Task 1

I have created a absErr.py file and relErr.py file each with their own testing file. I have documented both .py files in my Software Manual. In addition, I have added the error files to my shared library.
The routine for the absolute error is provided below:

```python
import numpy as np

def absErr(exVal, apprVal):
    absErr = np.abs(exVal - apprVal)
    print('The Approximate Value: ', apprVal)
    print('The Exact Value: ', exVal)
    print('The Absolute Error: ', absErr)
```

With the following code for testing:

```python
import numpy as np
from absErr import absErr

exVal = input('Enter the Exact Value:\n')

apprVal = input('Enter the Approximate Value:\n')

absErr(np.float32(exVal), np.float32(apprVal))
```

And the following output:

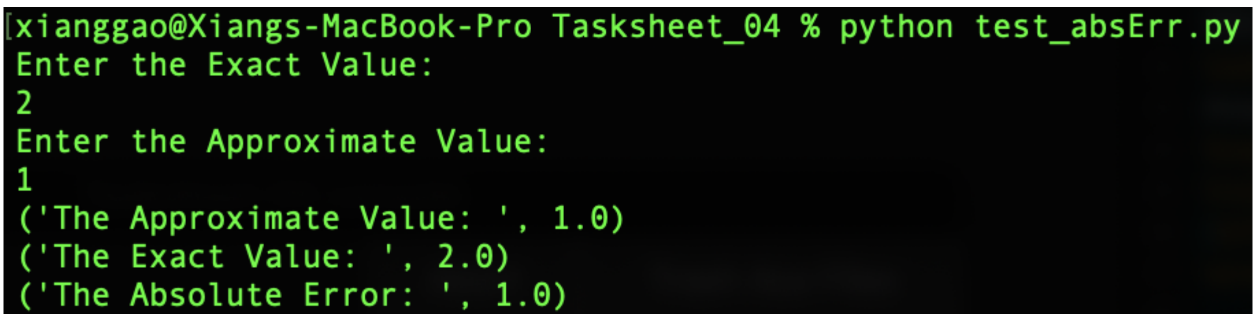

Figure 1. Machine Epsilon Single Precision Output:

The routine for relative error is provided below:

```python
import numpy as np

def relErr(exVal, apprVal):
    relErr = (np.abs(exVal - apprVal))/exVal
    print('The Approximate Value: ', apprVal)
    print('The Exact Value: ', exVal)
    print('The Absolute Error: ', relErr)
```

With the following code for testing:

```python
import numpy as np
from relErr import relErr

exVal = input('Enter the Exact Value:\n')

apprVal = input('Enter the Approximate Value:\n')

relErr(np.float32(exVal), np.float32(apprVal))
```

And the following output:

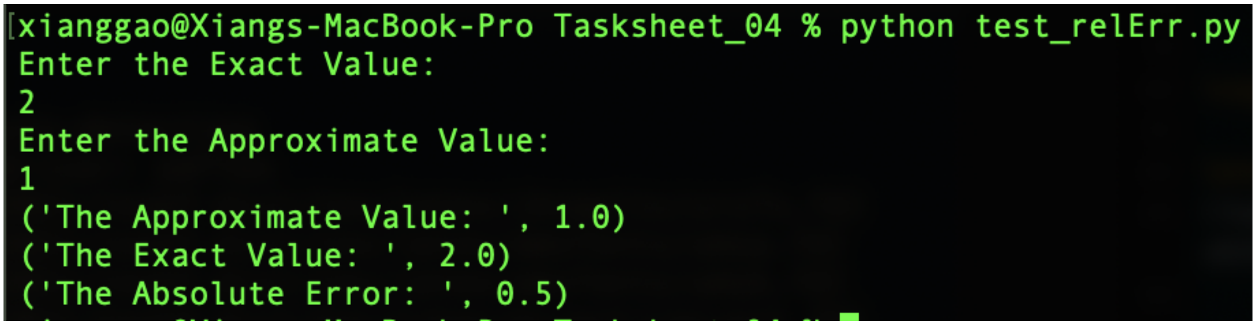

Figure 2. Machine Epsilon Double Precision Output.

