



Instituto Tecnológico
de Buenos Aires

TRABAJO PRÁCTICO N°1

IMPLEMENTACIÓN DE PUNTO FLOTANTE IEEE754 DE 16 BITS EN PYTHON

93.54 Métodos Numéricos

Grupo N°4

Legajo N°	Nombre
61428	Kevin Amiel Wahle
61430	Francisco Basili
61431	Nicolás Bustelo

24/03/2022

Índice

1. Introducción	2
2. Clase binary16	2
2.1. Constructor	2
2.1.1. dec2bin()	2
2.2. bin2dec()	2
2.3. Sobrecarga de Operadores	2
3. Funciones varias	3
3.1. IEEE2dec()	3
3.2. entero2dec()	3
3.3. frac2dec()	3
3.4. entero2bin() y frac2bin()	3
3.5. log2()	3
4. TestBench	4
5. Anexo	5
5.1. Código Clase binary16	5

1. Introducción

El objetivo de este informe es presentar y explicar el funcionamiento de un programa realizado en *Python*, el cual consiste en crear una clase que pueda convertir un número punto flotante de doble precisión (64 bits) a un número IEEE 754 de 16bits, a la vez que también se puedan realizar ciertas operaciones. El código se encuentra en el siguiente [repositorio de GitHub](#).

2. Clase `binary16`

RoundIEEE?

2.1. Constructor

En el constructor de la clase se definió "*bits*" que es una lista de 16 elementos donde se encuentran los bits correspondientes a los números convertidos al formato IEEE 754 de 16bits. También se definió "*d*" el cual contiene al número que está en el formato IEEE 754 convertido en float de 64bits. Las conversiones decimal a binario y binario a decimal se realizan gracias a las funciones "`dec2bin()`" y "`bin2dec()`" respectivamente.

2.1.1. `dec2bin()`

El método `dec2bin(number)` toma el argumento `number` e inicialmente determina si es un caso "especial" (+inf, -inf, NaN). Si lo es, genera el arreglo de bits con el formato que tiene el estándar. En caso contrario, con funciones auxiliares (ver apartado 3.1 y apartado 3.5) calcula el signo, el exponente y la mantiza (analizando si esta última corresponde a un número Normal o uno Sub-Normal). Finalmente, los guarda en `binary16.bits` concatenados.

2.2. `bin2dec()`

El método `bin2dec()` toma el número en formato binario, lo convierte a decimal y lo guarda en una variable de 64bits. Al iniciar, tiene una etapa en la que se verifica si el arreglo binario tiene el formato de algún caso de los considerados como "especial", para crearlos específicamente. Si no tiene ninguno de dichos formatos, calcula el número pasando exponente y mantisa de binario a decimal y luego aplicando la fórmula correspondiente al caso Sub-Normal o Normal según corresponda.

2.3. Sobrecarga de Operadores

Se realizó la sobrecarga del operador multiplicación por 1, multiplicación por -1, suma y resta. El código de cada una de las implementaciones, respectivamente, se puede observar en el código a continuación:

```
1  # Multiplicación por +1
2  def __pos__(self):
3      if self.d == float('-inf'):
4          return binary16(float('-inf'))
5      return binary16(self.d)
6
7  # Multiplicación por -1
8  def __neg__(self):
9      if self.d == float('inf'):
10         return binary16(float('-inf'))
11         return binary16(-self.d)
12
13  # Suma
14  def __add__(self, other):
15      d=self.d+other.d
16      return binary16(d)
17
18  # Resta
```

```

19 def __sub__(self, other):
20     d=self.d-other.d
21     return binary16(d)
22
23 # In-Place Suma
24 def __iadd__(self, other):
25     self = binary16(self.d+other.d)
26     return self
27
28 # In-Place Resta
29 def __isub__(self, other):
30     self=binary16(self.d-other.d)
31     return self

```

Se puede observar como en todos los casos se devuelve un nuevo objeto producto de instanciar la misma clase *binary16* con el resultado de la operación realizada en cuestión.

3. Funciones varias

Para preservar la legibilidad del código se definieron varias funciones que permitían convertir de decimal a binario y viceversa además de interesantes funciones como el logaritmo:

3.1. IEEE2dec()

Se utiliza para convertir un número en formato IEEE754 en un decimal, considerando si es sub-normal o normal para poder realizar la cuenta correspondiente. Utiliza tanto `frac2dec()` como `entero2dec()`.

3.2. entero2dec()

Se utiliza para convertir números enteros en binario a decimal, los exponentes en una lista de bits. En este caso la conversión se hace con potencias de 2 positivas ($2^1, 2^2, 2^3, \dots$), ya que de esta manera se ordenan en la representación IEEE 754.

