Métodos Numéricos

Trabajo práctico N°1

Punto Flotante

Grupo 11: Martin Amagliani
Olivia De Vincenti
Nicolás Fernandez Pelayo

Profesor: Pablo Ignacio Fierens

Fecha de entrega: 4 de septiembre de 2020

Buenos Aires, Argentina

Consigna

Grupo impar

1. Escriba una función que convierta un número binario punto flotante IEEE 754 de 16 bits a un número en notación decimal. El nombre de la función debe ser binf2dec, recibir como parámetro solamente un arreglo de Numpy, 16×1, de 1s y 0s y devolver el número correspondiente. La función debe estar dentro de un archivo de nombre bin2dec.py.

2. Escriba una función que pruebe el correcto funcionamiento de la implementada en el ítem anterior. La función debe llamarse test, sin argumentos, y se la debe colocar en el mismo archivo que la anterior.

Funciones

Índice de Códigos

1. Ejercicio 1

1.1. binf2dec

Recibe un arreglo numpy de 16 cifras binarias y lo transforma a punto flotante IEEE 754. Su primera acción es verificar si el arreglo es válido, mientras llama a conv_bin_exp y conv_bin_mant. En caso de encontrar error escribe en pantalla y no devuelve nada. En caso de ser válido se analiza para casos especiales. Si es NaN, 0 o $\pm\infty$ se devuelve ese valor. En caso de ser un número normal o subnormal se procede a su respectivo cálculo.

Código 1: Definición de binf2dec

```
def binf2dec(arr):
    # Verificación
    if len(arr) != 16:
        print("Por favor ingrese un número binario de 16 dígitos")
        return
    signo = arr[0]                              # Calculo signo
    if signo != 0 and signo != 1:                    # Reviso si hubo error
        print("Por favor ingrese un número en binario")
        return
    exp = conv_bin_exp(np.array(arr[1:6]))               # Calculo exponente
    if exp == -1:                                  # Reviso si hubo error
        print("Por favor ingrese un número en binario")
        return
    mantissa = conv_bin_mant(np.array(arr[5 + 1:16]))      # Calculo mantissa
    if mantissa == -1:                              # Reviso si hubo error
        print("Por favor ingrese un número en binario")
        return

    # Analizo casos
    if exp == 0:
        if mantissa == 0:                           # Caso 0
            n = 0
        else:                                  # Caso sub-normal
            n = sub_normal(signo, mantissa)
    elif exp == 31:
        if mantissa == 0:                           # Caso infinito
            n = ((-1)**signo)*np.inf
        else:                                  # Caso not a number
            n = np.nan
    else:
        n = normal(signo, exp, mantissa)                # Caso normal

    return n
```

1.2. conv_bin_exp

Recibe un arreglo que tiene solo la parte exponencial de un número binario. Su función es convertir la parte exponencial binaria en número decimal. Para esto, se hace un for que pase por todas las posiciones del arreglo, por cada 1 del arreglo se suma 2 al exponente de su posición antes de la coma.

Código 2: Definición de conv_bin_exp

```
def conv_bin_exp(arr):
    num = 0
    for i in range(len(arr)):
        if arr[i] == 1:
            num = num + 2 ** ((len(arr) - 1) - i)
        elif arr[i] != 0:
            num = -1
            break
    return num
```

1.3. conv_bin_mant

Calcula el valor de la mantissa, hace un for y cada vez que encuentra un 1 multiplica por su respectiva potencia de 2, sumando cada resultado hasta llegar a la última cifra y así devolver el valor de la mantissa. En caso de encontrar un número distinto de 0 o 1 se devuelve -1 para significar error.

Código 3: Definición de conv_bin_mant

```
def conv_bin_mant(arr):
    num = 0
    for i in range(len(arr)):
        if arr[i] == 1:
            num = num + 2 ** (-1 - i)
        elif arr[i] != 0:
            num = -1
            break
    return num
```

1.4. normal

Recibe el signo, exponente y mantissa para calcular el valor en punto flotante del número normal utilizando la fórmula vista en clase. Devuelve este valor.

Código 4: Definición de normal

```
def normal(s, exp, mantissa):
    exp = exp - SESGO
    n = ((-1) ** s) * (1+mantissa) * (2 ** exp)
    return n
```

1.5. sub_normal

Recibe el signo y mantissa para calcular el valor en punto flotante del número subnormal utilizando la fórmula vista en clase. Devuelve este valor.

Código 5: Definición de sub_normal

```
def sub_normal(s, mantissa):
    exp = 1 - SESGO
    n = ((-1) ** s) * mantissa * (2 ** exp)
    return n
```

2. Ejercicio 2

2.1. test

Función de prueba. Prueba distintos valores posibles a partir de un arreglo que contiene el número en formato punto flotante de 16 bits y su forma decimal. Compara ambas columnas e indica si hubo error. Se consideró incluir en el testeo:

- Cero
- $+\infty$
- $-\infty$
- Arreglo no binario
- Arreglo de menos de 16 bits
- Arreglo con texto
- Número normal entero
- Número normal racional y mayor a 1
- Número normal racional y menor a 1
- Número sub-normal
- Not a number

En los últimos dos casos tuvimos que tomar criterios de verificación distintos. Con los números sub-normales consideramos que, al ser tan pequeños, habría que tener en cuenta el error de la máquina por lo que verificamos que pertenezca a un intervalo en vez de chequear la igualdad. Para el caso not a number notamos que el testeo general no iba a funcionar nunca entonces usamos la función de numpy np.isnan.

Código 6: Definición de test

```
def test():

  test_arr = [["Cero: |0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|",
               np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),      0],
              ["Más infinito: |0|1|1|1|1|1|0|0|0|0|0|0|0|0|0|0|",
               np.array([0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),  np.inf],
              ["Menos infinito: |1|1|1|1|1|1|0|0|0|0|0|0|0|0|0|0|",
               np.array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]), -np.inf],
              ["Non Binary: |0|0|0|0|0|0|0|0|0|0|0|0|2|0|0|",
               np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]),  None],
              ["Bad size: |0|0|",
               np.array([0, 0]),                                          None],
              ["Not even a number: |h|o|l|a|c|o|m|o|e|s|t|a|s|v|o|s|",
               np.array(["holacomoestasvos"]),                             None],
              ["Normal (40): |0|1|0|1|0|0|0|1|0|0|0|0|0|0|0|0|",
               np.array([0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]),    40],
              ["Normal (-12.84): |1|1|0|0|1|0|1|0|0|1|0|0|0|0|0|0|",
               np.array([1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0]),   -12.5],
              ["Normal (0.09375): |0|0|1|0|1|1|1|0|0|0|0|0|0|0|0|0|",
               np.array([0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]), 0.09375],
              ["Sub-Normal (4e-5): |0|0|0|0|0|0|1|0|0|1|0|0|1|0|0|0|",
               np.array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1]),   4e-05],
              ["NaN: |1|1|1|1|1|1|1|1|1|1|1|1|1|1|1|1|",
               np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),  np.nan]]


   # Casos generales
   for i in range(len(test_arr) - 2):
      print("\n" + test_arr[i][0])
      n = binf2dec(test_arr[i][1])
      if(n != test_arr[i][2]):
         print("ERROR")

   # Caso Sub-Normal
   print("\n" + test_arr[len(test_arr) - 2][0])
   n = binf2dec(test_arr[len(test_arr) - 2][1])
   if not (n >= (test_arr[len(test_arr) - 2][2] - 2**-10))
   and ((n <= test_arr[len(test_arr) - 2][2]) + 2**-10):
      print("ERROR")

   # Caso Not a Number
   print("\n" + test_arr[len(test_arr) - 1][0])
   n = binf2dec(test_arr[len(test_arr) - 1][1])
   if not np.isnan(n):
      print("ERROR")

   print("\n" + "Fin del programa de prueba")

   return
```