Task 2

I have created a graphics routine as shown in the following:

```
import matplotlib.pyplot as plt
import numpy as np

# Initialize two arrays to store the input data in
# -------------------------------------------------
xpts=[]
ypts=[]

# Identify number of sub-plots from the user
# ------------------------------------------
plot_num = input('Enter number of expressions you want to plot:\n')

# Request from user the endpoints of the interval that defines the graphical domain
# --------------------------------------------------------------------------------
xmin = input('Enter left endpoint of an interval:\n')
xmin = float(xmin)
xmax = input('Enter right endpoint of an interval:\n')
xmax = float(xmax)

# Request from user the number of points for graphing the expression given
# -----------------------------------------------------------------------
nvals = input('Enter the number of points for graphing the expression:\n')
nvals = float(nvals)
delx = (xmax - xmin) / float(nvals)
i = 0
while i<=nvals:
        x = xmin + float(i) * delx
        xpts.append(x)
        i += 1

# Graphics for the input given
# ----------------------------
plt.xlim(xmin, xmax)

# Loop over the number of expressions specified
# ---------------------------------------------
expression = input('Enter the next expression (include np.) for f(x):\n')

# Loop over the points, evaluating the expression
# -----------------------------------------------
i = 0
while i <= nvals:
        x = xpts[i]
        ypts.append(eval(expression))
        i += 1

# Plot the data using matplotlib.pyplot
```

```python
# -------------------------------------
plt.plot(xpts, ypts, label=expression)

# Hardcoding axes labels for the 2D plot
# ---------------------------------------
plt.xlabel('x-axis')
plt.ylabel('y-axis')

# Create a legend for the plot
# ----------------------------
plt.legend()

# Show the plot of the data
# -------------------------
plt.show()
```

Where I wanted to draw the **topologist's sine wave**, so I give it the following input

```
Enter number of expressions you want to plot:
1
Enter left endpoint of an interval:
0.001
Enter right endpoint of an interval:
1
Enter the number of points for graphing the expression:
200
Enter the next expression (include np.) for f(x):
np.sin(np.power(x,-1))
```

Figure 3. Input for the Graphics Routine.

And it produced the following output

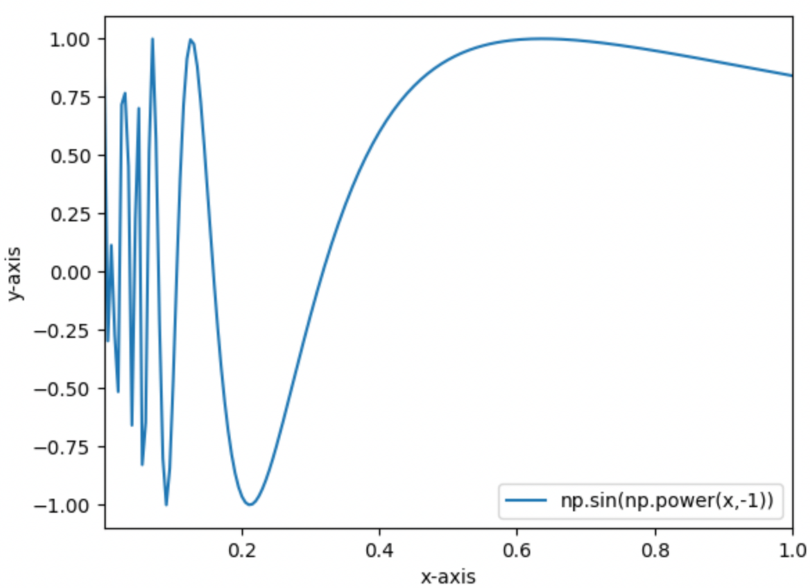

Figure 4. Output for the Graphics Routine.

Task 3

The routine is given by the following:

```python
import numpy as np

def f(x):
    return x * x * x + x * x - 1

# Re-writing f(x)=0 to x = g(x)
def g(x):
    return 1 / np.sqrt(1 + x)

def fixedPointIteration(x0, e, N):
    print('\n\n*** FIXED POINT ITERATION ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        x1 = g(x0)
        print('Iteration-%d, x1 = %0.6f and f(x1) = %0.6f' % (step, x1, f(x1)))
        x0 = x1
        step = step + 1
        if step > N:
            flag = 0
            break
        condition = abs(f(x1)) > e
    if flag == 1:
        print('\nRequired root is: %0.8f' % x1)
    else:
        print('\nNot Convergent.')

# Input Section
x0 = float(input('Guess the root: '))
e = float(input('Tolerable error: '))
N = int(input('Maximum steps: '))

fixedPointIteration(x0, e, N)
```

And the result is given by:

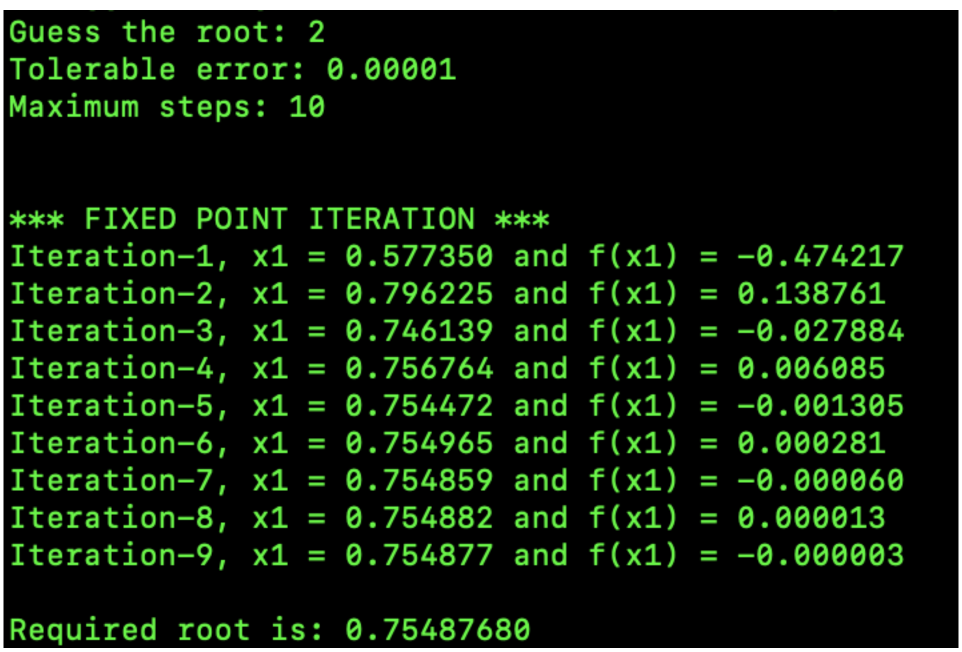

Figure 5. Fixed point method for an example function.

Task 4

Use the code for the previous task, I got the following result.

```
Guess the root: 2
Tolerable error: 0.5
Maximum steps: 20


*** FIXED POINT ITERATION ***
Iteration-1, x1 = 0.000086 and f(x1) = -0.000516

Required root is: 0.00008602
```

Figure 6. Fixed point method for $f(x) = xe^{3x^2} - 7x$.

Task 5

The routine is given by the following:

```python
import numpy as np

def f(x):
    return x * np.exp(3*np.power(x, 2)) - 7*x

def bisection(a,b,N):
    if f(a)*f(b) >= 0:
        print("Bisection method fails.")
        return None
    a_n = a
    b_n = b
    for n in range(1, N+1):
        m_n = (a_n + b_n)/2
        f_m_n = f(m_n)
        if f(a_n)*f_m_n < 0:
            a_n = a_n
            b_n = m_n
        elif f(b_n)*f_m_n < 0:
            a_n = m_n
            b_n = b_n
        elif f_m_n == 0:
            print("Found exact solution:", f_m_n)
            return m_n
        else:
            print("Bisection method fails.")
            return None
    return (a_n + b_n)/2

a = float(input('Guess the first root: '))
b = float(input('Guess the second root: '))
N = int(input('Maximum steps: '))

bisection(a, b, N)
```

And the result is given by:

```
[xianggao@Xiangs-MacBook-Pro Tasksheet_04 % python Task_5.py
Guess the first root: -0.5
Guess the second root: 0.5
Maximum steps: 20
('Found exact solution:', 0.0)
```

Figure 7. Bisection method for $f(x) = xe^{3x^2} - 7x$.

Task 6

From this website[1] I found, I realize that root finding methods are important when we can't find a root for a really complicated system, and finding them numerically is sometimes are our best shot.
There are also different kinds of shared libraries[2]. For examples:

1. Static Libraires;
2. Global Shared Libraries;
3. Folder-level Shared Libraries;
4. Automatic Shared Libraries

The pro of static libraries is its simplicity, however, you are bounded to program statically (doesn't really know what it means, but it sounds like a con).

[1]https://math.stackexchange.com/questions/29197/applied-math-finding-roots
[2]https://www.jenkins.io/doc/book/pipeline/shared-libraries/