3.3. frac2dec()

Se utiliza para convertir números fraccionarios como la mantisa en una lista de bits. En este caso la conversión se hace con potencias de 2 negativas ($2^{-1}, 2^{-2}, 2^{-3}, \dots$), ya que de esta manera se ordenan en la representación IEEE 754.

3.4. entero2bin() y frac2bin()

Son dos funciones que se encargan de pasar ya sea un entero a binario o una fracción respectivamente. Esto es especialmente útil ya que al momento de convertir el número en binario lo que hacemos es separar la parte entera de la fraccionaria y convertir ambos por separado. Por otro lado `entero2bin()` también es útil para convertir el exponente a binario ya que siempre utilizamos enteros.

3.5. log2()

Para calcular el logaritmo en base 2 primero se calcula mediante una ecuación de aproximación el logaritmo natural.

$$\lim_{n \rightarrow \infty} n(x^{1/n} - 1) ; n \rightarrow \infty$$

Luego gracias a las propiedades del logaritmo, se obtiene que: $\log_2(x) = \ln(x)/\ln(2)$.

4. TestBench

El objetivo del TestBench es comprobar el correcto funcionamiento de los operadores al mismo tiempo que se analiza la conversión de los números a la normativa IEEE 754. Se realizaron pruebas con todos los tipos de números posibles, dentro de los cuales se encontraban: los **normales** (negativos y positivos), los **sub-normales** (negativos y positivos), el **0** o los números muy pequeños indistinguibles con el 0, el **infinito** (tanto $-\infty$, como $+\infty$) y el **NaN**. En cada conversión se comparó el número decimal de IEEE754 con el float de 64 bits de la biblioteca *Numpy* y se registraron las pruebas pasadas con éxito

5. Anexo

5.1. Código Clase binary16

```

1 # -----
2 # @file      +PuntoFlotante.py+
3 # @brief     +Implementaci n del punto flotante IEEE 754 de 16 bits+
4 # @author    +Grupo 4+
5 # -----
6
7 # -----
8 # LIBRARIES
9 # -----
10 import math
11 import numpy as np
12 # -----
13 # CLASSES
14 # -----
15 ne = 5 # Cantidad de Bits de Exponente
16 nm = 10 # Cantidad de Bits de Mantisa
17 sesgo = 2**(ne-1)-1
18
19 class binary16:
20     def __init__(self, number):
21         self.d = 0
22         self.bits = [0]*(1+ne+nm)
23         self.dec2bin(number) # Devuelve si el numero es un caso extremo y cual
24         self.bin2dec()
25
26     def dec2bin(self, number):
27         # ne = 5 # Cantidad de Bits de Exponente
28         # nm = 10 # Cantidad de Bits de Mantisa
29         # sesgo = 2**(ne-1)-1
30         expTotal = 0 # Es a lo que se eleva el 2
31
32         # Si no es un caso extremo
33         if number != float('inf') and number != float('-inf') and not math.isnan(number):
34             self.bits[0] = 1 if number < 0 else 0 # Guardo el signo
35             modulo = abs(number) # y tomo su m dulo
36
37         # Si el n mero es NaN
38         if math.isnan(number):
39             self.bits = [0] + [1]*(ne+nm)
40             return True
41
42         # Si el n mero es infinito
43         if number == float('inf') or number == float('-inf') or modulo > (2-2**(-nm))
44         *2**(2**ne-sesgo-2):
45             self.bits[0] = 1 if number < 0 else 0 # Guardo el signo
46             self.bits = [self.bits[0]] + [1]*ne + [0]*nm # Coloco 1s en el exponente y
47             0s en la mantisa
48             return "+inf" if self.bits[0] == 0 else "-inf"
49
50         # Si el n mero es menor al n mero subnormal mas peque o , lo consideramos 0
51         if modulo < 2**(-nm)*2**(1-sesgo):
52             self.bits = [0]*(ne+nm+1) # Coloco 0s en el exponente y 0s en la
53             mantisa
54             return True
55
56         # Si el n mero es Sub-Normal
57         if modulo < 2**(1-sesgo):
58             expTotal=1-sesgo # Calculo el exponente (1 - sesgo)
59             self.bits[1:6]=[0]*ne # Coloco 0s en el exponente
60             mantisa=(modulo/2**expTotal) # Calculo la mantisa
61             type="Sub-Normal"
62
63         # Si el n mero es Normal
64         else:
65             expTotal = math.floor(log2(modulo)) # Calculo el exponente total (e -
66             sesgo)

