exercise1-newton (Score: 13.0 / 13.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 1.0 / 1.0)
3. Test cell (Score: 1.0 / 1.0)
4. Written response (Score: 1.0 / 1.0)
5. Test cell (Score: 1.0 / 1.0)
6. Coding free-response (Score: 2.0 / 2.0)
7. Test cell (Score: 1.0 / 1.0)
8. Coding free-response (Score: 2.0 / 2.0)
9. Written response (Score: 3.0 / 3.0)

Lab 2

1. 提交作業之前，建議可以先點選上方工具列的**Kernel**，再選擇**Restart & Run All**，檢查一下是否程式跑起來都沒有問題，最後記得儲存。
2. 請先填上下方的姓名(name)及學號(stduent_id)再開始作答，例如：

```
name = "我的名字"
student_id= "B06201000"
```

3. 四個求根演算法的實作可以參考lab-2 (https://yuanyuyuan.github.io/itcm/lab-2.html)，裡面有教學影片也有範例程式可以套用。
4. **Deadline: 10/9(Wed.)**

In [1]:

```
name = "黃宇文"
student_id = "B06201029"
```

Exercise 1 - Newton

Use the Newton's method to find roots of

$$f(x) = cosh(x) + cos(x) - c, \text{for } c = 1, 2, 3,$$

Import libraries

In [2]:

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

1. Define the function $g(c)(x) = f(x) = cosh(x) + cos(x) - c$ with parameter $c = 1, 2, 3$ and its derivative df.

In [3]:

(Top)

```python
def g(c):
    assert c == 1 or c == 2 or c == 3
    def f(x):
        return np.cosh(x) + np.cos(x) - c
    return f

def df(x):
    return np.sinh(x) - np.sin(x)
```

Pass the following assertion.

In [4]:

cell-b59c94b754b1fc9e (Top)

```python
assert g(1)(0) == np.cosh(0) + np.cos(0) - 1
assert df(0) == 0
### BEGIN HIDDEN TESTS
assert g(2)(0) == np.cosh(0) + np.cos(0) - 2
assert g(3)(0) == np.cosh(0) + np.cos(0) - 3
assert df(1) == np.sinh(1) - np.sin(1)
### END HIDDEN TESTS
```

2. Implement the algorithm

In [5]:

(Top)

```python
def newton(
    func,
    d_func,
    x_0,
    tolerance=1e-7,
    max_iterations=5,
    report_history=False
):
    '''
    Parameters
    ----------
    func : function
        The target function.
    d_func : function
        The derivative of the target function.
    x_0 : float
        Initial guess point for a solution f(x)=0.
    tolerance : float
        One of the termination conditions. Error tolerance.
    max_iterations : int
        One of the termination conditions. The amount of iterations allowed.
    report_history: bool
        Whether to return history.

    Returns
    -------
    solution : float
        Approximation of the root.
    history: dict
        Return history of the solving process if report_history is True.
    '''

     # Set the initial conditions
    x_n = x_0
    num_iterations = 0

    # history of solving process
    if report_history:
        history = {'estimation': [], 'error': []}
```

```
    while True:

        # Find the value of f(x_n)
        f_of_x_n = func(x_n)

        # Evaluate the error
        error = abs(f_of_x_n)

        if report_history:
            history['estimation'].append(x_n)
            history['error'].append(error)

        # Satisfy the criterion and stop
        if error < tolerance:
            print('Found solution after', num_iterations,'iterations.')
            if report_history:
                return x_n, history
            else:
                return x_n

        # Find the differential value of f'(x_n)
        d_f_of_x_n = d_func(x_n)

        # Avoid zero derivative
        if d_f_of_x_n == 0:
            print('Zero derivative. No solution found.')
            if report_history:
                return None, history
            else:
                return None

        # Check the number of iterations
        if num_iterations < max_iterations:
            num_iterations += 1

            # Find the next approximation solution
            x_n = x_n - f_of_x_n / d_f_of_x_n

        # Satisfy the criterion and stop
        else:
            print('Terminate since reached the maximum iterations.')
            if report_history:
                return x_n, history
            else:
                return x_n
```

Test your implementation with the assertion below.

In [6]:

cell-4d88293f2527c82d (Top)

```
root = newton(
    lambda x: x**2 - x - 1,
    lambda x: 2*x - 1,
    1.2,
    max_iterations=100,
    tolerance=1e-7,
    report_history=False
)
assert abs(root - ((1 + np.sqrt(5)) / 2)) < 1e-7
```

```
Found solution after 4 iterations.
```

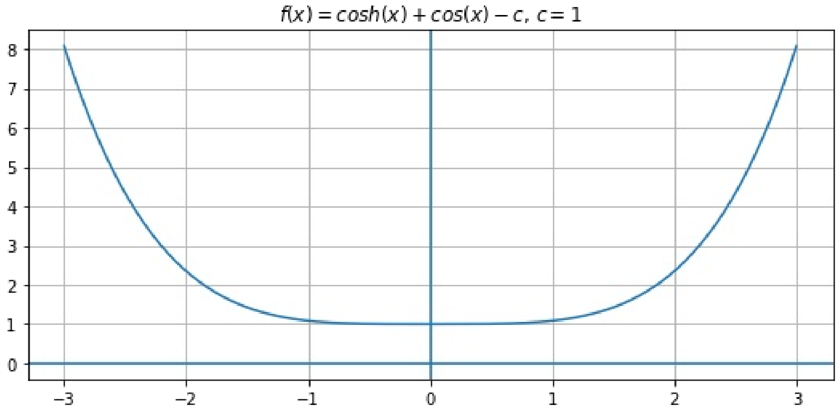
3. Answer the following questions under the case $c = 1$.

Plot the function to find an interval that contains the zero of f if possible.

In [7]:

(Top)

```
c = 1
f = g(c)

search_range = np.arange(-3,3,0.0001)
fig, ax = plt.subplots(figsize=(9, 4))
ax.plot(search_range, f(search_range))
ax.set_title(r'$f(x)=cosh(x)+cos(x)-c$, $c=$%d' % c)
ax.grid(True)
ax.axhline(y=0)
ax.axvline(x=0)
plt.show()
```



According to the figure above, estimate the zero of f.

For example,

```
root = 3        # 單根
root = -2, 1    # 多根
root = None     # 無解
```

In [8]:

(Top)

```
root = None
```

In [9]:

cell-d872c7c57f11c968 (Top)

```
print('My estimation of root:', root)
### BEGIN HIDDEN TESTS
if root == None:
    print('Right answer!')
else:
    raise AssertionError('Wrong answer!')
### END HIDDEN TESTS
```

```
My estimation of root: None
Right answer!
```

Try to find the zero with a tolerance of 10^{-10}. If it works, plot the error and estimation of each step. Otherwise, state the reason why the method failed on this case.

(Top)

The function $cosh(x) + cos(x) - 1$ is always positive. It has no roots.

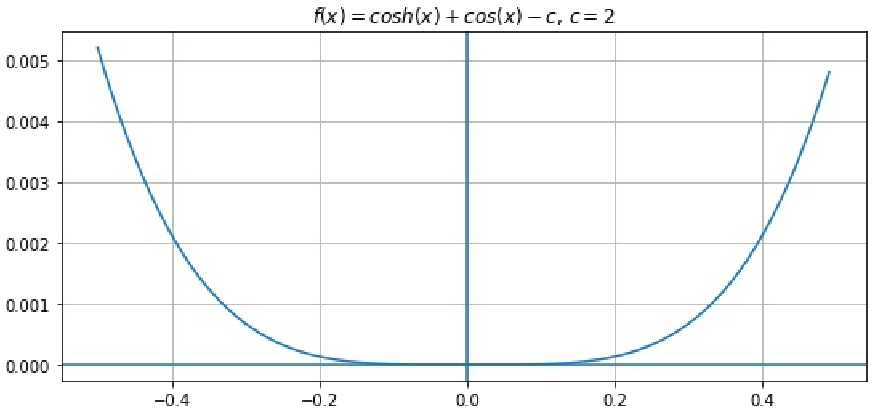
4. Answer the following questions under the case $c = 2$.

Plot the function to find an interval that contains the zero of f if possible.

In [10]:

(Top)

```
c = 2
f = g(c)

search_range = np.arange(-0.5,0.5,0.01)
fig, ax = plt.subplots(figsize=(9, 4))
ax.plot(search_range, f(search_range))
ax.set_title(r'$f(x)=cosh(x)+cos(x)-c$, $c=$%d' % c)
ax.grid(True)
ax.axhline(y=0)
ax.axvline(x=0)
plt.show()
```



According to the figure above, estimate the zero of f.

For example,

```
root = 3        # 單根
root = -2, 1    # 多根
root = None     # 無解
```

In [11]:

(Top)

```
root = newton(
    g(2),
    df,
    0.2,
    max_iterations=100,
    tolerance=1e-7,
    report_history=False
)
```

```
Found solution after 7 iterations.
```

In [12]:

cell-20fddbe6fa4c437b (Top)

```
print('My estimation of root:', root)

### BEGIN HIDDEN TESTS
assert type(root) is float or int, 'Wrong type!'
### END HIDDEN TESTS
```

```
My estimation of root: 0.02669678974427797
```

Try to find the zero with a tolerance of 10^{-10}. If it works, plot the error and estimation of each step. Otherwise, state the reason why the method failed on this case.

In [13]:

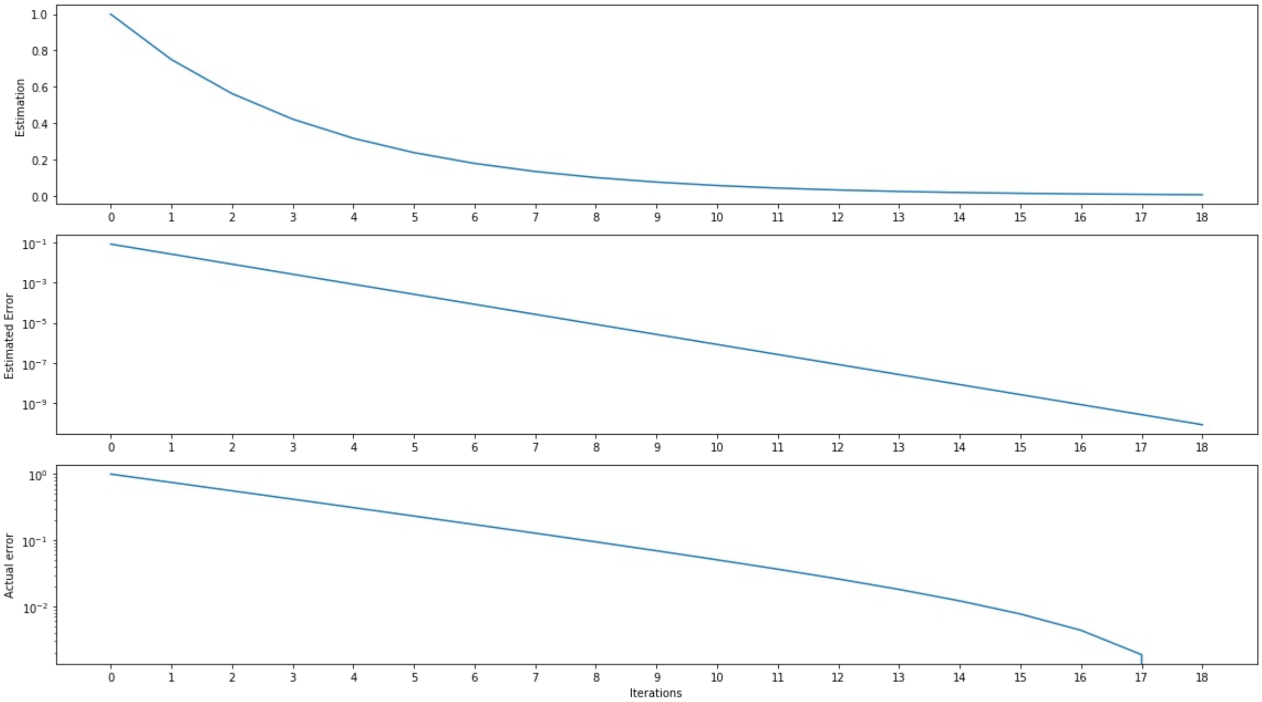
(Top)

```
solution, history = newton(
    g(2),
    df,
    1,
    tolerance=1e-10,
    max_iterations=100,
    report_history=True
)
print(solution)
```

```
Found solution after 18 iterations.
0.005639347364278358
```

