Trabajo Práctico N^{o}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
3.6. roundIEEE() . . . 4

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

2.1. Constructor

En el constructor de la clase se definió *"bits"* que es una lista de 16 elementos donde se encontraran 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:

```
    # Multiplicación por +1
    def __pos__(self):
        if self.d == float('-inf'):
            return binary16(float('-inf'))
        return binary16(self.d)

    # Multiplicación por -1
    def __neg__(self):
        if self.d == float('inf'):
            return binary16(float('-inf'))
        return binary16(-self.d)

    # Suma
    def __add__(self,other):
        d=self.d+other.d
        return binary16(d)

    # Resta
    def __sub__(self,other):
```

```
        d=self.d-other.d
        return binary16(d)

    # In-Place Suma
    def __iadd__(self,other):
        self = binary16(self.d+other.d)
        return self

    # In-Place Resta
    def __isub__(self,other):
        self=binary16(self.d-other.d)
        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 ademas de interesantes funciones como el logaritmo:

3.1. IEEE2dec()

Se utiliza para convertir un número en formato IEEE 754 a formato 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 codificados en binario a formato decimal. Recibe un arreglo con los coeficientes correspondientes a cada una de las potencias de 2 positivas (2^1, 2^2, 2^3 ,...), ya que de esta manera se ordenan en la representación IEEE 754.

3.3. frac2dec()

Se utiliza para convertir números fraccionarios, como la mantisa, que se encuentran en base binaria a base decimal. En este caso la conversión se hace con potencias de 2 negativas (2^{-1}, 2^{-2}, 2^{-3} ,...) ya que de esa forma se codifican los números menores a 1 en punto flotante.

3.4. entero2bin() y frac2bin()

Son dos funciones que se encargan de convertir ya sea un entero o una fracción a binario. Esto es especialmente útil ya que al momento de convertir el número en binario lo que se hace es calcular el exponente y la mantiza, para luego 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 calculó, mediante una ecuación de aproximación, el logaritmo natural.

$$ln(x) = \lim_{n\to\infty} n(x^{1/n} - 1)$$

Utilizando *n=1e15* logramos tener una buena aproximación y luego gracias a las propiedades del logaritmo, logramos calcular: $log_2(x) = ln(x)/ln(2)$.

$$log_2(x) = ln(x)/ln(2)$$

3.6. roundIEEE()

Llamando a varias de las funciones anteriormente explicadas, esta función calcula el número anterior y el siguiente del que se le pase como argumento, manteniendo fijo el exponente. Luego, calcula el error de los tres comparándolos con el número original y retorna el que menor error tenga en el formato propuesto por el estándar IEEE 754.

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

```
# ------------------------------------------------------------------------------
#  @file     +PuntoFlotante.py+
#  @brief    +Implementación del punto flotante IEEE 754 de 16 bits+
#  @author   +Grupo 4+
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# LIBRARIES
# ------------------------------------------------------------------------------
import math
import numpy as np
# ------------------------------------------------------------------------------
# CLASSES
# ------------------------------------------------------------------------------
ne = 5  # Cantidad de Bits de Exponente
nm = 10 # Cantidad de Bits de Mantisa
sesgo = 2**(ne-1)-1

class binary16:
    def __init__(self, number):
        self.d = 0
        self.bits = [0]*(1+ne+nm)
        self.dec2bin(number)  # Devuelve si el numero es un caso extremo y cual
        self.bin2dec()

    def dec2bin(self, number):
        # ne = 5  # Cantidad de Bits de Exponente
        # nm = 10 # Cantidad de Bits de Mantisa
        # sesgo = 2**(ne-1)-1
        expTotal = 0 # Es a lo que se eleva el 2

        # Si no es un caso extremo
        if number != float('inf') and number != float('-inf') and not math.isnan(number):
            self.bits[0] = 1 if number < 0 else 0   # Guardo el signo
            modulo = abs(number)                    # y tomo su módulo

        # Si el número es NaN
        if math.isnan(number):
            self.bits = [0] + [1]*(ne+nm)
            return True

        # Si el número es infinito
        if number == float('inf') or number == float('-inf') or modulo > (2-2**(-nm))
    *2**(2**ne-sesgo-2):
            self.bits[0] = 1 if number < 0 else 0           # Guardo el signo
            self.bits = [self.bits[0]] + [1]*ne + [0]*nm    # Coloco 1s en el exponente y
     0s en la mantisa
            return "+inf" if self.bits[0] == 0 else "-inf"

        # Si el número es menor al  número subnormal mas pequeño, lo consideramos 0
        if modulo < 2**(-nm)*2**(1-sesgo):
            self.bits = [0]*(ne+nm+1)       # Coloco 0s en el exponente y 0s en la
    mantisa
            return True

        # Si el número es Sub-Normal
        if modulo < 2**(1-sesgo):
            expTotal=1-sesgo                # Calculo el exponente (1 - sesgo)
            self.bits[1:6]=[0]*ne           # Coloco 0s en el exponente
            mantisa=(modulo/2**expTotal)    # Calculo la mantisa
            type="Sub-Normal"

        # Si el número es Normal
        else:
            expTotal = math.floor(log2(modulo))    # Calculo el exponente total (e -
    sesgo)
```