```

```

63         exp = expTotal + sesgo                                # Calculo el exponente (e)
64         self.bits[1:6]=entero2bin(exp, ne)                    # Guardo el exponente en
65     binario                                                    # Calculo la mantisa
66         mantisa = (modulo/2**expTotal) - 1
67         type="Normal"
68         self.bits[6:16]=frac2bin(mantisa, nm)                 # Guardo la mantisa en binario
69         self.roundIEEE(number, type)                           # Redondeo el n mero
70
71
72     def roundIEEE(self, number, type):
73         auxnum = IEEE2dec(self.bits, type)
74         aux0b = self.bits[0:6]+entero2bin(entero2dec(self.bits[6:16])+1, nm)    # Calculo
75     el n mero anterior
76         aux0 = IEEE2dec(aux0b, type)
77         aux1b = self.bits[0:6]+entero2bin(entero2dec(self.bits[6:16])-1, nm)    # Calculo
78     el n mero siguiente
79         aux1 = IEEE2dec(aux1b, type)
80         err = [abs(aux0-number), abs(auxnum-number), abs(aux1-number)]          # Calculo los
81     errores
82         cercano = listMinIndex(err)                                           # Me quedo
83     con el n mero que menor error tenga
84         if cercano == 0:
85             self.bits = aux0b
86         elif cercano == 2:
87             self.bits = aux1b
88
89     def bin2dec(self):
90         # ne = 5 # Cantidad de Bits de Exponente
91         # nm = 10 # Cantidad de Bits de Mantisa
92         # sesgo = 2**(ne-1)-1
93
94         # Caso infinito
95         if self.bits[1:] == [1]*ne + [0]*nm:
96             self.d= float('inf') if self.bits[0] == 0 else float('-inf')      #
97     Dependiendo del signo el n mero sera -inf o +inf
98         return True
99
100         # Caso NaN
101         elif self.bits[1:6] == [1]*ne:
102             self.d= float('NaN')
103             return True
104
105         # Caso 0
106         if self.bits[1:] == [0]*ne + [0]*nm:
107             self.d = float(0)
108
109         # Caso Sub-Normal
110         elif self.bits[1:6] == [0]*ne:
111             mantis = 0
112             for i in range(nm):
113                 mantis += self.bits[i+6]* 2**(-i-1)                        # Calculo la mantisa en
114     decimal
115             self.d = (-1)**self.bits[0]*mantis*2**(1-sesgo)                # Armo mi n mero en
116     decimal
117
118         # Caso Normal
119         else:
120             mantis = 1
121             for i in range(nm):
122                 mantis += self.bits[i+6] * 2**(-i-1)                        # Calculo la mantisa en
123     decimal
124             expo = 0
125             for j in range(ne):
126                 expo += self.bits[j+1] * 2**(ne-j-1)                        # Calculo el exponente en
127     decimal
128             self.d = (-1)**self.bits[0]*mantis*2**(expo-sesgo)              # Armo mi n mero decimal
129
130     # Multiplicaci n por +1
131     def __pos__(self):
132         if self.d == float('-inf'):
133             return binary16(float('-inf'))

```

```

125         return binary16(self.d)
126
127     # Multiplicación por -1
128     def __neg__(self):
129         if self.d == float('inf'):
130             return binary16(float('-inf'))
131         return binary16(-self.d)
132
133     # Suma
134     def __add__(self, other):
135         d=self.d+other.d
136         return binary16(d)
137
138     # Resta
139     def __sub__(self, other):
140         d=self.d-other.d
141         return binary16(d)
142
143     # In-Place Suma
144     def __iadd__(self, other):
145         self = binary16(self.d+other.d)
146         return self
147
148     # In-Place Resta
149     def __isub__(self, other):
150         self=binary16(self.d-other.d)
151         return self
152
153 # -----
154 # FUNCTION DEF
155 # -----
156 def entero2bin (exp, expBits):      # Convierte un numero entero base 10 en binario con
157     # potencias positivas
158     cont = 0
159     expb=[]
160
161     if exp > 2**expBits:             # Numero mayor de lo que puedo guardar
162         return [1]*expBits          # Coloco 1s en el exponente
163
164     while cont<expBits:
165         expb = [exp%2] + expb        # Divido por 2 y me quedo con el resto
166         exp = exp // 2
167         cont += 1
168     return expb
169
170 def entero2dec(bits):                # Convierte un numero entero binario en decimal
171     sum=0
172     for i in range(len(bits)):
173         sum += bits[len(bits)-1-i] * 2**i
174     return sum
175
176 def frac2bin (man, manBits):         # Convierte un numero decimal en binario con potencias
177     # negativas
178     cont = 0
179     manb=[]
180
181     while cont<manBits:
182         man *= 2                     # Multiplico al numero por 2, y dependiendo de si es
183         # mayor a 1 o no, armo el binario
184         if man>=1:
185             manb = manb + [1]
186             man -= 1
187         else:
188             manb = manb + [0]
189         cont += 1
190     return manb
191
192 def frac2dec(bits):                  # Convierte un numero racional codificado en binario a
193     # decimal
194     mant = 0
195     for i in range(-len(bits),0):

