exercise2 (Score: 21.0 / 21.0)

1. Test cell (Score: 2.0 / 2.0)
2. Test cell (Score: 2.0 / 2.0)
3. Test cell (Score: 2.0 / 2.0)
4. Test cell (Score: 2.0 / 2.0)
5. Test cell (Score: 2.0 / 2.0)
6. Test cell (Score: 2.0 / 2.0)
7. Test cell (Score: 2.0 / 2.0)
8. Test cell (Score: 2.0 / 2.0)
9. Task (Score: 5.0 / 5.0)

Lab 5

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-5 (https://yuanyuyuan.github.io/itcm/lab-5.html)](https://yuanyuyuan.github.io/itcm/lab-5.html), 有任何問題歡迎找助教詢問。
4. **Deadline: 12/11(Wed.)**

In [1]:

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

Exercise 2

Suppose that a planet follows an elliptical orbit, which can be represented in a Cartesian coordinate system by the equation of the form

$$\alpha_1 y^2 + \alpha_2 xy + \alpha_3 x + \alpha_4 y + \alpha_5 = x^2. \qquad (1)$$

Based on the observation of the planet's position:

$$ \left [

```
  \begin{array}{c}
   x \\
   y
  \end{array}
\right ] =
\left [
  \begin{array}{cccccccccc}
```

1.02 & 0.95 & 0.87 & 0.77 & 0.67 & 0.56 & 0.44 & 0.30 & 0.16 & 0.01\ 0.39 & 0.32 & 0.27 & 0.22 & 0.18 & 0.15 & 0.13 & 0.12 & 0.13 & 0.15 \end{array} \right],$$

we want to determine the orbital parameters α_i, $i = 1, 2, \cdots, 5$, that solve the linear least squares problem of the form: $\min_{\alpha_i} \|b - A\alpha\|_2$, where the vector $b \in \mathrm{R}^{10}$, $\alpha = [\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5]^T \in \mathrm{R}^5$ and the matrix $A \in \mathrm{R}^{10\times 5}$ can be obtained easily when we substitute the aboe data to the equation (1).

Part 0

Import necessary libraries

In [2]:

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

Part 1

Find the solution of the problem by solving the associated normal equations via Cholesky factorization.

Part 1.1

Prepare data vector x, y and store them into 1D arrays: `data_x`, `data_y`.

In [3]:

(Top)

```
data_x = np.array([1.02 , 0.95 , 0.87 , 0.77 , 0.67 , 0.56 , 0.44 , 0.30 , 0.16 , 0.01])
data_y = np.array([0.39 , 0.32 , 0.27 , 0.22 , 0.18 , 0.15 , 0.13 , 0.12 , 0.13 , 0.15])
```

Check your `data_x` and `data_y`.

In [4]:

cell-3b704739d6fd2990 (Top)

```
print('x =', data_x)
print('y =', data_y)
### BEGIN HIDDEN TESTS
assert np.mean(data_x - np.array([1.02, 0.95, 0.87, 0.77, 0.67, 0.56, 0.44, 0.30, 0.16, 0.01])) < 1e-7
assert np.mean(data_y - np.array([0.39, 0.32, 0.27, 0.22, 0.18, 0.15, 0.13, 0.12, 0.13, 0.15])) < 1e-7
### END HIDDEN TESTS
```

```
x = [1.02 0.95 0.87 0.77 0.67 0.56 0.44 0.3  0.16 0.01]
y = [0.39 0.32 0.27 0.22 0.18 0.15 0.13 0.12 0.13 0.15]
```

Part 1.2

Construct the matrix A and the vector b with the data x, y and the equation (1).

In [5]:

(Top)

```
def construct_A_and_b(x, y):
    '''
    Arguments:
        x : 1D np.array, data x
        y : 1D np.array, data y

    Returns:
        A : 2D np.array
        b : 1D np.array
    '''
    A = np.column_stack((y**2 , x * y , x , y , np.ones(len(x))))
    b = x**2
    return (A , b)
```

Check your A and b.

In [6]:

cell-ab0180156b91fc0c (Top)

```
A, b = construct_A_and_b(data_x, data_y)
print('A:\n', A)
print('b:\n', b)
```

```
A:
 [[0.1521 0.3978 1.02   0.39   1.    ]
 [0.1024 0.304  0.95   0.32   1.    ]
 [0.0729 0.2349 0.87   0.27   1.    ]
 [0.0484 0.1694 0.77   0.22   1.    ]
 [0.0324 0.1206 0.67   0.18   1.    ]
 [0.0225 0.084  0.56   0.15   1.    ]
 [0.0169 0.0572 0.44   0.13   1.    ]
 [0.0144 0.036  0.3    0.12   1.    ]
 [0.0169 0.0208 0.16   0.13   1.    ]
 [0.0225 0.0015 0.01   0.15   1.    ]]
b:
 [1.0404e+00 9.0250e-01 7.5690e-01 5.9290e-01 4.4890e-01 3.1360e-01
 1.9360e-01 9.0000e-02 2.5600e-02 1.0000e-04]
```

