ESE 417 Homework 1

Problem 1, Problem 3, Problem 5(c) are completed using python.
Problem 2, Problem 4, Problem 5(a)(b) are written by hand.

Problem 1

(Use Python numpy package to finish this)
Consider the following two real vectors: $\boldsymbol{x} = [-2.4, 1.5, 0.0, 3.2]^T$ *and* $\boldsymbol{y} = [1.3, 4.2, 3.0, -2.5]^T$
a) Find the 1-norm, 2-norm, 3-norm and infinity-norm of $\boldsymbol{x}$, then sort them in a descending order.
b) Find the distance between $\boldsymbol{x}$ *and* $\boldsymbol{y}$, $d(\boldsymbol{x},\boldsymbol{y})=\|\boldsymbol{x}-\boldsymbol{y}\|$, using 1-norm,2-norm, and infinity-norm, then sort them in an ascending order.

In [21]:
```
# a)

import numpy as np

# Define the two vectors
x = np.array([-2.4, 1.5, 0.0, 3.2])
y = np.array([1.3, 4.2, 3.0, -2.5])

print("x = ", x)
print("y = ", y)

# Calculate norms of x
norm_1_x = np.linalg.norm(x, ord=1)
norm_2_x = np.linalg.norm(x, ord=2)
norm_3_x = np.linalg.norm(x, ord=3)
norm_inf_x = np.linalg.norm(x, ord=np.inf)

print("Norms of x:")
print("L1 norm: ", norm_1_x)
print("L2 norm: ", norm_2_x)
print("L3 norm: ", norm_3_x)
print("L_inf norm: ", norm_inf_x)

# Sort norms of x in descending order
norms_x = np.array([norm_1_x, norm_2_x, norm_3_x, norm_inf_x])
norms_x_sorted = np.sort(norms_x)[::-1]
print("Sorted norms of x: ", norms_x_sorted)
```

```
x =  [-2.4  1.5  0.   3.2]
y =  [ 1.3  4.2  3.  -2.5]
Norms of x:
L1 norm:  7.1
L2 norm:  4.272001872658765
L3 norm:  3.683220833338153
L_inf norm:  3.2
Sorted norms of x:  [7.1        4.27200187 3.68322083 3.2       ]
```

In [22]:
```
# b)

# Cauculate the distance between x and y in L1, L2 and L_inf norms
dist_1 = np.linalg.norm(x - y, ord=1)
dist_2 = np.linalg.norm(x - y, ord=2)
dist_inf = np.linalg.norm(x - y, ord=np.inf)

print("Distance between x and y:")
print("L1 distance: ", dist_1)
print("L2 distance: ", dist_2)
print("L_inf distance: ", dist_inf)
```

```
# Sort distances in ascending order
dists = np.array([dist_1, dist_2, dist_inf])
dists_sorted = np.sort(dists)
print("Sorted distances: ", dists_sorted)
```

```
Distance between x and y:
L1 distance:  15.100000000000001
L2 distance:  7.903796556086196
L_inf distance:  5.7
Sorted distances:  [ 5.7         7.90379656 15.1       ]
```

Problem 3

(use Numpy package in this problem)

Consider the following matrix:

$$B = \begin{bmatrix} 2 & 2 & 4 \\ 1 & 4 & 7 \\ 2 & 5 & 4 \end{bmatrix}$$

Find the eigenvalues and corresponding eigenvectors of matrix $\boldsymbol{B}$ and then find the eigendecomposition of matrix $\boldsymbol{B}$. Verify that B=$\boldsymbol{Q\Lambda Q}$-1

In [23]:
```
import numpy as np

# Define the matrix B
B = np.array([[2, 2, 4],
              [1, 4, 7],
              [2, 5, 4]])

print("Matrix B:")
print(B)

# Calculate eigenvalues and eigenvectors of B
eigenvalues, eigenvectors = np.linalg.eig(B)

print("Eigenvalues of B:")
print(eigenvalues)
print("Eigenvectors of B:")
print(eigenvectors)
```

```
Matrix B:
[[2 2 4]
 [1 4 7]
 [2 5 4]]
Eigenvalues of B:
[10.93777843  1.25234419 -2.19012261]
Eigenvectors of B:
[[-0.42228933 -0.92515036 -0.29500861]
 [-0.67326244  0.37923364 -0.69446846]
 [-0.6069509  -0.01669331  0.65626479]]
```

In [24]:
```
# Find the eigndecomposition of B
Q = eigenvectors
Lambda = np.diag(eigenvalues)
Q_inv = np.linalg.inv(Q)

print("Eigendecomposition of B:")
print("Q:")
print(Q)
print("Lambda:")
print(Lambda)
print("Q_inv:")
print(Q_inv)
```

```
Eigendecomposition of B:
Q:
[[-0.42228933 -0.92515036 -0.29500861]
 [-0.67326244  0.37923364 -0.69446846]
 [-0.6069509  -0.01669331  0.65626479]]
Lambda:
[[10.93777843  0.          0.        ]
 [ 0.          1.25234419  0.        ]
 [ 0.          0.         -2.19012261]]
Q_inv:
[[-0.24458613 -0.63090205 -0.77757728]
 [-0.88991249  0.47022663  0.09756081]
 [-0.24884369 -0.57153292  0.80710925]]
```