```



```

193     mant += bits[-i-1] * 2**i
194     return mant
195
196 def IEEE2dec(bits, type=""):
197     mantis = frac2dec(bits[6:])
198     if type == "Sub-Normal":
199         return (-1)**bits[0]*mantis*2**(1-sesgo)           # Armo mi n mero en decimal
200
201     elif type == "Normal":
202         expo = entero2dec(bits[1:6])
203         return (-1)**bits[0]*(1+mantis)*2**(expo-sesgo)      # Armo mi n mero decimal
204
205     return 0
206
207 def listMinIndex(list):
208     min=0
209     for i in range(len(list)): # Recorro toda la lista
210         if list[i]<list[min]:   # Si el elemento actual es menor al que tengo que
211             min=i              # Actualizo el ndice del valor del minimo
212     return min
213
214 def ln(x):
215     n = 1e15
216     return n * ((x ** (1/n)) - 1)
217
218 def log2(x):
219     return ln(x)/ln(2)
220
221 # -----
222 # -----
223 #                               TestBench DEF
224 # -----
225 # -----
226 def operationTest(num):
227     IeeeNumb = binary16(num)      # Tomo un n mero
228     IeeeNumb2 = binary16(num*2)   # Tomo otro n mero
229     res = binary16(0)             # Aqu se almacenar el resultado
230
231     print('Numero inicial:', num)
232     print('Numero guardado:', IeeeNumb.d)
233
234     print("SUMA")
235     res = IeeeNumb + IeeeNumb2
236     print('a+b: ', IeeeNumb.d, '+', IeeeNumb2.d, '=', res.d, '-> IEEE754: ', res.bits)
237
238     print("RESTA")
239     res = IeeeNumb - IeeeNumb2
240     print('a-b: ', IeeeNumb.d, '-', IeeeNumb2.d, '=', res.d, '-> IEEE754: ', res.bits)
241
242     print("SUMA2")
243     res = IeeeNumb
244     res += IeeeNumb2
245     print('a+=b: ', IeeeNumb.d, '+=' , IeeeNumb2.d, '=', res.d, '-> IEEE754: ', res.bits)
246
247     print("RESTA2")
248     res = IeeeNumb
249     res -= IeeeNumb2
250     print('a-=b: ', IeeeNumb.d, '-=' , IeeeNumb2.d, '=', res.d, '-> IEEE754: ', res.bits)
251
252     print("POS")
253     res = +IeeeNumb
254     print('+a: ', IeeeNumb.d, '=', res.d, '-> IEEE754: ', res.bits)
255
256     print("NEG")
257     res = -IeeeNumb
258     print('-a: ', '-', IeeeNumb.d , '=', res.d, '-> IEEE754: ', res.bits)
259
260     return 1 if (IeeeNumb.d==np.float16(num) and IeeeNumb2.d==np.float16(num*2)) else 0
261
262 def test():
263     passCases=0 #Casos que se pasaron

```

```
264 totalCases=0
265
266 # Numero positivo
267 passCases+=operationTest(4.2)
268 totalCases+=1
269 print("\n")
270
271 # Numero negativo
272 passCases+=operationTest(-3.14)
273 totalCases+=1
274 print("\n")
275
276 # Numero subnormal
277 passCases+=operationTest(3e-7)
278 totalCases+=1
279 print("\n")
280
281 # Numero subnormal negativo
282 passCases+=operationTest(-3e-7)
283 totalCases+=1
284 print("\n")
285
286 # Numero muy cercano a cero -> 0
287 passCases+=operationTest(-5e-10)
288 totalCases+=1
289 print("\n")
290
291 # Numero mayor al mas grande -> inf
292 passCases+=operationTest(999999999)
293 totalCases+=1
294 print("\n")
295
296 # Numero menor al mas chico -> -inf
297 passCases+=operationTest(-999999999)
298 totalCases+=1
299 print("\n")
300
301 # Infinito de float
302 passCases+=operationTest(float('+inf'))
303 totalCases+=1
304 print("\n")
305
306 # Infinito negativo de float
307 passCases+=operationTest(float('-inf'))
308 totalCases+=1
309 print("\n")
310
311 # Not a Number de float
312 operationTest(float('NaN'))
313 print("\n")
314
315 print ('Las pruebas pasadas con xito fueron', passCases, '/', totalCases)
316
317 # -----
318 # MAIN
319 # -----
320 test()
```