```
            exp = expTotal + sesgo                          # Calculo el exponente (e)
            self.bits[1:6]=entero2bin(exp, ne)              # Guardo el exponente en
    binario
            mantisa = (modulo/2**expTotal) - 1              # Calculo la mantisa
            type="Normal"

        self.bits[6:16]=frac2bin(mantisa, nm)               # Guardo la mantisa en binario
        self.roundIEEE(number, type)                        # Redondeo el número


    def roundIEEE(self, number, type):
        auxnum = IEEE2dec(self.bits, type)
        aux0b = self.bits[0:6]+entero2bin(entero2dec(self.bits[6:16])+1, nm)    # Calculo
     el número anterior
        aux0 = IEEE2dec(aux0b, type)
        aux1b = self.bits[0:6]+entero2bin(entero2dec(self.bits[6:16])-1, nm)    # Calculo
     el número siguente
        aux1 = IEEE2dec(aux1b, type)
        err = [abs(aux0-number), abs(auxnum-number), abs(aux1-number)]      # Calculo los
     errores
        cercano = listMinIndex(err)                                         # Me quedo
    con el número que menor error tenga
        if cercano == 0:
            self.bits = aux0b
        elif cercano == 2:
            self.bits = aux1b

    def bin2dec(self):
        # ne = 5  # Cantidad de Bits de Exponente
        # nm = 10 # Cantidad de Bits de Mantisa
        # sesgo = 2**(ne-1)-1

        # Caso infinito
        if self.bits[1:] == [1]*ne + [0]*nm:
            self.d= float('inf') if self.bits[0] == 0 else float('-inf')    #
    Depenendiendo del signo el número sera -inf o +inf
            return True

        # Caso NaN
        elif self.bits[1:6] == [1]*ne:
            self.d= float('NaN')
            return True

        # Caso 0
        if self.bits[1:] == [0]*ne + [0]*nm:
            self.d = float(0)

        # Caso Sub-Normal
        elif self.bits[1:6] == [0]*ne:
            mantis = 0
            for i in range(nm):
                mantis += self.bits[i+6]* 2**(-i-1)             # Calculo la mantisa en
    decimal
            self.d = (-1)**self.bits[0]*mantis*2**(1-sesgo)     # Armo mi número en
    decimal

        # Caso Normal
        else:
            mantis = 1
            for i in range(nm):
                mantis += self.bits[i+6] * 2**(-i-1)            # Calculo la mantisa en
    decimal
            expo = 0
            for j in range(ne):
                expo += self.bits[j+1] * 2**(ne-j-1)            # Calculo el exponente en
     decimal
            self.d = (-1)**self.bits[0]*mantis*2**(expo-sesgo)  # Armo mi número decimal

    # Multiplicación por +1
    def __pos__(self):
        if self.d == float('-inf'):
            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
    potencias positivas
157     cont = 0
158     expb=[]
159
160     if exp > 2**expBits:         # Numero mayor de lo que puedo guardar
161         return [1]*expBits       # Coloco 1s en el exponente
162
163     while cont<expBits:
164         expb = [exp%2] + expb    # Divido por 2 y me quedo con el resto
165         exp = exp // 2
166         cont += 1
167     return expb
168
169 def entero2dec(bits):            # Convierte un numero entero binario en decimal
170     sum=0
171     for i in range(len(bits)):
172         sum += bits[len(bits)-1-i] * 2**i
173     return sum
174
175 def frac2bin (man, manBits):     # Convierte un numero decimal en binario con potencias
    negativas
176     cont = 0
177     manb=[]
178
179     while cont<manBits:
180         man *= 2                 # Multiplico al numero por 2, y dependiendo de si es
    mayor a 1 o no, armo el binario
181         if man>=1:
182             manb = manb + [1]
183             man -= 1
184         else:
185             manb = manb + [0]
186         cont += 1
187
188     return manb
189
190 def frac2dec(bits):              # Convierte un numero racional codificado en binario a
    decimal
191     mant = 0
192     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 número 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 número 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
    comparar
211             min=i               # Actualizo el índice 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(numb):
227     IeeeNumb = binary16(numb)       # Tomo un número
228     IeeeNumb2 = binary16(numb*2)    # Tomo otro número
229     res = binary16(0)               # Aquí se almacenará el resultado
230 
231     print('Numero inicial:', numb)
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(numb) and IeeeNumb2.d==np.float16(numb*2)) else 0
261 
262 def test():
263     passCases=0 #Casos que se pasaron
```

```
    totalCases=0

    # Numero positivo
    passCases+=operationTest(4.2)
    totalCases+=1
    print("\n")

    # Numero negativo
    passCases+=operationTest(-3.14)
    totalCases+=1
    print("\n")

    # Numero subnormal
    passCases+=operationTest(3e-7)
    totalCases+=1
    print("\n")

    # Numero subnormal negativo
    passCases+=operationTest(-3e-7)
    totalCases+=1
    print("\n")

    # Numero muy cercano a cero -> 0
    passCases+=operationTest(-5e-10)
    totalCases+=1
    print("\n")

    # Numero mayor al mas grande -> inf
    passCases+=operationTest(999999999)
    totalCases+=1
    print("\n")

    # Numero menor al mas chico -> -inf
    passCases+=operationTest(-999999999)
    totalCases+=1
    print("\n")

    # Infinito de float
    passCases+=operationTest(float('+inf'))
    totalCases+=1
    print("\n")

    # Infinito negativo de float
    passCases+=operationTest(float('-inf'))
    totalCases+=1
    print("\n")

    # Not a Number de float
    operationTest(float('NaN'))
    print("\n")

    print ('Las pruebas pasadas con éxito fueron', passCases, '/', totalCases)

# ------------------------------------------------------------------------------
# MAIN
# ------------------------------------------------------------------------------
test()
```