In [14]:

```
fig, axes = plt.subplots(3, 1, figsize=(16, 9))
ax1, ax2, ax3 = axes

num_iterations = len(history['estimation'])
iterations = range(num_iterations)
for ax in axes:
    ax.set_xticks(iterations)

# Plot the estimation in history
ax1.plot(iterations, history['estimation'])
ax1.set_ylabel('Estimation')

# Plot the estimation error (log(error)) in history
ax2.plot(iterations, history['error'])
ax2.set_ylabel('Estimated Error')
ax2.set_yscale('log')

# Plot the estimation actual error (estimation - exact solution) in history
actual_error = np.abs(history['estimation']-solution)
ax3.plot(iterations, actual_error)
ax3.set_ylabel('Actual error')
ax3.set_yscale('log')
ax3.set_xlabel('Iterations')

plt.tight_layout()
plt.show()
```



---

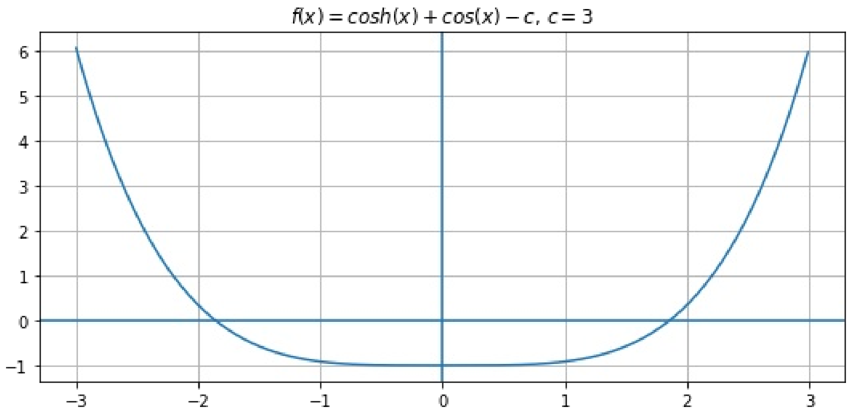
5. Answer the following questions under the case $c = 3$.

Plot the function to find an interval that contains the zeros of f if possible.

In [15]:

(Top)

```
c = 3
f = g(c)

search_range = np.arange(-3.0, 3.0, 0.01)
fig, ax = plt.subplots(figsize=(9, 4))
ax.plot(search_range, f(search_range))
ax.set_title(r'$f(x)=cosh(x)+cos(x)-c$, $c=$%d' % c)
ax.grid(True)
ax.axhline(y=0)
ax.axvline(x=0)
plt.show()
```



According to the figure above, estimate the zero of f.

For example,

```
root = 3        # 單根
root = -2, 1    # 多根
root = None     # 無解
```

In [16]:

(Top)

```
root_1 = newton(
    lambda x: np.cosh(x) + np.cos(x) - 3,
    lambda x: np.sinh(x) - np.sin(x),
    2,
    max_iterations=100,
    tolerance=1e-7,
    report_history=False
)

root_2 = newton(
    lambda x: np.cosh(x) + np.cos(x) - 3,
    lambda x: np.sinh(x) - np.sin(x),
    -2,
    max_iterations=100,
    tolerance=1e-7,
    report_history=False
)

root = root_1, root_2
```

```
Found solution after 3 iterations.
Found solution after 3 iterations.
```

In [17]:

cell-06ec0b20844075c7 (Top)

```
print('My estimation of root:', root)

### BEGIN HIDDEN TESTS
assert type(root) == tuple, 'Should be multiple roots!'
### END HIDDEN TESTS
```

```
My estimation of root: (1.8579208547469979, -1.8579208547469979)
```

Try to find the zero with a tolerance of 10^{-10}. If it works, plot the error and estimation of each step. Otherwise, state the reason why the method failed on this case.

In [18]:

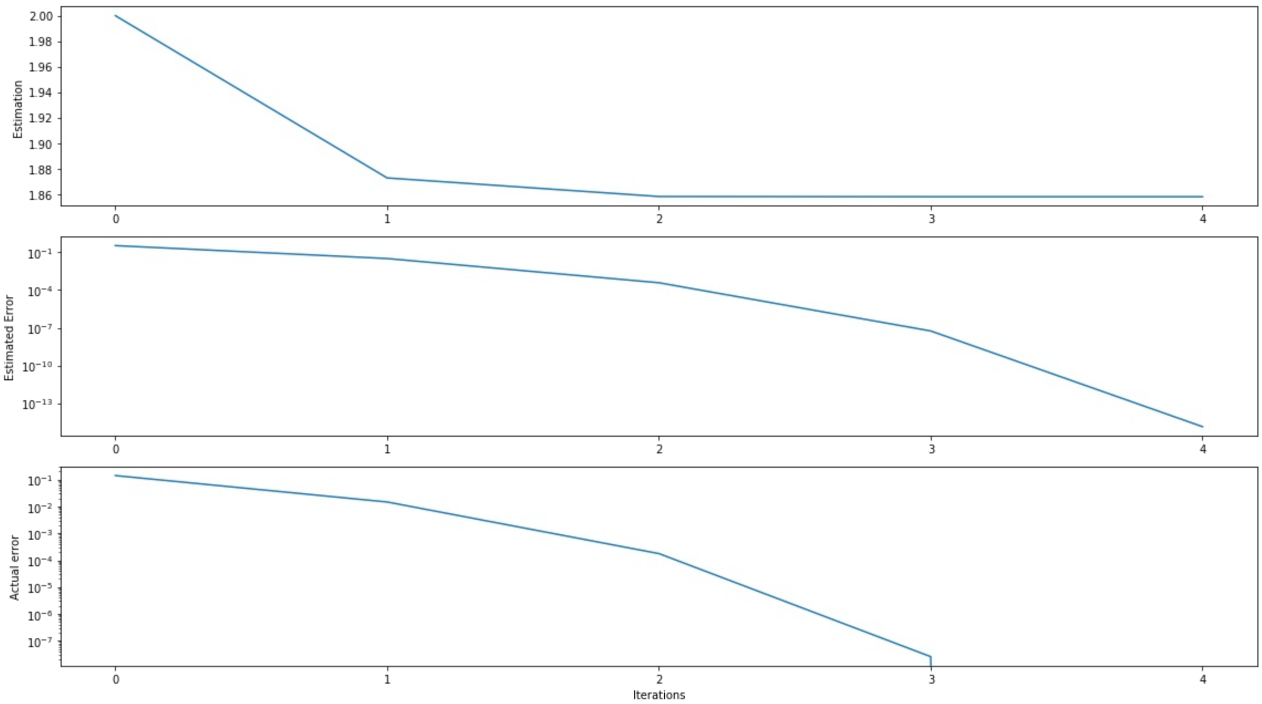
(Top)

```
solution, history = newton(
    g(3),
    df,
    2,
    tolerance=1e-10,
    max_iterations=100,
    report_history=True
)
print(solution)
```

```
Found solution after 4 iterations.
1.8579208291501987
```

In [19]:

```
fig, axes = plt.subplots(3, 1, figsize=(16, 9))
ax1, ax2, ax3 = axes

num_iterations = len(history['estimation'])
iterations = range(num_iterations)
for ax in axes:
    ax.set_xticks(iterations)

# Plot the estimation in history
ax1.plot(iterations, history['estimation'])
ax1.set_ylabel('Estimation')

# Plot the estimation error (log(error)) in history
ax2.plot(iterations, history['error'])
ax2.set_ylabel('Estimated Error')
ax2.set_yscale('log')

# Plot the estimation actual error (estimation - exact solution) in history
actual_error = np.abs(history['estimation']-solution)
ax3.plot(iterations, actual_error)
ax3.set_ylabel('Actual error')
ax3.set_yscale('log')
ax3.set_xlabel('Iterations')

plt.tight_layout()
plt.show()
```



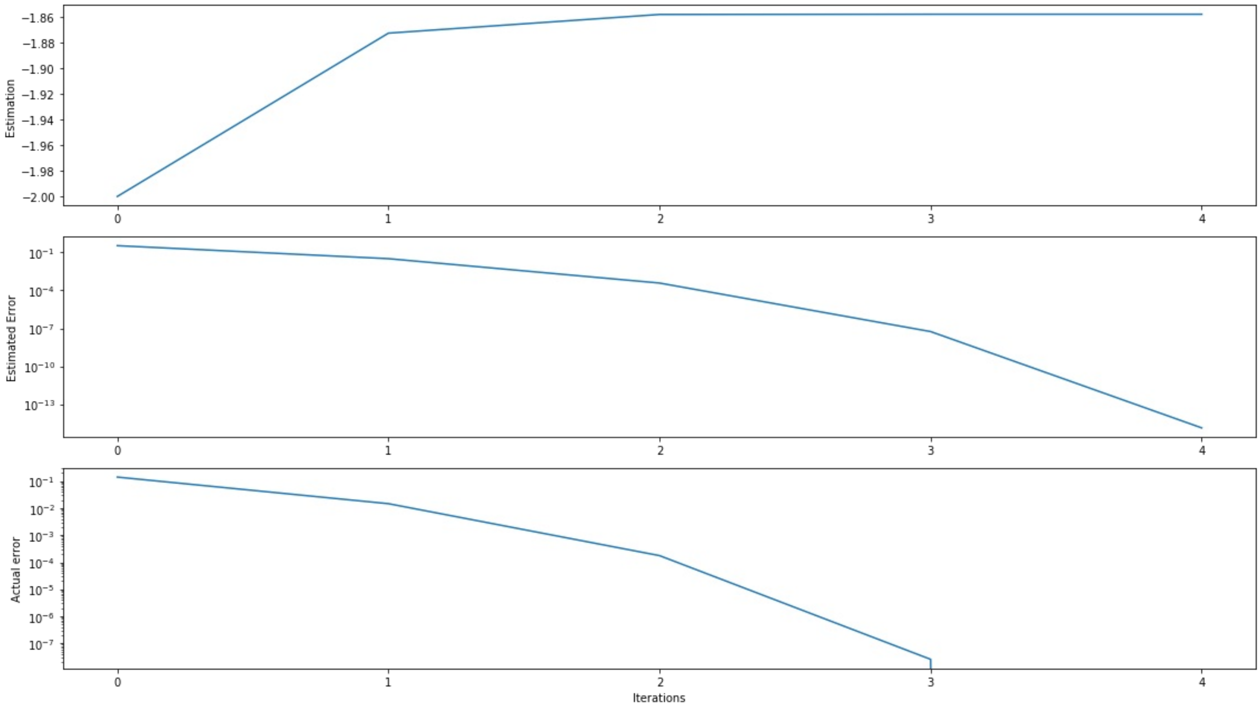
In [20]:

```
solution, history = newton(
    g(3),
    df,
    -2,
    tolerance=1e-10,
    max_iterations=100,
    report_history=True
)
print(solution)
```

```
Found solution after 4 iterations.
-1.8579208291501987
```

In [21]:

```
fig, axes = plt.subplots(3, 1, figsize=(16, 9))
ax1, ax2, ax3 = axes

num_iterations = len(history['estimation'])
iterations = range(num_iterations)
for ax in axes:
    ax.set_xticks(iterations)

# Plot the estimation in history
ax1.plot(iterations, history['estimation'])
ax1.set_ylabel('Estimation')

# Plot the estimation error (log(error)) in history
ax2.plot(iterations, history['error'])
ax2.set_ylabel('Estimated Error')
ax2.set_yscale('log')

# Plot the estimation actual error (estimation - exact solution) in history
actual_error = np.abs(history['estimation']-solution)
ax3.plot(iterations, actual_error)
ax3.set_ylabel('Actual error')
ax3.set_yscale('log')
ax3.set_xlabel('Iterations')

plt.tight_layout()
plt.show()
```



Discussion

For all cases above(c=1,2,3), do the results(e.g. error behaviors, estimations, etc) agree with the theoretical analysis?

(Top)

For $c = 1$, the function does not satisfy the prerequisties.\ For $c = 2$, although the function cut the x axis at only one point, Newton method is able to deal with the case. It converges poorly. \ For $c = 3$, the results agree with the theoretical analysis. Newton method has a poor convergence as the estimated error does not converge linearly.

In [ ]: