exercise1 (Score: 20.0 / 20.0)

1. Task (Score: 4.0 / 4.0)
2. Test cell (Score: 2.0 / 2.0)
3. Test cell (Score: 4.0 / 4.0)
4. Test cell (Score: 2.0 / 2.0)
5. Task (Score: 4.0 / 4.0)
6. Task (Score: 4.0 / 4.0)

Lab 4

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-4 (https://yuanyuyuan.github.io/itcm/lab-4.html)](https://yuanyuyuan.github.io/itcm/lab-4.html), 有任何問題歡迎找助教詢問。
4. **Deadline: 11/20(Wed.)**

In [1]:

```
name = "李澤諺"
student_id = "B05902023"
```

Exercise 1. Finite Difference

Part 0.

Import necessary libraries. Note that diags library from scipy is used to construct the differentiation matrix below.

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import diags
```

Part 1.

Given a function $u(x)$ which we want to find its derivative with numerical methods.

Consider a uniform grid partitioning x into $\{x_1, x_2, \ldots, x_n\}$ with grid size $\Delta x = x_{j+1} - x_j, j \in \{1, 2, \ldots, n\}$, and a set of corresponding data values $U = \{U_1, U_2, \ldots, U_n\}$, where

$$U_{j+k} = u(x_j + k\Delta x) = u(x_{j+k}), j \in \{1, 2, \ldots, n\}.$$

We want to use one-sided finite-difference formula

$$\alpha_1 U_j + \alpha_2 U_{j+1} + \alpha_3 U_{j+2}$$

to approximate the derivative of u at all the points $x_j, j \in \{1, 2, \ldots, n\}$, that is

$$u'(x_j) \approx W_j \triangleq \alpha_1 U_j + \alpha_2 U_{j+1} + \alpha_3 U_{j+2}.$$

(Top)

Part 1.1

Find the coefficients α_j for $j = 1, 2, 3$ which make the stencil above accurate for as high degree polynomials as possible.

Write down your derivation in detail with Markdown/LaTeX.

利用method of undetermined coefficients

$$\begin{aligned}
u'(x_j) =& \ \alpha_1 U_j + \alpha_2 U_{j+1} + \alpha_3 U_{j+2} \\
=& \ \alpha_1 u(x_j) + \alpha_2 u(x_{j+1}) + \alpha_3 u(x_{j+2}) \\
=& \ \alpha_1 u(x_j) + \alpha_2 u(x_j + \Delta x) + \alpha_3 u(x_j + 2\Delta x) \\
=& \ \alpha_1 u(x_j) + \alpha_2 (u(x_j) + u'(x_j)\Delta x + \frac{u''(x_j)}{2}\Delta x^2 + \cdots) + \alpha_3 (u(x_j) + u'(x_j)(2\Delta x) + \frac{u''(x_j)}{2}(2\Delta x)^2 + \cdots) \\
=& \ (\alpha_1 + \alpha_2 + \alpha_3)u(x_j) + (\alpha_2 + 2\alpha_3)\Delta x u'(x_j) + (\frac{\alpha_2}{2} + 2\alpha_3)\Delta x^2 u''(x_j) + \cdots
\end{aligned}$$

因此可得

$$\begin{aligned}
\alpha_1 + \alpha_2 + \alpha_3 = 0 \\
(\alpha_2 + 2\alpha_3)\Delta x = 1 \\
(\frac{\alpha_2}{2} + 2\alpha_3)\Delta x^2 = 0
\end{aligned}$$

所以

$$\alpha_1 = -\frac{3}{2\Delta x}, \ \alpha_2 = \frac{2}{\Delta x}, \ \alpha_3 = -\frac{1}{2\Delta x}$$

Part 1.2

Fill in the tuple variable `alpha` of lenght 3 with your answer above. (Suppose $\Delta x = 1$)

In [3]:

(Top)

```python
alpha = [-1.5 , 2 , -0.5]
```

In [4]:

cell-e7c9469885bebc80 (Top)