In [25]:
```
# Verify the equation B = Q * Lambda * Q_inv
B_reconstructed = Q @ Lambda @ Q_inv

print("Reconstructed matrix B:")
print(B_reconstructed)

# Find B and B_reconstructed are equal
print("Are B and B_reconstructed equal? " + str(np.allclose(B, B_reconstructed)))
```

```
Reconstructed matrix B:
[[2. 2. 4.]
 [1. 4. 7.]
 [2. 5. 4.]]
Are B and B_reconstructed equal? True
```

Problem 5

Consider the following objective function: $f(x, y) = x^2 + 2y^2 - 2xy$

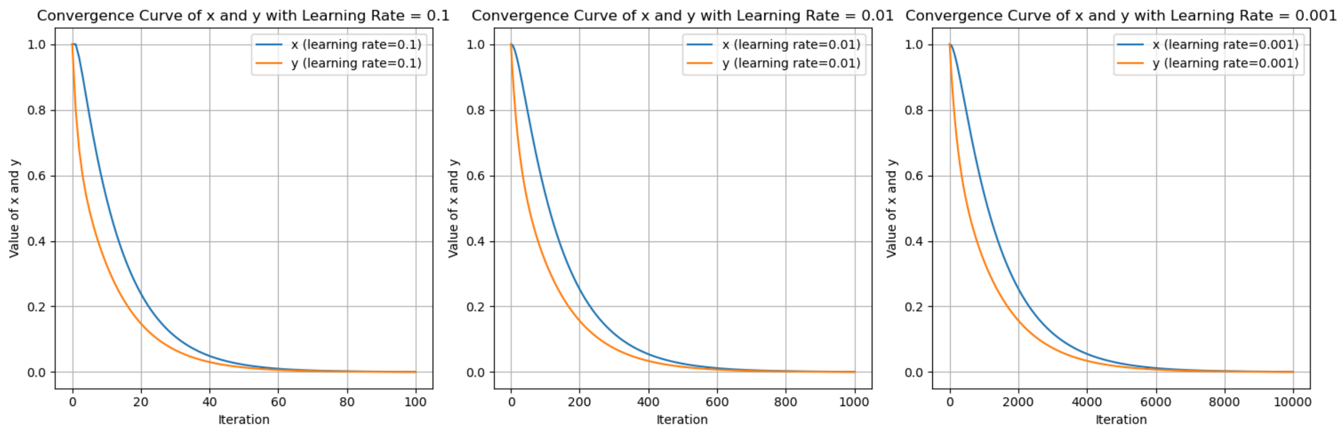
a) Compute the Hessian matrix H of function f, and find the definiteness of matrix H.

b) Use calculus method to find the minimum value of $f(x,y)$

c) Use Python to implement the gradient descent search method to find the global minimum value of function f, suppose the initial values of [x, y] are [1,1]. Show the convergence curve of x *and* y with learning rate: 0.1, 0.01, and 0.001. (Don't use sk-learn package)

In [26]:
```
import numpy as np
import matplotlib.pyplot as plt

# Function f(x, y)
def f(x, y):
    return x**2 + 2*y**2 - 2*x*y
# Gradient of f(x, y)
def grad_f(x, y):
    df_dx = 2*x - 2*y
    df_dy = 4*y - 2*x
    return np.array([df_dx, df_dy])

# Gradient descent implementation returning x and y values
def gradient_descent(initial_values, learning_rate, num_iterations):
    # Initialize lists to store x and y values
    x_vals = [initial_values[0]]
    y_vals = [initial_values[1]]
    for _ in range(num_iterations):
        # Calculate gradient at current point
        grad = grad_f(x_vals[-1], y_vals[-1])
        # Update x and y values
        new_x = x_vals[-1] - learning_rate * grad[0]
        new_y = y_vals[-1] - learning_rate * grad[1]
        # Append new x and y values to the lists
        x_vals.append(new_x)
        y_vals.append(new_y)
    return np.array(x_vals), np.array(y_vals)
```

```python
# Initial values for (x, y)
initial_values = [1, 1]

# Learning rates and number of iterations
learning_rates = [0.1, 0.01, 0.001]
num_iterations_list = [100, 1000, 10000]

# Plot convergence for different learning rates
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
for i, (lr, num_iterations) in enumerate(zip(learning_rates, num_iterations_list)):
    # Run gradient descent and store x and y values
    x_vals, y_vals = gradient_descent(initial_values, lr, num_iterations)
    axes[i].plot(x_vals, label=f'x (learning rate={lr})')
    axes[i].plot(y_vals, label=f'y (learning rate={lr})')
    axes[i].set_title(f'Convergence Curve of x and y with Learning Rate = {lr}')
    axes[i].set_xlabel('Iteration')
    axes[i].set_ylabel('Value of x and y')
    axes[i].legend(loc='upper right')
    axes[i].grid(True)

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



```
In [ ]:
```