ to be 19999973849.225224.

**2.c**

When the perturbations $\Delta b_1$ and $\Delta b_1$ are of the magnitude $10^{-5}$ and not the same value, the relative error in the solution $\mathbf{x}$ is huge! Setting $\Delta b_1 = 2 * 10^{-5}$ and $\Delta b_1 = 7 * 10^{-5}$, and using python to calculate this error as the 2-norm of $A^{-1}\Delta b$ divided by the 2-norm of $\mathbf{x}$, I found the relative error to be 50000.0. This error is of the magnitude of $10^5$, which is the logarithmic inverse of the magnitude of the perturbation. Thus, when the perturbations are different, the relative error is inversely proportional to the perturbation, which is logarithmically half the magnitude of the of the condition number.

When the perturbations $\Delta b_1$ and $\Delta b_1$ are of the magnitude $10^{-5}$ and they are the same value, the relative error in the solution $\mathbf{x}$ is much smaller. Setting $\Delta b_1 = 5 * 10^{-5}$ and $\Delta b_1 = 5 * 10^{-5}$, I found the relative error to be $5.0000002374881036e-05$ using the same calculation method. This error is of the magnitude of $10^{-5}$, which is the same as the magnitude of the perturbation. Thus, when the perturbations are the same, the relative error is the same magnitude of the perturbation, which is significantly smaller than the condition number.

I hypothesize that the ill-conditioned matrix $A$ is responsible for the large errors when the perturbations are different. This is because the columns of $A$ are very close to being linearly dependent, so any perturbations cause large discrepancies. In practice, it is likely that perturbations will be different values, which means it is important to figure out appropriate algorithms for ill-conditioned matrices.

All code for calculations within this problem can be found in 5.

# 3 Problem 3

This problem deals with a few questions surrounding the condition number of the equation $f(x) = e^x - 1$.

**3.a**

The relative condition number $\kappa_{f(x)}$ is:

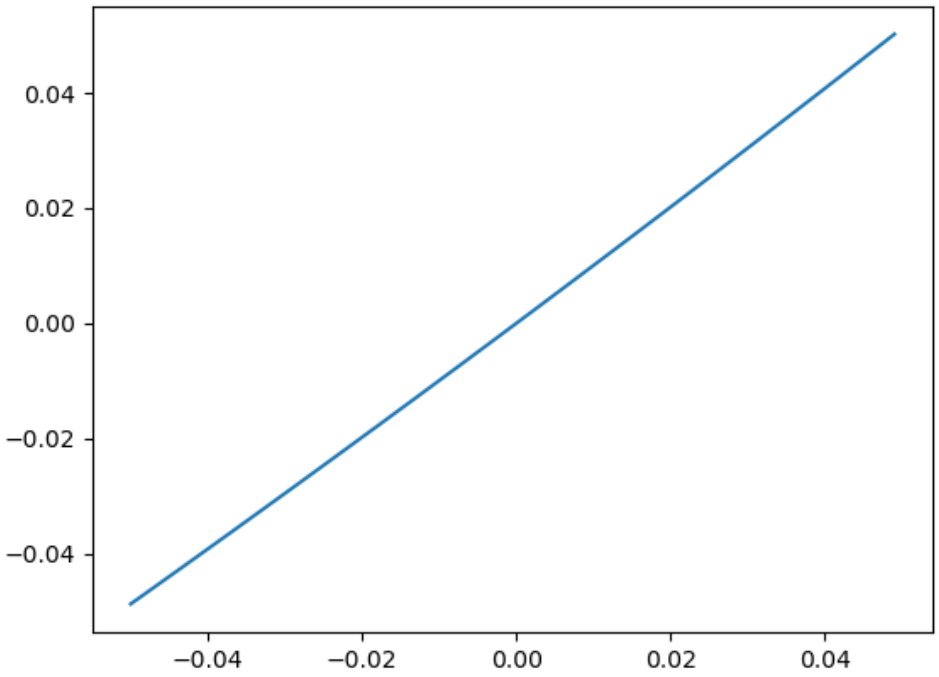
$$\kappa_{f(x)} = |f'(x)| \frac{|x|}{|f(x)|} \tag{14}$$
$$= |e^x| \frac{|x|}{|e^x - 1|} \tag{15}$$
$$= |x \frac{e^x}{e^x - 1}| \tag{16}$$

Since $\frac{e^x}{e^x-1} \to 1$ as $x \to \infty$, $\kappa_{f(x)} \to x$ as $x \to \infty$. This means for large values of $x$ this function is ill-conditioned. Likewise, when $e^x$ is close to 1, then $\kappa_{f(x)} \to \infty$, and is therefore also ill-conditioned for these values.

**3.b**

To test if $math.e^x - 1$ is stable, I plotted values this algorithm yields close to $x = 0$ in the following chart:

**Figure 1: Algorithm Output For x Values Near 0**




This line was plotted with x value increments of the magnitude $10^{-3}$. The smoothness of the line shows that this algorithm is stable; otherwise noise would be expected in the data.

**3.c**

For $x = 9.999999995000000 * 10^{-10}$, the algorithm outputs a value of $1.000000082740371 * 10^{-9}$ compared to the actual value of $10^{-9}$. This means the algorithm gives 8 correct digits. This makes sense because the condition number for this value of $x$ is 0.9999999177596358, which has 8 digits of precision.

**3.d and 3.e**

I can approximate $f(x) = e^x - 1$ using the Taylor polynomial $x + \frac{x^2}{2!} + \frac{x^3}{3!} + ....$ The first term of the Taylor polynomial for $e^x$ is 1, so this gets subtracted off from my approximation.

Using python, I determined that only 2 terms are required in the Taylor polynomial approximation in order to achieve 16 digits of relative accuracy. This is equivalent to when the relative error becomes 0.0 within the computer. See 5 for code calculations.

# 4 Problem 4

### 4.a

Summing the product of each element in $t$ and $y$ yields a total of $-20.68685236434684$. See 5 for code.

### 4.b

Plotting these arrays resulted in the following charts:

**Figure 2: Polar Plot**

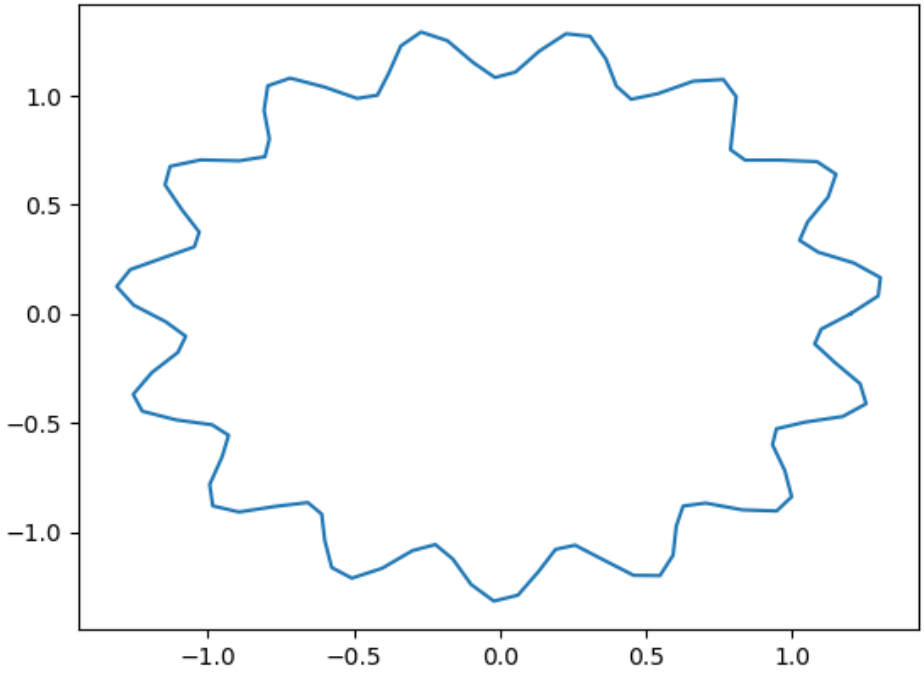
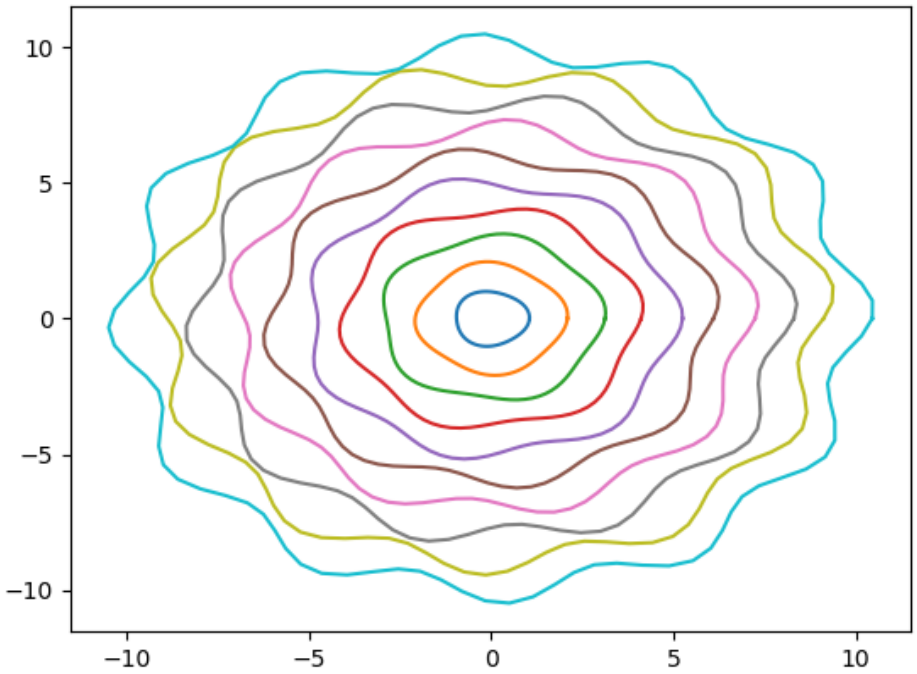


**Figure 3: Iterative Polar Plot**




See 5 for code.

# 5 Appendix

## 5.1 Code

**Description of Code Here:**

```
import numpy as np
import matplotlib.pyplot as plt
import math
import random

# -----Problem 2-----

# Initialize Matrices

A = np.array([[(1/2), (1/2)], [(1/2)*(1+(10**(-10))), ...
    (1/2)*(1-(10**(-10)))]])
b = np.array([1,1])
x = np.array([1,1])

cond_A = np.linalg.cond(A) # 1 / (1-(10**(-10)))

```

```
A_inv = np.array([[(1-(10**10)), (10**10)], [(1+(10**10)), ...
    (-(10**10))]])

# Changed b_Δ values to answer part c
b_Δ = np.array([5*(10**(-5)), 5*(10**(-5))])

x_tilde = [(A_inv[0][0]*(b_Δ[0] + b[0])) + (A_inv[0][1]*(b_Δ[1] + ...
    b[1])), (A_inv[1][0]*(b_Δ[0] + b[0])) + (A_inv[1][1]*(b_Δ[1] + ...
    b[1]))]

x_relerr = np.linalg.norm(np.matmul(A_inv,b_Δ))/np.linalg.norm(x) # ...
    [abs((x[0] - x_tilde[0])/x[0]), abs((x[1] - x_tilde[1])/x[1])]

# print("Condition number of A: ", cond_A)
# print("Relative error of x_tilde after perturbation b_Δ: ", x_relerr)


# -----Problem 3-----

def alg_1(x):
    y = math.e**(x)
    return y - 1

def cond(x):
    return x * ((math.e ** x)/(alg_1(x)))

xvals = np.arange(-0.05,0.05,0.001)
yvals = np.array([alg_1(x) for x in xvals])

plt.plot(xvals,yvals)
plt.savefig("HW2.3.b.png")
plt.clf()

part_c = (9.999999995000000 * (10**(-10)))

print("Algorith output for value in part c: ", alg_1(part_c))
print("Condition number for value in part c: ", cond(part_c))

def taylor(x,n):

    # Initialize variable for approximation
    approx = 0

    # Adds terms to taylor polynomical. First term in e^x is 1, ...
        which is cancelled by the subtraction of 1
    for i in range(1,n+1):
        approx = approx + (x**i)/(math.factorial(i))

    return approx

def relerr(x,n):
    return abs(np.expm1(x) - taylor(x,n))/abs(np.expm1(x))

for n in range(1,10):
    print("Relative error for value in part c given ", n, " terms in ...
        series: ", relerr(part_c,n))


```

```
# -----Problem 4a-----

t = np.linspace(0, math.pi, 31)
y = np.array([math.cos(T) for T in t])

# Calculate sum and print:
count = 0
S = 0
while count < len(t):
    S += t[count]*y[count]
    count += 1

print("The sum is: ", S)


# -----Problem 4b-----

R = 1.2
delr = 0.1
f = 15
p = 0

theta = np.linspace(0, 2*math.pi, 100)

x = np.array([R * (1 + delr*math.sin(f*th + p)) * math.cos(th) for ...
    th in theta])
y = np.array([R * (1 + delr*math.sin(f*th + p)) * math.sin(th) for ...
    th in theta])

plt.plot(x,y)
plt.savefig("HW2.4.b.i.png")
plt.clf()

for i in range(1,11):

    R = i
    delr = 0.05
    f = 2 + i
    p = random.uniform(0,2)

    x = np.array([R * (1 + delr*math.sin(f*th + p)) * math.cos(th) ...
        for th in theta])
    y = np.array([R * (1 + delr*math.sin(f*th + p)) * math.sin(th) ...
        for th in theta])

    plt.plot(x,y)

plt.savefig("HW2.4.b.ii.png")
plt.clf()
```

NOTE: ALL CODE FOR THIS ASSIGNMENT IS ORIGINAL, WRITTEN BY JESSE HETTLEMAN