```python
print('My alpha =', alpha)
### BEGIN HIDDEN TESTS
assert alpha == [-1.5, 2, -0.5] or alpha == (-1.5, 2, -0.5)
### END HIDDEN TESTS
```

```
My alpha = [-1.5, 2, -0.5]
```

Part 2.

Suppose we use the finite-difference formula above to approximate and assume the problem is periodic, i.e. take $U_0 = U_n$, $U_1 = U_{n+1}$, and so on.

Find the differentiation matrix D so that the numerical differentiation problem can be represented as a matrix-vector multiplication $W \triangleq DU$, where $D \in \mathbb{R}^{n \times n}$, $U \in \mathbb{R}^n$, and $W \in \mathbb{R}^n$.

Part 2.1

Complete the following function to construct the desired differentiation matrix under the **periodic boundary condition** with given number of partition n, coefficients of 3-point finite-difference formula α, and mesh size Δx.

In [5]:

(Top)

```python
def construct_differentiation_matrix(n, alpha, delta_x):
    ''' Construct
    Parameters
    ----------
    n : int
        number of partition
    alpha : tuple of length 3
       alpha = (α1, α2, α3)
    delta_x : float
        mesh size

    Returns
    -------
    D : scipy.sparse.diags
    '''
    diagonals = []
    diagonals.append([alpha[0] / delta_x for i in range(n)])
    diagonals.append([alpha[1] / delta_x for i in range(n - 1)])
    diagonals.append([alpha[1] / delta_x])
    diagonals.append([alpha[2] / delta_x for i in range(n - 2)])
    diagonals.append([alpha[2] / delta_x , alpha[2] / delta_x])
    offsets = [0 , 1 , 1 - n , 2 , 2 - n]
    D = diags(diagonals , offsets)
    return D
```

Part 2.2

Print and check your implementation.

In [6]:

cell-2ca00ba5ff115302 (Top)

```
print("For n = 8 and mesh size 1, D in dense form is")
sparse_D = construct_differentiation_matrix(8, alpha, 1)
dense_D = sparse_D.toarray()
print(dense_D)
### BEGIN HIDDEN TESTS
answer = np.array([
    [-1.5,  2.,  -0.5,  0.,   0.,   0.,   0.,   0. ],
    [ 0.,  -1.5,  2.,  -0.5,  0.,   0.,   0.,   0. ],
    [ 0.,   0.,  -1.5,  2.,  -0.5,  0.,   0.,   0. ],
    [ 0.,   0.,   0.,  -1.5,  2.,  -0.5,  0.,   0. ],
    [ 0.,   0.,   0.,   0.,  -1.5,  2.,  -0.5,  0. ],
    [ 0.,   0.,   0.,   0.,   0.,  -1.5,  2.,  -0.5],
    [-0.5,  0.,   0.,   0.,   0.,   0.,  -1.5,  2. ],
    [ 2.,  -0.5,  0.,   0.,   0.,   0.,   0.,  -1.5]
])
assert np.linalg.norm(dense_D - answer) < 1e-7
### END HIDDEN TESTS
```

```
For n = 8 and mesh size 1, D in dense form is
[[-1.5  2.  -0.5  0.   0.   0.   0.   0. ]
 [ 0.  -1.5  2.  -0.5  0.   0.   0.   0. ]
 [ 0.   0.  -1.5  2.  -0.5  0.   0.   0. ]
 [ 0.   0.   0.  -1.5  2.  -0.5  0.   0. ]
 [ 0.   0.   0.   0.  -1.5  2.  -0.5  0. ]
 [ 0.   0.   0.   0.   0.  -1.5  2.  -0.5]
 [-0.5  0.   0.   0.   0.   0.  -1.5  2. ]
 [ 2.  -0.5  0.   0.   0.   0.   0.  -1.5]]
```

Part 3.

Take $u(x) = e^{\sin x}$ on the domain $[-\pi, \pi]$. Find the finite difference approximation W for $\{u'(x_j)\}_{j=1}^{n}$ for various values of $n = 2^k$, $k = 3, 4, \ldots, 10$, and analyze the errors.

Part 3.1

Define the functinos u and $u'(x)$.

In [7]:

(Top)

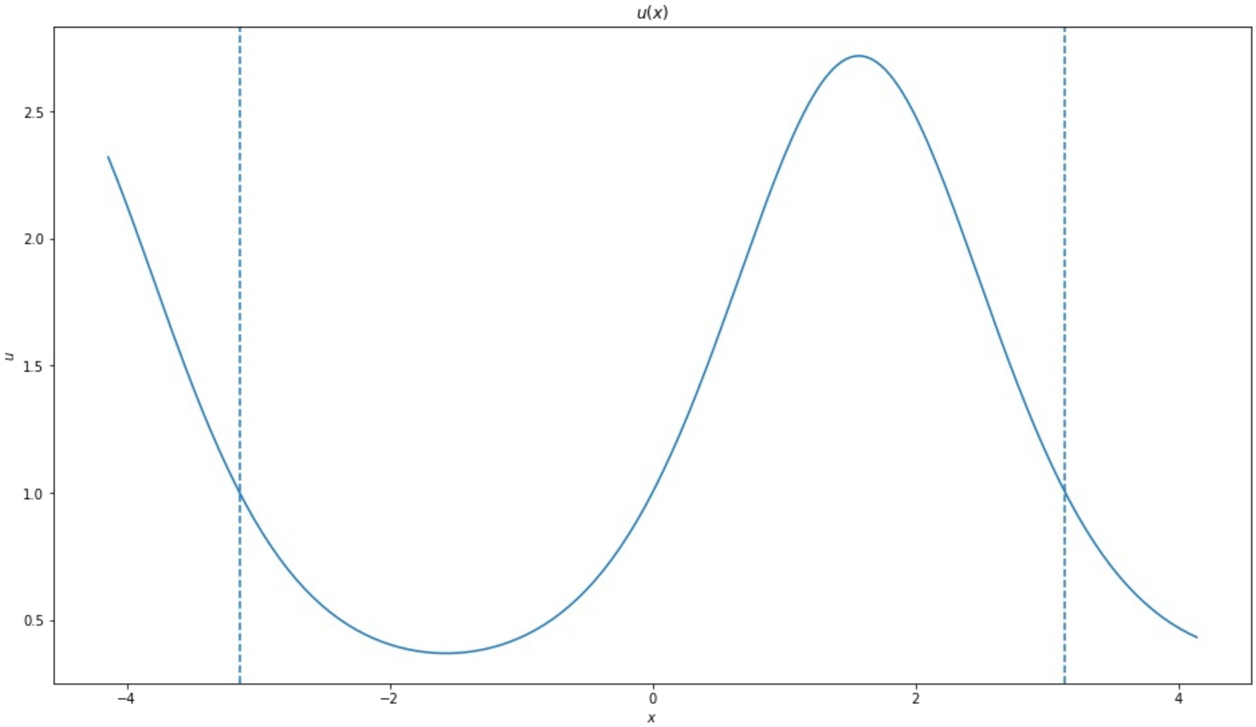
```
def u(x):
    return np.exp(np.sin(x))

def d_u(x):
    return np.cos(x) * np.exp(np.sin(x))
```

Plot and check the functions

In [8]:

cell-f97d6fb0842a6055 (Top)

```
x_range = np.linspace(-np.pi-1, np.pi+1, 2**8)
plt.figure(figsize=(16, 9))
plt.plot(x_range, u(x_range))
plt.axvline(x=np.pi, linestyle='--')
plt.axvline(x=-np.pi, linestyle='--')
plt.ylabel(r'$u$')
plt.xlabel(r'$x$')
plt.title(r'$u(x)$')
plt.show()
### BEGIN HIDDEN TESTS
assert u(1) == np.exp(np.sin(1))
assert u(3.14) == np.exp(np.sin(3.14))
assert d_u(1) == np.cos(1) * np.exp(np.sin(1))
assert d_u(0) == np.cos(0) * np.exp(np.sin(0))
### END HIDDEN TESTS
```



(Top)

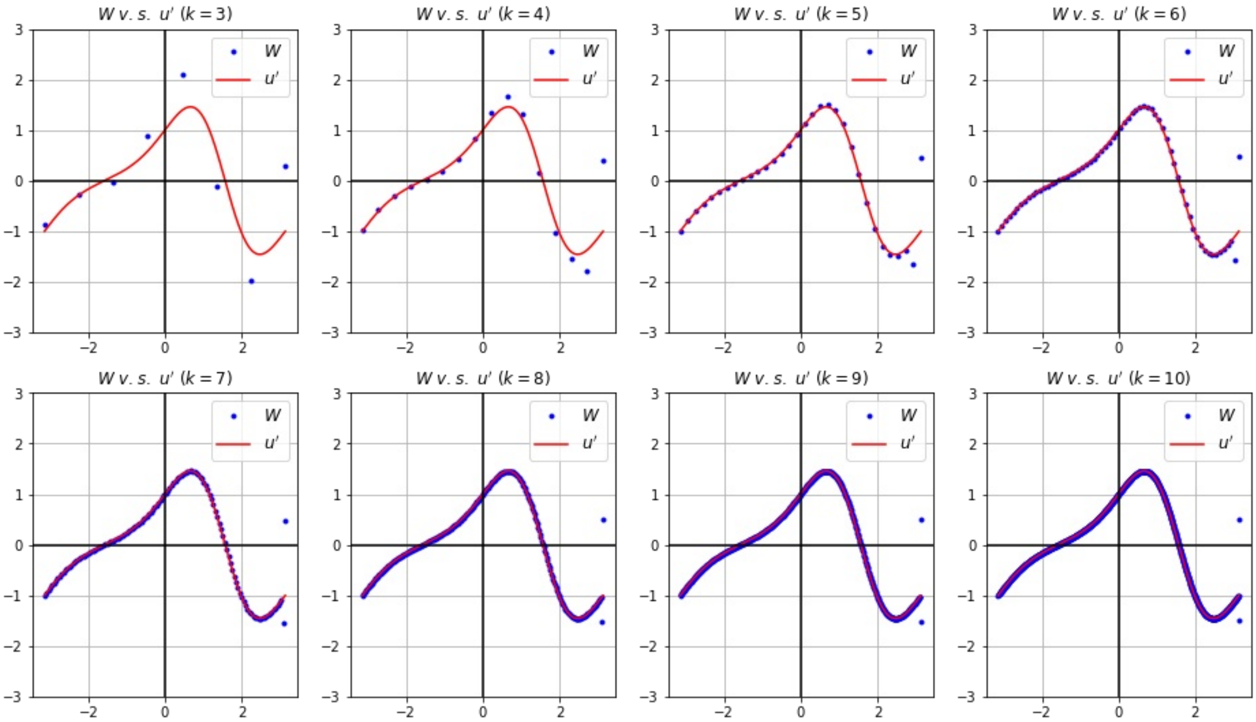
Part 3.2

Plot the u' and W together for each point $x_j, j \in \{1, 2, \ldots, n\}$ with $n = 2^k, k \in \{3, 4, \ldots, 10\}$. Note that there're total 8 figures to be plotted. And you need to compute the error, display them in the plots, and store them into the list variable `error_list` for further analysis below.

In [9]:

(Top)

```
error_list = []
fig, axes = plt.subplots(2, 4, figsize=(16,9))
for idx, ax in enumerate(axes.flatten()):
    k = idx + 3
    n = 2**k
    x = np.array([-np.pi + i * ((2 * np.pi) / (n - 1)) for i in range(n)]).reshape((-1 , 1))
    U = u(x)
    d_U = d_u(x)
    D = construct_differentiation_matrix(n , alpha , (2 * np.pi) / (n - 1))
    W = np.dot(D.toarray() , U)
    error = np.mean(np.abs(W - d_U))
    error_list.append(error)

    x_range = np.arange(-np.pi , np.pi , 0.01)
    ax.set_title(r'$W\ v.s.\ u^\prime\ (k = %d)$' % k)
    ax.plot(x , W , '.' , color = 'blue' , label = r'$W$')
    ax.plot(x_range , d_u(x_range) , '-' , color = 'red' , label = r'$u^\prime$')
    ax.legend(loc = 'upper right' , fontsize = 12)
    ax.set_ylim([-3, 3])
    ax.grid(True)
    ax.axhline(y = 0 , color = 'black')
    ax.axvline(x = 0 , color = 'black')

plt.show()
```

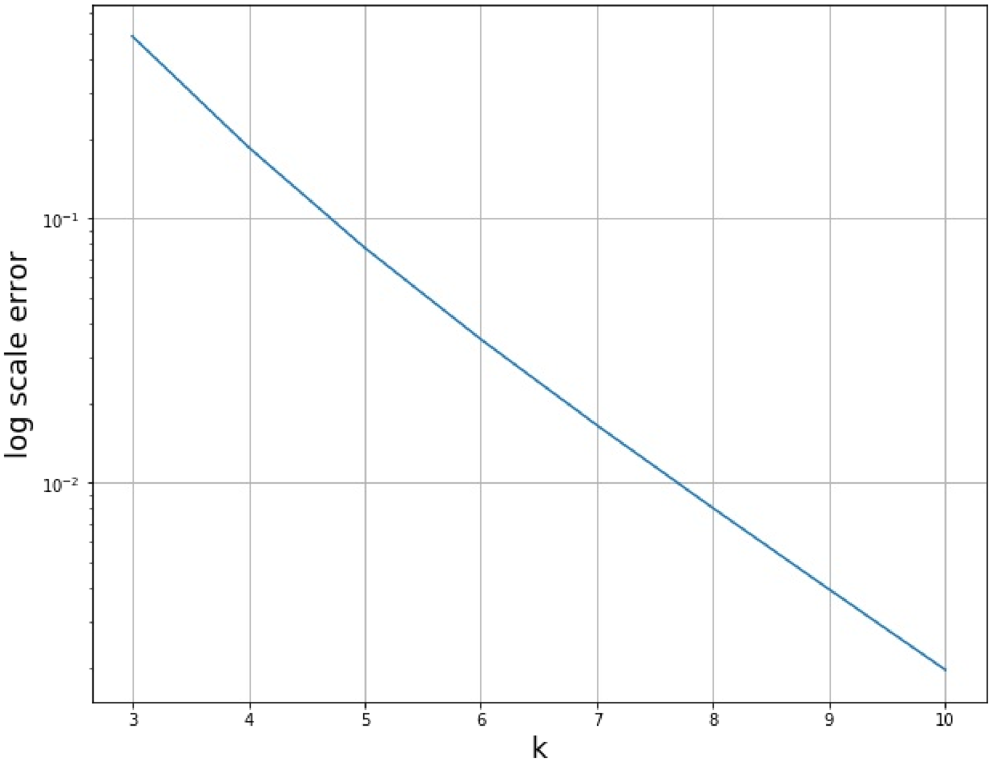


Plot the `error_list` with respect to $k = 3, 4, \ldots, 10$ in log scale to show the error behavior.

In [10]:

(Top)

```
plt.figure(figsize = (10 , 8))
plt.plot([k for k in range(3 , 11)] , error_list)
plt.xlabel('k' , fontsize = 18)
plt.ylabel('log scale error' , fontsize = 18)
plt.yscale('log')
plt.grid(True)
plt.show()
```



(Top)

Part 3.3

From the figure above, what rates of convergence do you observe as $\Delta x \to 0$?

由上圖可以看出error在取對數後大致會隨著k線性下降，因此推測error應該為隨著k指數下降。

In [ ]: