Numerical analysis Assignment 1

Q1:



|  |  |  |
| --- | --- | --- |
| Initial number | Times 2 | Rest |
| 0.1 | 0.2 | 0 |
| 0.2 | 0.4 | 0 |
| 0.4 | 0.8 | 0 |
| 0.8 | 1.6 | 1 |
| 0.6 | 1.2 | 1 |
| 0.2 | 0.4 | 0 … |

So:

1. Notice that at section d we didn’t depend on the absolute number of hours, but on the difference between n2 and n1, which does’t changed in this section. So we get the same answer.

Q2 a. 32bit:

1. The smallest positive normalized number with 32bit in IEEE754 is when we have the lowest normalized exponent possible which is 1, and then by subtracting the bias we will get exponent of -126 and the mantissa will be all zeros.

So the number will be: =

1. The smallest positive un-normalized number with 32bit in IEEE754 is when we have the exponent field at 0, which indicates the exponent to be -126 and that the number is un-normalized and starts with 0., and the mantissa will be all zeros and 1 at the last bit.

So the number will be:=

b.

1. The smallest positive normalized number with 64bit in IEEE754 is when we have the lowest normalized exponent possible which is 1, and then by subtracting the bias we will get exponent of -1022 and the mantissa will be all zeros.

So the number will be: =

1. The smallest positive un-normalized number with 64bit in IEEE754 is when we have the exponent field at 0, which indicates the exponent to be -1022 and that the number is un-normalized and starts with 0., and the mantissa will be all zeros and 1 at the last bit.

So the number will be:=

Q3

We saw in class that for x,y we get .

We saw in practical session that for x,y we get

We will use both of those inequalities to prove that

Q4

1. Let :‎

We want

From a:

1. שתי השיטות זהות לחלוטין מבחינה נומרית. אין יתרון ל"סילוק" החיסור מכיוון שאיבוד משמעות היה מתקיים רק אם , מצב שלא יתכן בפונקציית האקספוננט. מבחינת כמות החישובים עדיפה השיטה הראשונה הדורשת פחות הפעלות של פונקציית ה .

Q5.

1. err 70 = 0.0

err 7000 = 27.001

* שגיאת הצובר גדולה יותר ככל שעובר הזמן מכיוון שאנחנו תומכים רק ב-3 ספרות עשרוניות, אז כאשר מגיעים למצב שיש באקומולטור 1 ואנחנו מנסים להוסיף לו 0.00x עבור אז לאחר החיתוך המנטיסה לא משתנה, ולכן השגיאה גדלה.

1. err72 - err70 = 5.551115123125783e-17

err8002 - err8000 = 0.008000000000002672

* ההפרשים הנ״ל כל כך שונים זה מזה מכיוון שלאחר 70 איטרציות השגיאה היא 0, ולאחר 72 איטרציות השגיאה היא בערך , כלומר השגיאה מתחילה להצטבר החל מהצעד ה72.

בתיאוריה, ההפרש בין err\_72 ל- err\_70 הוא 0, כיוון שבצעדים אלו כמעט ואין שגיאה, ובין err\_8002 ל- err\_8000 הוא בערך 0.008 כי זה ההפרש האבסולוטי בין המספרים האמיתיים שיחושבו בצעדים ה-8002 וה- 8000.

קוד התכנית מצורף בעמוד הבא.

import math  
  
  
# [i] python has a simple typing mechanism.  
# ': <type>' signifies the type of the parameter (integer in the following)  
def most\_significant(num: int, digits\_to\_keep: int):  
 *""" return the digits\_to\_keep most significant digits of num* ***:return*** *(the kept digits, the number of non-significant (zeroed- out) digits)  
 """* # [i] 'assert' is a special keyword in python.  
 # it verifies if the condition is True  
 assert digits\_to\_keep > 0, 'digits\_to\_keep should be positive'  
 num\_digits = math.floor(math.log10(num)) + 1  
 non\_significant = max(0, num\_digits - digits\_to\_keep)  
 # [i] '//' is division without reminder in python 3+  
 return (num // 10 \*\* non\_significant), non\_significant  
  
  
def adder(man\_a: int, exp\_a: int, man\_b: int, exp\_b: int):  
 if exp\_a > exp\_b:  
 return adder(man\_b, exp\_b, man\_a, exp\_a)  
 exp\_diff = exp\_b - exp\_a  
 man\_b \*= (10 \*\* exp\_diff)  
 man\_output, exp\_output = most\_significant(man\_a + man\_b, 3)  
 exp\_output += exp\_a  
 return man\_output, exp\_output  
  
  
def accumulate(n: int):  
 man\_acc = 100  
 exp\_acc = -5  
 man\_c = 400  
 exp\_c = -5  
 for i in range(n):  
 man\_acc, exp\_acc = adder(man\_a=man\_acc, exp\_a=exp\_acc, man\_b=man\_c, exp\_b=exp\_c)  
  
 real\_out = 0.001 + 0.004 \* n  
 approximate\_out = man\_acc \* (10 \*\* exp\_acc)  
 absolute\_err = math.fabs(real\_out - approximate\_out)  
 return absolute\_err  
  
  
def main():  
 err\_70 = accumulate(70)  
 err\_72 = accumulate(72)  
 err\_7000 = accumulate(7000)  
 err\_8000 = accumulate(8000)  
 err\_8002 = accumulate(8002)  
 print(f"err 70 = {err\_70}")  
 print(f"err 7000 = {err\_7000}")  
 print(f"err72 - err70 = {err\_72 - err\_70}")  
 print(f"err8002 - err8000 = {err\_8002 - err\_8000}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()