Part 1.3

As the [lecture (https://ceiba.ntu.edu.tw/course/7a770d/content/cmath2019_note4_linear_system_cholesky.pdf)](https://ceiba.ntu.edu.tw/course/7a770d/content/cmath2019_note4_linear_system_cholesky.pdf) noted, to solve the noraml eqaution via Cholesky factorization we need additional **Forward substitution** and **Backward substituion** besides the **Cholesky factorization**. Please implement and check these three algorithms at below.

Algorithm 1: Implement forward substitution to solve

$$Lx = b,$$

where L is a lower triangular matrix and b is a column vector.

(Note that you need to implement it by hand, simply using some package functions is not allowed.)

In [7]:

(Top)

```
def forward_substitution(L, b):
    '''
    Arguments:
        L : 2D lower triangular np.array
        b : 1D np.array

    Return:
        x : solution to Lx = b
    '''
    n = L.shape[0]
    x = np.zeros(n)
    for i in range(n):
        s = sum([L[i][j] * x[j] for j in range(i)])
        x[i] = (b[i] - s) / L[i][i]
    return x
```

Check your function.

In [8]:

cell-55c3537517a849a7 (Top)

```
L = np.array([
    [1, 0, 0, 0],
    [2, 1, 0, 0],
    [4, 5, 6, 0],
    [1, 2, 3, 4]
])
x = np.array([11, 22, 33, 24])
print('L:\n', L)
print('x:\n', x)
print('My answer:\n', forward_substitution(L, L @ x))
```

```
L:
 [[1 0 0 0]
 [2 1 0 0]
 [4 5 6 0]
 [1 2 3 4]]
x:
 [11 22 33 24]
My answer:
 [11. 22. 33. 24.]
```

Algorithm 2: Implement backward substitution to solve

$$Rx = b,$$

where R is an upper triangular matrix and b is a column vector.

(Note that you need to implement it by hand, simply using some package functions is not allowed.)

In [9]:

(Top)

```
def backward_substitution(R, b):
    '''
    Arguments:
        R : 2D upper triangular np.array
        b : 1D np.array

    Return:
        x : solution to Rx = b
    '''
    n = R.shape[0]
    x = np.zeros(n)
    for i in range(n):
        s = sum([R[n - i - 1][j] * x[j] for j in range(n - i - 1 , n)])
        x[n - i - 1] = (b[n - i - 1] - s) / R[n - i - 1][n - i - 1]
    return x
```

Check your function.

In [10]:

cell-b139cd9ef4098615 (Top)

```
R = np.array([
    [1, 2, 3],
    [0, 4, 5],
    [0, 0, 9]
])
x = np.array([11, 22, 33])
print('R:\n', R)
print('x:\n', x)
print('My answer:\n', backward_substitution(R, R @ x))
```

```
R:
 [[1 2 3]
 [0 4 5]
 [0 0 9]]
x:
 [11 22 33]
My answer:
 [11. 22. 33.]
```

Algorithm 3: Implement Cholesky decompostion to decompose a nonsingualr [PSD (https://www.wikiwand.com/en/Definiteness_of_a_matrix)](https://www.wikiwand.com/en/Definiteness_of_a_matrix) matrix A into

$$A = R^T R,$$

where R is an upper triangular matrix.

(Note that you need to implement it by hand, simply using some package functions is not allowed.)

In [11]:

(Top)

```
def cholesky_decomposition(A):
    '''
    Arguments:
        A : 2D np.array

    Return:
        R : 2D np.array, A = R^T R
    '''
    n = A.shape[0]
    R = np.zeros((n , n))
    for i in range(n):
        s = sum([R[j][i]**2 for j in range(i)])
        R[i][i] = np.sqrt(A[i][i] - s)
        for j in range(i + 1 , n):
            s = sum([R[k][i] * R[k][j] for k in range(j)])
            R[i][j] = (A[i][j] - s) / R[i][i]
    return R
```

Check your function.

In [12]:

cell-cc45a402f856cb26 (Top)

```
# Construct a PSD matrix A
_A = np.array([
    [1, 3, 2, 4],
    [4, 2, 1, 7],
    [2, 5, 9, 0],
    [3, 5, 8, 2]
])
A = _A.T @ _A

# Do Cholesky decomposition
R = cholesky_decomposition(A)
print('A:\n', A)
print('R:\n', R)
print('A = R.T @ R:\n', R.T @ R)
```

```
A:
 [[ 30  36  48  38]
 [ 36  63  93  36]
 [ 48  93 150  31]
 [ 38  36  31  69]]
R:
 [[ 5.47722558  6.57267069  8.76356092  6.93781906]
 [ 0.          4.44971909  7.95555838 -2.15743956]
 [ 0.          0.          3.14787085 -4.01425733]
 [ 0.          0.          0.          0.31282475]]
A = R.T @ R:
 [[ 30.  36.  48.  38.]
 [ 36.  63.  93.  36.]
 [ 48.  93. 150.  31.]
 [ 38.  36.  31.  69.]]
```

Part 1.4

Implement the function `solve_alpha` to find α from the associated the normal equation.

In [13]:

(Top)

```
def solve_alpha(x, y):
    '''
    Arguments:
        x : 1D np.array, data x
        y : 1D np.array, data y

    Returns:
        alpha : 1D np.array

    Hints:
        1. Find matrix A, vector b
        2. Find the associated normal equation
        3. Do Cholesky decomposition
        4. Solve the equation with forward/backward substition
    '''

    (A , b) = construct_A_and_b(x , y)
    new_A = np.dot(np.transpose(A) , A)
    new_b = np.dot(np.transpose(A) , b)
    R = cholesky_decomposition(new_A)
    temp = forward_substitution(np.transpose(R) , new_b)
    alpha = backward_substitution(R , temp)
    return alpha
```

Solve α !

In [14]:

cell-ada65b7c60848c59 (Top)

```
alpha = solve_alpha(data_x, data_y)
print('alpha:\n', alpha)
### BEGIN HIDDEN TESTS
assert np.mean(alpha - np.array([-2.63562548,  0.14364618,  0.55144696,  3.22294034, -0.43289427])) < 1e-
7
### END HIDDEN TESTS
```

```
alpha:
 [-2.63562548  0.14364618  0.55144696  3.22294034 -0.43289427]
```

Part 2

Perturb the input data slightly by adding to each coordinate of each data point a uniformly distributed random number, and solve the least square problem as before with the perturbed data.

Compare the new values for the parameters with those previously computed. What effect does this difference have on the plot of the orbit ?

Part 2.1

In order to plot the orbit, we need to transform the equation (1) into a graph $z = f(x, y, \alpha)$ and then plot the contour at $z = 0$ by the tool `plt.contour` .

In [15]:

(Top)

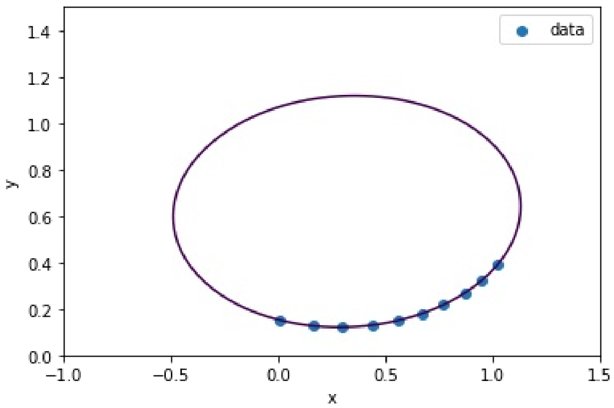
```
def ellipse(x, y, alpha):
    '''
    Arguments:
        x : 1D np.array, data x
        y : 1D np.array, data y
        alpha : 1D np.array, the coefficients

    Returns:
        z : 1D np.array, z=f(x, y, alpha) from equation (1)
    '''
    z = alpha[0] * (y**2) + alpha[1] * (x * y) + alpha[2] * x + alpha[3] * y + alpha[4] - x**2
    return z
```

Plot the orbit.

In [16]:

cell-c944b24065f4673f (Top)

```
# Plot the exact data points (x,y)
plt.scatter(data_x, data_y, label='data')

# Prepare mesh data points (X,Y) to plot the orbit
X, Y = np.meshgrid(
    np.linspace(-1, 1.5, 100),
    np.linspace(0, 1.5, 100)
)
# Plot the level curve at z = 0 only
plt.contour(X, Y, ellipse(X, Y, alpha), [0])

plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



Part 2.2

Now perturb the original data with some slight, uniformly random noise and follow the steps as before to find new `perturbed_x`, `perturbed_y`, `perturbed_alpha`.

In [17]:

(Top)

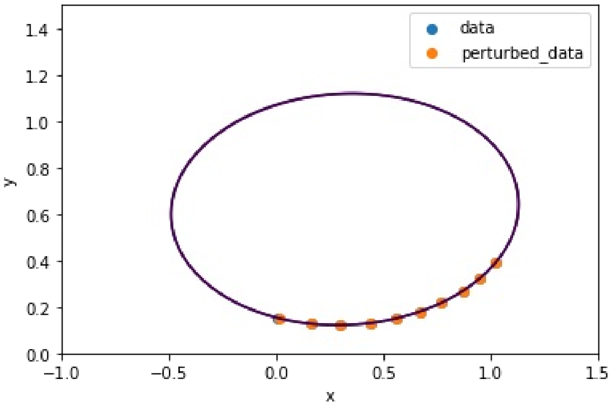
```
perturbed_x = data_x + 0.001 * np.random.rand(10)
perturbed_y = data_y + 0.001 * np.random.rand(10)
perturbed_alpha = solve_alpha(perturbed_x , perturbed_y)
```

Overlay the new perturbed orbit on the plot.

In [18]:

cell-7428d2eef3884195 (Top)

```
# Plot the exact data points (x,y)
plt.scatter(data_x, data_y, label='data')

# Plot the perturbed data points
plt.scatter(perturbed_x, perturbed_y, label='perturbed_data')

# Prepare mesh data points (X,Y) to plot the orbits
X, Y = np.meshgrid(
    np.linspace(-1, 1.5, 100),
    np.linspace(0, 1.5, 100)
)

# Plot the level curve at z = 0
plt.contour(X, Y, ellipse(X, Y, alpha), [0])

# Plot the level curve at z = 0 after perturbed
plt.contour(X, Y, ellipse(X, Y, perturbed_alpha), [0])

plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



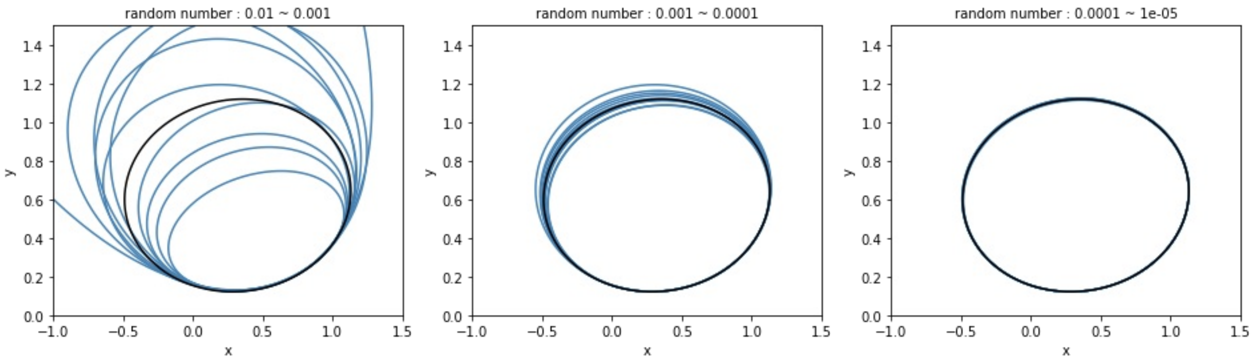
(Top)

Part 2.3

Try some different perturbations and compare the orbits before and after your perturbation. What's your observation?

In [19]:

```python
fig, axes=plt.subplots(1, 3, figsize=(16, 4))

# Prepare mesh data points (X,Y) to plot the orbits
X, Y = np.meshgrid(
    np.linspace(-1, 1.5, 100),
    np.linspace(0, 1.5, 100)
)

for i, ax in enumerate(axes.flatten()):
    (m , M) = (10**(-i - 2) , 10**(-i - 3))
    for j in range(10):
        perturbed_x = data_x + ((M - m) * np.random.rand(10) + m)
        perturbed_y = data_y + ((M - m) * np.random.rand(10) + m)
        perturbed_alpha = solve_alpha(perturbed_x , perturbed_y)

        # Plot the perturbed data points
        # ax.scatter(perturbed_x, perturbed_y, label='perturbed_data')

        # Plot the level curve at z = 0 after perturbed
        ax.contour(X, Y, ellipse(X, Y, perturbed_alpha), [0], colors='steelblue')

    # Plot the exact data points (x,y)
    # ax.scatter(data_x, data_y, label='data')

    # Plot the level curve at z = 0
    ax.contour(X, Y, ellipse(X, Y, alpha), [0], colors='black')

    ax.set_title('random number : {} ~ {}'.format(m , M) , fontsize = 10)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    # ax.legend()

plt.show()
```



上圖中，黑色曲線為使用exact data所得，而藍色曲線為使用perturbed data所得。
其中，perturbed data為在data point中加上random number而得。
最左邊的圖片中data point所加上的random number為介於0.01到0.001之間。
中間的圖片中data point所加上的random number為介於0.001到0.0001之間。
最右邊的圖片中data point所加上的random number為介於0.0001到0.00001之間。
而每個圖片中，皆分別在data point中加入了10次的random number，分別得到10組perturbed data，並以此分別得到了10條曲線。
由此可以看出，若data point中加入的random noise越大，則perturbed data所得到的曲線歧異越大，且與exact data所得到的曲線相差越多，其結果似乎還滿直觀的。

In [ ]: