A1

In [47]:
```
# Standard imports
import numpy as np
np.seterr(all='ignore');  # allows floating-point exceptions
import matplotlib.pyplot as plt
```

Q2: fp2dec

In [3]:
```
# Supplied Code

def fpMath(b1, b2, fcn, t, L, U):
    '''
     b = fpMath(b1, b2, fcn, t, L, U)

     Performs a binary arithmetic operation.

     Inputs:
       b1 and b2 are the input binary strings, from F(2, t, L, U)
       fcn is a function that takes 2 inputs, and returns a value.
       t, L, and U specify the FPNS.

     Output:
       b is a binary string for the nearest value in F(2, t, L, U).

     For example, 3 - (-1) in binary, is

      fpMath('+0.11b2', '-0.01b2', (lambda z1,z2: z1-z2), 2, -5, 5)

     would perform subtraction, and return the string '+0.10b3'.
    '''

    x1 = fp2dec(b1)
    x2 = fp2dec(b2)

    y = fcn(x1, x2)

    b = dec2fp(y, t, L, U)

    return b

def dec2fp(x_orig, t, L, U):
    '''
     b = dec2fp(x, t, L, U)

     Converts the number x to a binary floating-point representation,
     rounding to t digits, and L and U as the lower and upper bounds on
     the exponent.
    '''

    x = x_orig
```

```python
# First, handle the exceptions
if x==0:
    y = '+0.'
    for k in range(t):
        y += '0'
    y += 'b0'
    return y
elif x==np.inf:
    return 'Inf'
elif x==-np.inf:
    return '-Inf'
elif np.isnan(x):
    return 'NaN'

if x<0:
    x = -x
    bx = '-0.'
else:
    bx = '+0.'

if x>=1:
    myexp = 0
    while x>=1:
        x /= 2.
        myexp += 1
else:
    myexp = 1
    while x<1:
        x *= 2.
        myexp  -= 1
    x /= 2.

remainder = x - np.floor(x)

# Process the fractional part for t binary digits
for d in range(t):
    remainder = remainder * 2.
    if remainder>=1:
        bx += '1'
        remainder -= 1
    else:
        bx += '0'

bx += 'b' + str(myexp)

# Round up if remainder is >= 0.5
if False: #remainder>=0.5:
    delta = list(bx)
    delta[3:3+t] = '0'*t
    delta[2+t] = '1'
    bx_up = fpMath(bx, ''.join(delta), (lambda z1,z2: z1+z2), t, L, U);
    print('Rounding '+bx+' up to '+bx_up+' by adding '+''.join(delta))
    bx = bx_up

y = bx
```

```
    if myexp>U:
        y = 'overflow'
    elif myexp<L:
        y = '+0.'
        for k in range(t):
            y += '0'
        y += 'b0'

    return y
```

In [85]:
```
def fp2dec(B):
    '''
     x = fp2dec(B)

     Converts the string B to a decimal value (double-precision).
     Examples:
      fp2dec('+0.11000b1')  ->  1.5
      fp2dec('-0.10101b-2') -> -0.1640625
      fp2dec('-0.101b5')    -> -20
    '''

    # ==== YOUR CODE HERE ====
    pos_of_b = 0;

    for i in range(0, len(B)):
        if (B[i] == 'b'):
            # Ignore the sign and 0.
            pos_of_b = i;

    exp = np.double(0);
    if (B[pos_of_b +1] == '-'):
        exp = int(B[pos_of_b+2:len(B)]);
        exp = exp * -1;

    else:
        exp = int(B[pos_of_b+1:len(B)]);

    sum = 0;
    exp = exp -1;
    for decimal in range (3, pos_of_b):
        sum += np.double(B[decimal]) * np.double(np.float_power(2,exp));
        exp = exp - 1;

    if B[0] == '-':
        sum = sum * -1;
    return sum
```

In [86]:
```
fp2dec('+0.11000b1')
```

Out[86]: 1.5

In [87]:
```
fp2dec('-0.10101b-2')
```

Out[87]: -0.1640625

In [88]:
```
fp2dec('-0.101b5')
```
Out[88]:
```
-20.0
```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```