exercise1 (Score: 17.0 / 17.0)

Lab 3

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. 演算法的實作可以參考[lab-3 (https://yuanyuyuan.github.io/itcm/lab-3.html)](https://yuanyuyuan.github.io/itcm/lab-3.html), 有任何問題歡迎找助教詢問。
4. **Deadline: 10/30(Wed.)**

In [1]:

```python
name = "馬宗儀"
student_id = "b06201006"
```

Exercise 1

Let $g(x) = \ln(4 + x - x^2)$ and α is a fixed point of $g(x)$ i.e. $\alpha = g(\alpha)$.

- ### Part A. Implement your fixed-point algorithm and solve it with initial guess $x_0 = 2$ within tolerance 10^{-10}, and answer the questions of error behavior analysis below.
- ### Part B. Redo Part A. by applying Aitken's acceleration.

Import libraries

In [2]:

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

Implement the target function $g(x) = \ln(4 + x - x^2)$

In [3]:

(Top)

```python
def g(x):
    # ===== 請實做程式 =====
    return np.log(4+x-x*x)
    # ===================
```

In [4]:

cell-c0f08330aec65e17 (Top)

```
assert round(g(0), 4) == 1.3863
### BEGIN HIDDEN TESTS
import random
x = random.random()
assert g(x) == np.log(4 + x - x**2), 'Failed on x = %f' % x
### END HIDDEN TESTS
```

Run built-in [fixed-point method (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fixed_point.html#rf(1)](https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fixed_point.html) with Python SciPy, and use this accurate value as the fixed point α

In [5]:

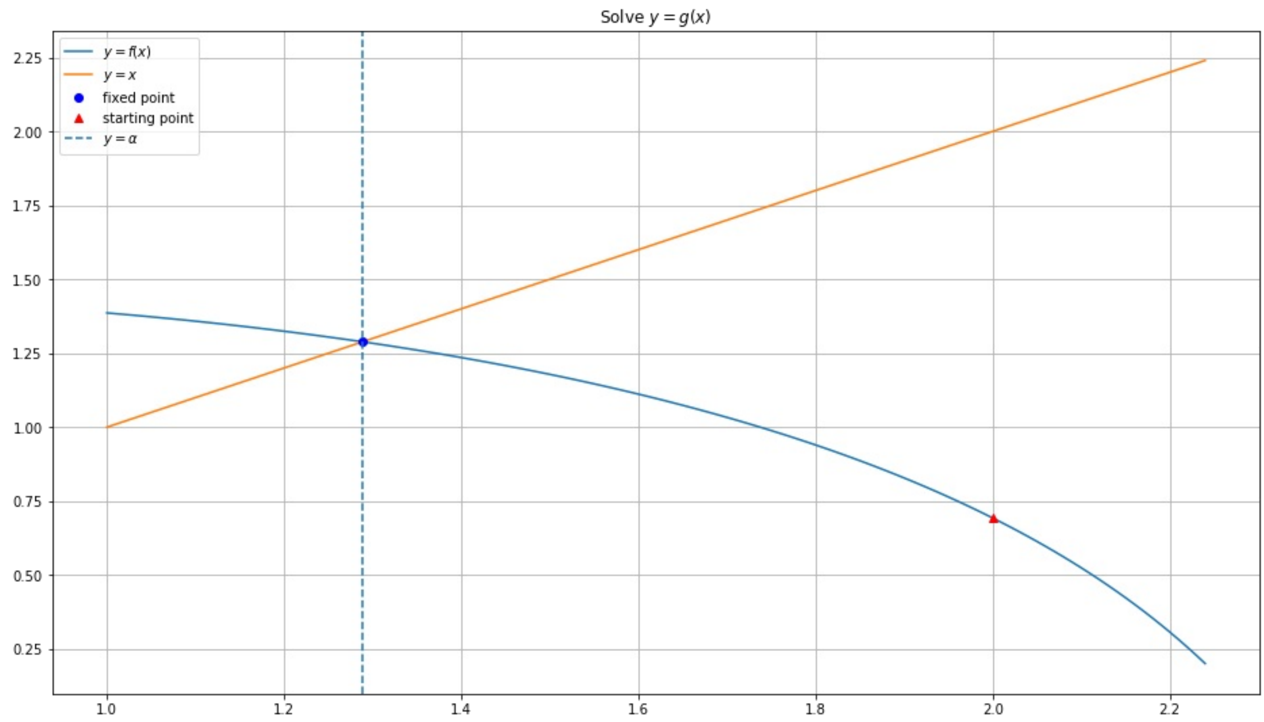
```
from scipy import optimize
alpha = optimize.fixed_point(g, x0=2, xtol=1e-12)
print('The fixed point is', alpha)
```

```
The fixed point is 1.2886779668238684
```

