exercise2 (Score: 11.0 / 14.0)

1. Written response (Score: 0.0 / 3.0)
2. Comment
3. Test cell (Score: 1.0 / 1.0)
4. Test cell (Score: 1.0 / 1.0)
5. Test cell (Score: 2.0 / 2.0)
6. Test cell (Score: 3.0 / 3.0)
7. Test cell (Score: 1.0 / 1.0)
8. Test cell (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 = "歐陽秉志"
student_id = "B05201012"
```

Exercise 2

Kepler's equation

In celestial mechanics, *Kepler's equation*

$$M = E - e\sin(E)$$

relates the mean anomaly M to the eccentric anomaly E of an elliptical orbit of eccentricity e, where $0 < e < 1$, see *Wiki website* (https://en.wikipedia.org/wiki/Kepler's_laws_of_planetary_motion) for the details.

1. Prove that fixed-point iteration using the iteration function

$$g(E) = M + e\sin(E)$$

is convergent locally.

[Hint: You may use Ostrowski's Theorem mentioned in the lecture note.]

proof.

(Top)

請點此cell兩下開始作答（如要打文字記得選Markdown, 寫程式則選Code, 一個cell不夠可以再新增在下方）

Comments:
No response.

---

2. Use the fixed-point iteration scheme in "Q.1" to solve Kepler’s equation for the eccentric anomaly E corresponding to a mean anomaly $M = \frac{2\pi}{3}$ and an eccentricity $e = 0.5$.

Part 0. Import libraries

In [2]:

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

Part 1. Define the fixed point function

In [3]:

(Top)

```python
def fixed_point(
    func,
    x_0,
    tolerance=1e-7,
    max_iterations=5,
    report_history=False,
):
    '''Approximate solution of f(x)=0 on interval [a,b] by the secant method.

    Parameters
    ----------
    func : function
        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 : (positive) integer
        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.
    '''

    # 請參考 hands-on 的 fixed point method
    #
    # ===== 請實做程式 =====
    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 - 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

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

            # Find the next approximation solution
            x_n = 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 implementaion with the assertion below.

In [4]:

```
          test_fixed_method                                                                    (Top)
```

```
root = fixed_point(lambda x: x - (x**2 - 4*x + 3.5), 2, tolerance=1e-7, max_iterations=100, report_histor
y=False)

error = np.inf
for solution in np.roots([1, -4, 3.5]):
    if abs(root - solution) < error:
        exact_solution = solution
        error = abs(root - solution)

assert error < 1e-7
```

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

Part 2. Assign values to variables anomaly mean "*M*" and eccentricity "*e*".

$$M = \frac{2\pi}{3} \quad \text{and} \quad e = 0.5$$

In [5]:

```
                                                                                               (Top)
```

```
# Hint:
# M = ?
# e =?
# ===== 請實做程式 =====
M = 2*np.pi/3
e = 0.5
# ====================
```

In [6]:

```
          M_and_e                                                                              (Top)
```

```
print('M =', M)
print('e =', e)

### BEGIN HIDDEN TESTS
assert M == 2*np.pi/3, 'M is wrong!'
assert e == 0.5, 'e is wrong!'
### END HIDDEN TESTS
```

```
M = 2.0943951023931953
e = 0.5
```

Part 3. Define the function of Kepler’s equation

Recall Kepler’s equation **:**

$$M = E - e\sin(E).$$

So we let the function $f(E) = E - e\sin(E) - M$, then

$$g(E) = E - f(E) = M + e\sin(E)$$

For the instance:

If we want to implement "$\sin(x)$", we will call `np.sin(x)` with numpy in python.

In [7]:

(Top)

```
def f(E):
    # Hint: return ...
    # ===== 請實做程式 =====
    return E - e*np.sin(E) - M
    # ====================

def g(E):
    # Hint: return ...
    # ===== 請實做程式 =====
    return E - f(E)
    # ====================
```

In [8]:

test_f_and_g (Top)

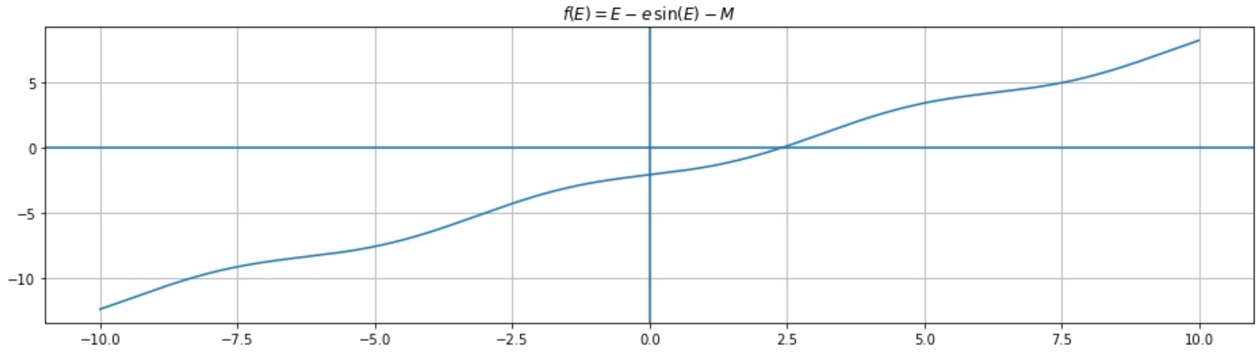
```
print('M =', M)

# f(0) = -M, g(0) = M
print('f(0) =', f(0))
print('g(0) =', g(0))

### BEGIN HIDDEN TESTS
from random import random
rd_number = random()
assert f(rd_number) == rd_number - 0.5*np.sin(rd_number) - 2*np.pi/3, 'f is wrong!'
assert g(rd_number) == 2*np.pi/3 + 0.5*np.sin(rd_number), 'g is wrong!'
### END HIDDEN TESTS
```

```
M = 2.0943951023931953
f(0) = -2.0943951023931953
g(0) = 2.0943951023931953

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-8-0844454f93d4> in <module>
      9 rd_number = random()
     10 assert f(rd_number) == rd_number - 0.5*np.sin(rd_number) - 2*np.pi/3, 'f is wrong!'
---> 11 assert g(rd_number) == 2*np.pi/3 + 0.5*np.sin(rd_number), 'g is wrong!'
     12 ### END HIDDEN TESTS

AssertionError: g is wrong!
```

Part 4. Plot the function $f(E)$ and $g(E)$

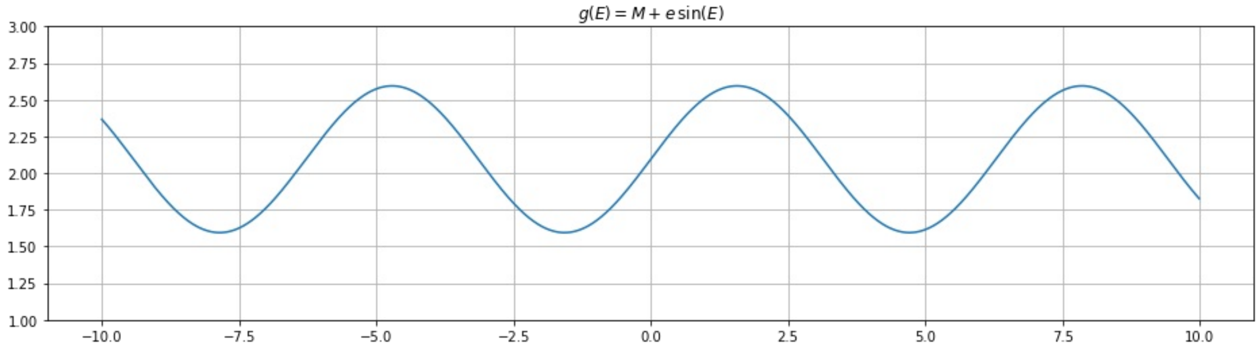
In [9]:

```
fig, ax = plt.subplots(figsize=(16, 4))
search_range = np.arange(-10, 10, 0.01)
ax.plot(search_range, f(search_range))
ax.set_title(r'$f(E) = E - e\,\sin(E) - M$')
ax.grid(True)
ax.axhline(y=0)
ax.axvline(x=0)
plt.show()
```



In [10]:

```
fig, ax = plt.subplots(figsize=(16, 4))
search_range = np.arange(-10, 10, 0.01)
ax.plot(search_range, g(search_range))
ax.set_title(r'$g(E) = M + e\,\sin(E)$')
ax.grid(True)
ax.axhline(y=0)
plt.ylim(1,3)
plt.show()
```



Part 5. Find the solution of "E"

In [11]:

(Top)

```
init_pt = 2.5

root, history = fixed_point(
    # ===== 請實做程式 =====
    g,
    init_pt,
    tolerance=1e-7,
    max_iterations=100,
    report_history=True
    # ===================
)
```

```
Found solution after 15 iterations.
```

In [12]:

the_root_of_E (Top)

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

### BEGIN HIDDEN TESTS
assert abs(root - 2.425) < 0.002, 'root is wrong!'
### END HIDDEN TESTS
```

```
My estimation of root: 2.4234054245671937
```

---

3. An " *exact* " formula for E is known:

$$E = M + 2\sum_{m=1}^{\infty}\frac{1}{m}J_m(me)\sin(mM);$$

where $J_m(x)$ is the Bessel function of the first kind of order m.

Use this formula to compute E. How many terms are needed to produce the value obtained in "Q.2" until convergence?

Part 0. Import package

In [13]:

```python
from scipy.special import jn  # Bessel function
```

Part 1. Define the function

For the convenience, we define the function $h(m)$ as

$$h(m) \triangleq \frac{2}{m}J_m(me)\sin(mM)$$

If we want to implement " ***Bessel function*** " $J_m(x)$, we can call `jn(m,x)` in Python.

In [14]:

(Top)

```python
def h(m):
    # Hint: return ...
    # ===== 請實做程式 =====
    return (2/m)*jn(m, m*e)*np.sin(m*M)
    # ====================
```

In [15]:

```
        h                                                                       (Top)
```

```
# test the function of h
print('h(1) =', h(1))
assert round(h(1), 5) == 0.41962

### BEGIN HIDDEN TESTS
from random import random
rd_number = random()
assert h(rd_number) == 2*jn(rd_number, rd_number*0.5)*np.sin(rd_number*(2*np.pi/3))/rd_number, 'h is wron
g!'
### END HIDDEN TESTS
```

```
h(1) = 0.41962127776423175

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-15-69cf0965b94d> in <module>
      6 from random import random
      7 rd_number = random()
----> 8 assert h(rd_number) == 2*jn(rd_number, rd_number*0.5)*np.sin(rd_number*(2*np.pi/3))/r
d_number, 'h is wrong!'
      9 ### END HIDDEN TESTS

AssertionError: h is wrong!
```

Part 2. Find how many terms we need to achieve the result obtained Q.2 in a tolerance 10^{-7}.

That is to find _num*terms* such that

$$\left|\text{root} - \left(M + \sum_{k=1}^{\text{num\_terms}} h(k)\right)\right| < 10^{-7}$$

For example, the following cell shows the implmentation with only 1 term.

In [16]:

```
LHS = root
RHS = M + h(1)
error = abs(LHS-RHS)
print('Left hand side is the estimation of root by the fixed-point method:', LHS)
print('Right hand side is the approximation by the formula in only 1 term:', RHS)
print('The error between LHS and RHS:', error)
```

```
Left hand side is the estimation of root by the fixed-point method: 2.4234054245671937
Right hand side is the approximation by the formula in only 1 term: 2.514016380157427
The error between LHS and RHS: 0.09061095559023347
```

In [17]:

```
                                                                                (Top)
```

```
LHS = root
RHS = M + h(1)
num_terms = 1
tolerance = 1e-7

# ===== 請實做程式 =====
while error >= tolerance:
    h_1k = [h(i) for i in range(1, num_terms+1)]
#     print(h_1k)
    num_terms += 1
    RHS = M + sum(h_1k)
    error = abs(LHS - RHS)
print(error)
# ====================
```

```
4.331598635332057e-09
```

In [18]:

number_of_term

(Top)

```
print('Number of terms to approximate:', num_terms)

### BEGIN HIDDEN TESTS
assert num_terms > 20 , '%d is too few!' % num_terms
### END HIDDEN TESTS
```

```
Number of terms to approximate: 24
```