Visualization

In [6]:

```
x_range = np.arange(1, 2.25, 0.01)
plt.figure(figsize=(16, 9))
plt.title(r'Solve $y=g(x)$')
plt.plot(x_range, g(x_range), label=r'$y=f(x)$')
plt.plot(x_range, x_range, label=r'$y=x$')
plt.plot(alpha, g(alpha), 'bo', label='fixed point')
plt.plot(2.0, g(2.0), 'r^', label='starting point')
plt.axvline(x=alpha, linestyle='--', label=r'$y=\alpha$')
plt.gca().legend()
plt.grid()
plt.show()
```



Part A.

(Top)

1. Find the fixed point of $g(x)$ using your fixed-point iteration to within tolerance 10^{-10} with initial guess $x_0 = 2$.

1-1. Implement the fixed point method

In [7]:

(Top)

```
def fixed_point(
    func,
    x_0,
    tolerance=1e-7,
    max_iterations=5
):
    '''Find the fixed point of the given function func

    Parameters
    ----------
    func : function
        The target function.
    x_0 : float
        Initial guess point for a solution func(x)=x.
    tolerance: float
        One of the termination conditions. Error tolerance.
    max_iterations : (positive) integer
        One of the termination conditions. The amount of iterations allowed.

    Returns
    -------
    solution : float
        Approximation of the root.
    history: dict
        Return history of the solving process
        history: {'x_n': list}
    '''

    # ===== 請實做程式 =====
    x_n=x_0
    iterations=0
    history={'x_n':[]}

    while True:
        f_n=func(x_n)
        error=abs(f_n-x_n)
        history['x_n'].append(x_n)

        if error<tolerance :
            print('Found solution after', iterations, 'iterations.')
            return (x_n, history)

        elif iterations>=max_iterations:
            print('Terminate since reached the maximum iterations.')
            return (x_n, history)

        else:
            iterations+=1
            x_n=f_n
    # ====================
```

1-2. Find the root

In [8]:

(Top)

```
solution, history = fixed_point(
    # ===== 請實做程式 =====
    g,
    x_0=2.0,
    tolerance=1e-10,
    max_iterations=1000
    # ====================
)
```

```
Found solution after 28 iterations.
```

In [9]:

cell-2d72f68109ee500c (Top)

```
print('My estimation is', solution)
### BEGIN HIDDEN TESTS
assert np.round(solution, 9) == np.round(alpha, 9), 'Wrong answer!'
### END HIDDEN TESTS
```

```
My estimation is 1.2886779668876651
```

(Top)

2. Estimate graphically the asymptotic error constant C

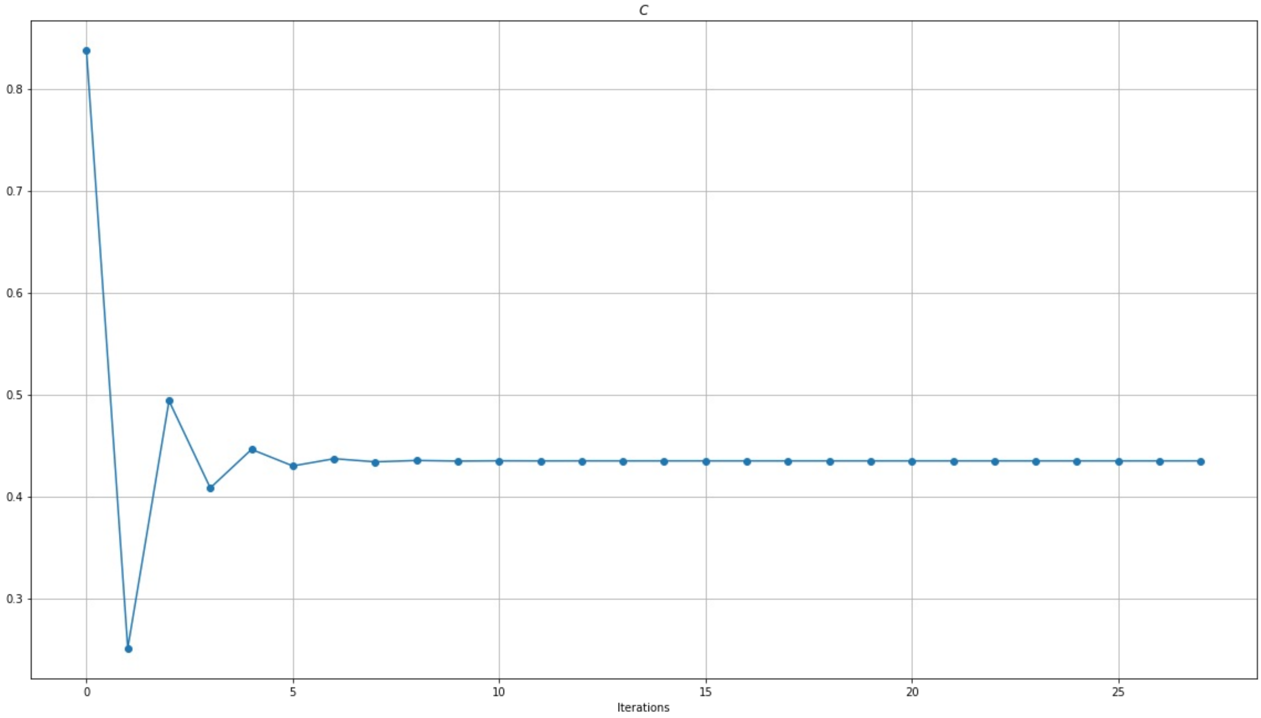
$$\lim_{n \to \infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|} = C$$

In [10]:

(Top)

```
'''
Hint:
    1. Prepare the sequences: x_n(from the history of algorithm)
    2. Compute the error of sequence: e_n
    3. Compute the sequence: e_{n+1}/e_{n}
    4. Plot the curve
    5. Fill in the name of x,y axes
    6. Show the plot
'''
# ===== 請實做程式 =====
x_n=history['x_n']
e_n=abs(alpha-x_n)

plt.figure(figsize=(16,9))
plt.plot(e_n[1:]/e_n[:-1],'o-')
plt.title(r'$C$')
plt.tight_layout()
plt.xlabel('Iterations')
plt.grid()
plt.show()
# ====================
```



Part B.

(Top)

1. Accelerate the convergence of the sequence $\{x_n\}$ obtained in *Part A.* using Aitken's Δ^2 method, yielding sequence $\{\hat{x}_n\}$.

1-1. Introduce Aitken's acceleration into the original method.

In [11]:

(Top)

```
def aitken(
    func,
    x_0,
    tolerance=1e-7,
    max_iterations=5,
):
    '''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)=x.
    tolerance: float
        One of the termination conditions. Error tolerance.
    max_iterations : (positive) integer
        One of the termination conditions. The amount of iterations allowed.

    Returns
    -------
    solution : float
        Approximation of the root.
    history: dict
        Return history of the solving process
        history: {'x_n': list}
    '''

    # ===== 請實做程式 =====
    x_n=x_0
    iterations=0
    history={'x_n':[]}

    while True:

        x_1=func(x_0)
        x_2=func(x_1)
        x_n=x_2-((x_2-x_1)**2/((x_2-x_1)-(x_1-x_0)))
        error=abs(x_n-x_0)
        history['x_n'].append(x_n)

        if error<tolerance :
            print('Found solution after', iterations, 'iterations.')
            return (x_n, history)

        elif iterations>=max_iterations:
            print('Terminate since reached the maximum iterations.')
            return (x_n, history)

        else:
            iterations+=1
            x_0=x_n
    # ====================
```

1-2. Find the root

In [12]:

(Top)

```
solution, history = aitken(
    # ===== 請實做程式 =====
    g,
    x_0=2.0,
    tolerance=1e-10,
    max_iterations=1000
    # ====================
)
```

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

In [13]:

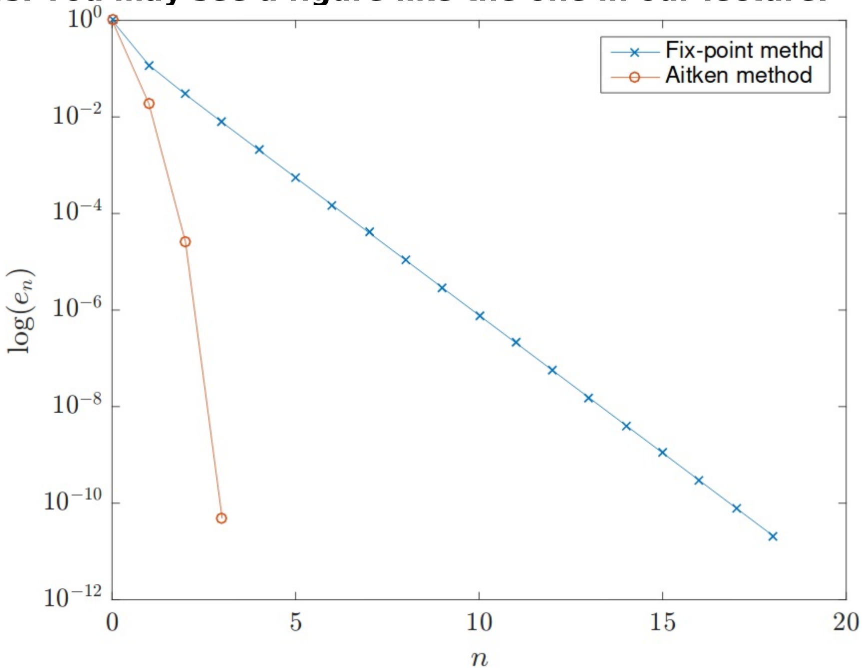
cell-5c862e35ba0aa7d9 (Top)

```
print('My estimation is', solution)
### BEGIN HIDDEN TESTS
assert np.round(solution, 9) == np.round(alpha, 9), 'Wrong answer!'
### END HIDDEN TESTS
```

```
My estimation is 1.2886779668238684
```

(Top)

2. Plot the error curves of each algorithm w.r.t iterations n in log scale to compare the convergence rates. You may see a figure like the one in our lecture.



Ref. Page15 of [cmath2019_note1_aitken.pdf (https://ceiba.ntu.edu.tw/course/7a770d/content/cmath2019_note1_aitken.pdf)](https://ceiba.ntu.edu.tw/course/7a770d/content/cmath2019_note1_aitken.pdf)

In [14]:

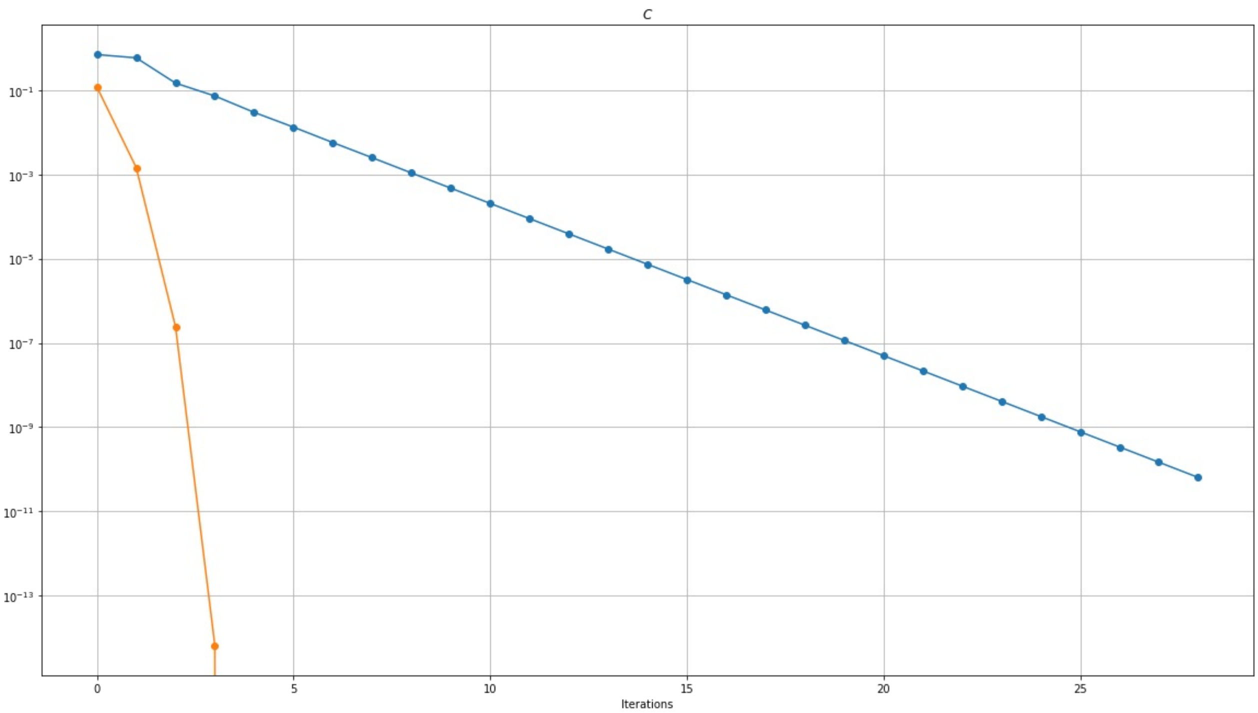
(Top)

```python
'''
Hint:
    1. Prepare the sequences: x_n, x_n_hat(from the history of each algorithm)
    2. Compute the error of sequences: e_n, e_n_hat
    3. Plot the curves of e_n, e_n_hat respectively
    4. Change scale into log
    5. Fill in the name of x,y axes
    6. Enable legend(show curve names)
    7. Show the plot
'''
# ===== 請實做程式 =====
solution, history = fixed_point(g,x_0=2.0,tolerance=1e-10,max_iterations=1000)
x_n=history['x_n']
e_n=abs(alpha-x_n)

solution, history = aitken(g,x_0=2.0,tolerance=1e-10,max_iterations=1000)
y_n=history['x_n']
d_n=abs(alpha-y_n)

plt.figure(figsize=(16,9))
plt.plot(e_n,'o-')
plt.plot(d_n,'o-')
plt.title(r'$C$')
plt.tight_layout()
plt.xlabel('Iterations')
plt.yscale("log")
plt.grid()
plt.show()
# ====================
```

```
Found solution after 28 iterations.
Found solution after 4 iterations.
```



In [ ]: