# hw1

September 16, 2025

[1]:
```
# Run this if not already installed

# !pip install numpy matplotlib pandas
```

[2]:
```
# imports

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

Problem 1

[3]:
```
# Define the arrays

x = np.array([[-2.4], [1.5], [0.0], [3.2]])
y = np.array([[1.3], [4.3], [3.0], [-2.5]])

# a) Find the 1-norm, 2-norm, 3-norm and infinity-norm of  , then sort them in␣
 ↪a descending order

print("a) Find the 1-norm, 2-norm, 3-norm and infinity-norm of  , then sort␣
 ↪them in a descending order")

x_flat = x.flatten()
y_flat = y.flatten()

norm_1_x = np.linalg.norm(x_flat, ord=1)
norm_2_x = np.linalg.norm(x_flat, ord=2)
norm_3_x = np.linalg.norm(x_flat, ord=3)
norm_inf_x = np.linalg.norm(x_flat, ord=np.inf)

# print("1-norm of x =", norm_1_x)
# print("2-norm of x =", norm_2_x)
# print("3-norm of x =", norm_3_x)
# print("infinity-norm of x =", norm_inf_x)

norms = {
    "1-norm": norm_1_x,
```

```
    "2-norm": norm_2_x,
    "3-norm": norm_3_x,
    "Infinity-norm": norm_inf_x
}

# Sort descending
sorted_norms = dict(sorted(norms.items(), key=lambda item: item[1],␣
 ↪reverse=True))

for name, val in sorted_norms.items():
    print(f"{name}: {val:.4f}")

print("")

# b) Find the distance between x and y, d(x, y) = ||x-y||, using 1-norm,␣
 ↪2-norm, and infinity-norm, then sort them in an ascending order.

print("b) Find the distance between $x$ and $y$, $d(x, y) = ||x-y||$, using␣
 ↪1-norm, 2-norm, and infinity-norm, then sort them in an ascending order.")

# Difference
diff = x_flat - y_flat

# Norms (distances)
dist_1   = np.linalg.norm(diff, ord=1)
dist_2   = np.linalg.norm(diff, ord=2)
dist_inf = np.linalg.norm(diff, ord=np.inf)

# Collect results
distances = {
    "1-norm": dist_1,
    "2-norm": dist_2,
    "Infinity-norm": dist_inf
}

# Sort ascending
sorted_distances = dict(sorted(distances.items(), key=lambda item: item[1]))

# Print results
for name, val in sorted_distances.items():
    print(f"{name}: {val:.4f}")
```

```
a) Find the 1-norm, 2-norm, 3-norm and infinity-norm of  , then sort them in a
descending order
1-norm: 7.1000
2-norm: 4.2720
3-norm: 3.6832
```

```
Infinity-norm: 3.2000

b) Find the distance between      ,  ( , )=|| - ||, using 1-norm, 2-norm, and
infinity-norm, then sort them in an ascending order.
Infinity-norm: 5.7000
2-norm: 7.9385
1-norm: 15.2000
```

### 0.0.1 Problem 2

Consider the following matrix:

$$A = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

---

**(a) Find the eigenvalues and eigenvectors of $A$.**

$$|A - \lambda I| = \begin{vmatrix} 5-\lambda & 4 \\ 4 & 5-\lambda \end{vmatrix} = (5-\lambda)^2 - 4^2 = \lambda^2 - 10\lambda + 9 = (\lambda - 1)(\lambda - 9) = 0$$

$$\Rightarrow \lambda_1 = 1, \quad \lambda_2 = 9$$

Let the eigenvector of $\lambda_1 = 1$ be $v_1$. We will now solve for $v_1$

$$(A - \lambda_1 I)v_1 = \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix} v_1 = \vec{0}$$

Via row-reduction, we obtain:

$$\left[\begin{array}{cc|c} 4 & 4 & 0 \\ 4 & 4 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 0 & 0 \end{array}\right] \Rightarrow v_1 = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Let the eigenvector of $\lambda_2 = 9$ be $v_2$. We will now solve for $v_2$

$$(A - \lambda_2 I)v_2 = \begin{bmatrix} -4 & 4 \\ 4 & -4 \end{bmatrix} v_2 = \vec{0}$$

Via row reduction, we obtain

$$\left[\begin{array}{cc|c} -4 & 4 & 0 \\ 4 & -4 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{cc|c} -1 & 1 & 0 \\ 0 & 0 & 0 \end{array}\right] \Rightarrow v_2 = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{where}$$

Thus,

$$\boxed{\lambda_1 = 1,\ v_1 = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 \\ -1 \end{bmatrix}; \quad \lambda_2 = 9,\ v_2 = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix}}$$

---

**(b) Find the eigendecomposition $A = Q\Lambda Q^T$**

Choosing orthonormal eigenvectors:

$$v_1 = \frac{\sqrt{2}}{2}\begin{bmatrix}1\\-1\end{bmatrix}, \quad v_2 = \frac{\sqrt{2}}{2}\begin{bmatrix}1\\1\end{bmatrix}$$

So,

$$Q = \frac{\sqrt{2}}{2}\begin{bmatrix}1 & 1\\-1 & 1\end{bmatrix}, \quad \Lambda = \begin{bmatrix}1 & 0\\0 & 9\end{bmatrix}$$

Verifying our solution,

$$A = Q\Lambda Q^T = \frac{\sqrt{2}}{2}\begin{bmatrix}1 & 1\\-1 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\0 & 9\end{bmatrix}\frac{\sqrt{2}}{2}\begin{bmatrix}1 & -1\\1 & 1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1 & 9\\-1 & 9\end{bmatrix}\begin{bmatrix}1 & -1\\1 & 1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}10 & 8\\8 & 10\end{bmatrix} = \begin{bmatrix}5 & 4\\4 & 5\end{bmatrix}$$

---

**(c) What is the definiteness of the matrix?**

$A$ is positive definite as $\lambda_1, \lambda_2 > 0$.

Problem 3

[4]:
```
# Define B
B = np.array([[2, 2, 4],
              [1, 4, 7],
              [2, 5, 4]], dtype=float)

# Eigen-decomposition: B v =  v
# w: eigenvalues, V: eigenvectors as columns
w, V = np.linalg.eig(B)

# Form Λ (Lambda) and Q
Lambda = np.diag(w)
Q = V

# Reconstruct B from Q Λ Q^{-1}
B_recon = Q @ Lambda @ np.linalg.inv(Q)

# Verification
tol = 1e-10
is_equal = np.allclose(B, B_recon, atol=tol, rtol=0)

print("Eigenvalues ( ):")
print(w)
print("\nEigenvectors (columns of Q):")
print(Q)
```

```
print("\nΛ (diagonal of eigenvalues):")
print(Lambda)
print("\nReconstruction Q Λ Q^{-1}:")
print(B_recon)
print("\nMax abs reconstruction error:", np.max(np.abs(B - B_recon)))
print(f"\nDoes Q Λ Q^(-1) == B (within tol={tol})? {is_equal}")
```

```
Eigenvalues ( ):
[10.93777843  1.25234419 -2.19012261]

Eigenvectors (columns of Q):
[[-0.42228933 -0.92515036 -0.29500861]
 [-0.67326244  0.37923364 -0.69446846]
 [-0.6069509  -0.01669331  0.65626479]]

Λ (diagonal of eigenvalues):
[[10.93777843  0.          0.        ]
 [ 0.          1.25234419  0.        ]
 [ 0.          0.         -2.19012261]]

Reconstruction Q Λ Q^{-1}:
[[2. 2. 4.]
 [1. 4. 7.]
 [2. 5. 4.]]

Max abs reconstruction error: 3.552713678800501e-15

Does Q Λ Q^(-1) == B (within tol=1e-10)? True
```

### 0.0.2 Problem 4

*Use the definition of a convex function to show that affine functions $f(x) = a^T x + b$ are convex, where $a, x \in \mathcal{R}^n$ and $b \in \mathcal{R}$.*

Let $x_1, x_2 \in \mathcal{R}^n$. Recall that a function $f : \mathcal{R}^n \to \mathcal{R}$ is convex if

$$f(tx_1 + (1-t)x_2) \leq t\, f(x_1) + (1-t)\, f(x_2) \quad \text{for all } x_1, x_2 \text{ and } t \in [0,1].$$

Looking at $f$,

$$f(tx_1 + (1-t)x_2) = a^T(tx_1 + x_2 - tx_2) + b \geq t\, f(x_1) + (1-t)\, f(x_2) = t(a^T x_1 + b)$$

$$a^T(tx_1 + x_2 - tx_2) \geq t(a^T x_1 + b) + (1-t)(a^T x_2 + b)$$

$$ta^T x_1 + a^T x_2 - ta^T x_2 \geq ta^T x_1 + tb + a^T x_2 + b - ta^T x_2 - tb$$

$$0 \geq 0$$

The inequality holds, and so affine functions $f(x) = a^T x + b$ are convex.

### 0.0.3 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**

First, we will compute the Hessian matrix of $f$, $H$:

$$f \Rightarrow \nabla f = \begin{bmatrix} 2x - 2y \\ 4y - 2x \end{bmatrix} \Rightarrow H = \begin{bmatrix} 2 & -2 \\ -2 & 4 \end{bmatrix}$$

Now we will determine the definiteness of $H$ by finding its eigenvalues:

$$\left|(H - \lambda I)\right| = \left|\begin{bmatrix} 2-\lambda & -2 \\ -2 & 4-\lambda \end{bmatrix}\right| = (2-\lambda)(4-\lambda) - (-2)^2 = \lambda^2 - 6\lambda + 4 = 0$$

$$\lambda = \frac{6 \pm \sqrt{(-6)^2 - 4(1)(4)}}{2(1)} = 3 \pm \sqrt{5} > 0$$

Thus, $H$ is positive-definite

---

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

$$\nabla f = \vec{0} \Rightarrow (x, y) = (0, 0)$$

$$f_{xx} = 2 \quad \text{and} \quad |H| = 8 - (-2)^2 = 4 \Rightarrow (0, 0) \text{ is the global minimum of } f$$

---

**(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 $x$ $y$ with learning rate: 0.1, 0.01, and 0.001. (Don't use sk-learn package)**

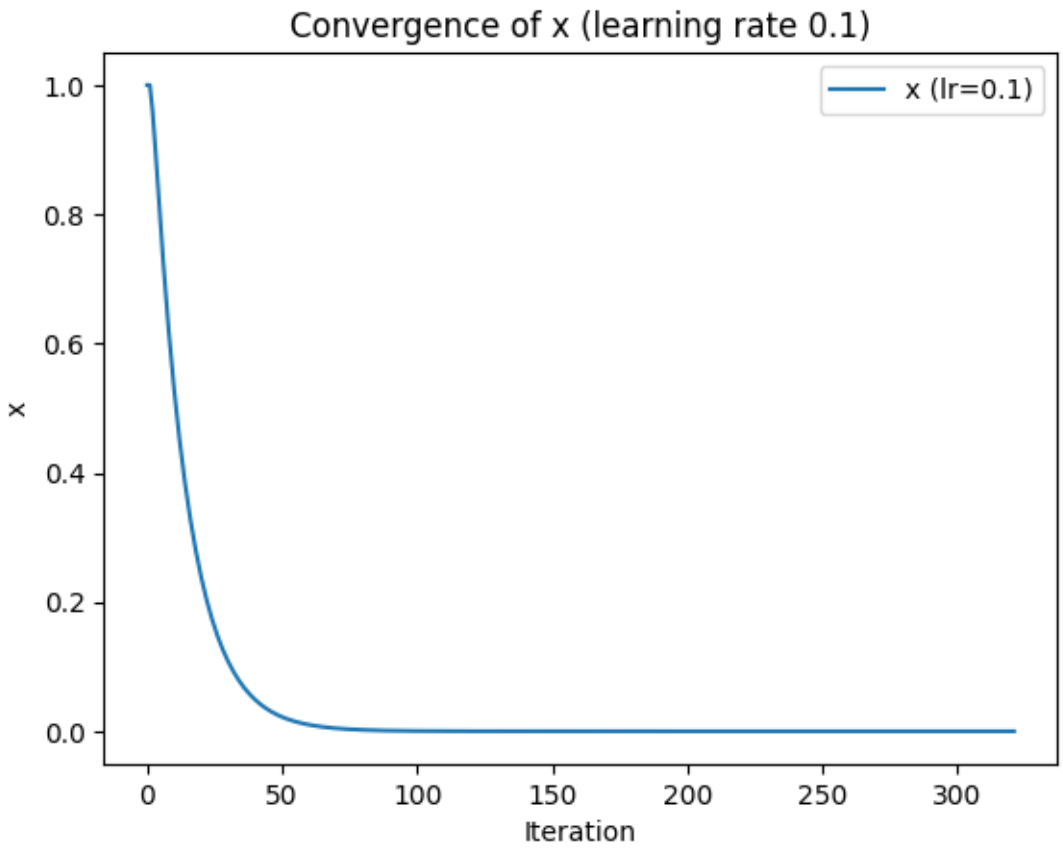
[5]:
```
# Defining f and \nabla f

def f(x, y):
    return x**2 + 2*y**2 - 2*x*y

def grad(x, y):
    #  f = [2x - 2y, 4y - 2x]
    return np.array([2*x - 2*y, 4*y - 2*x], dtype=float)
```

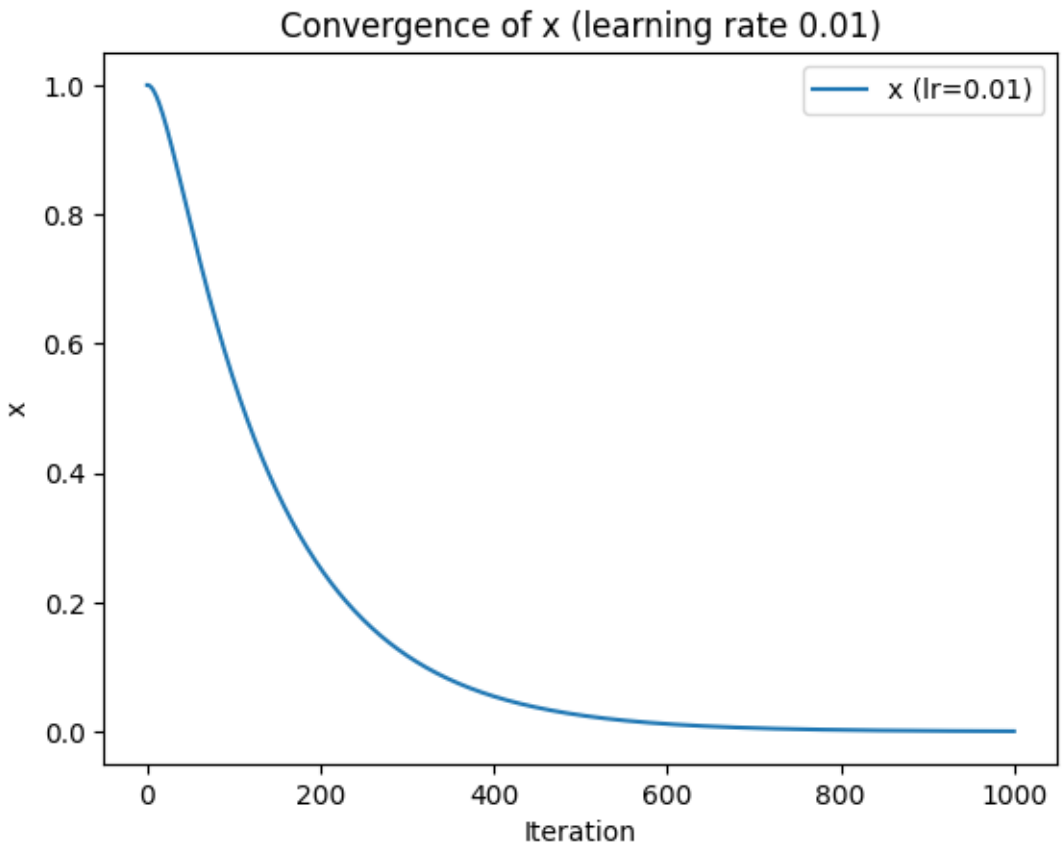
[6]:
```
# Define gradient descent function

def gradient_descent(x0, y0, lr, max_iters=2000, tol=1e-10):
    x, y = float(x0), float(y0)
```

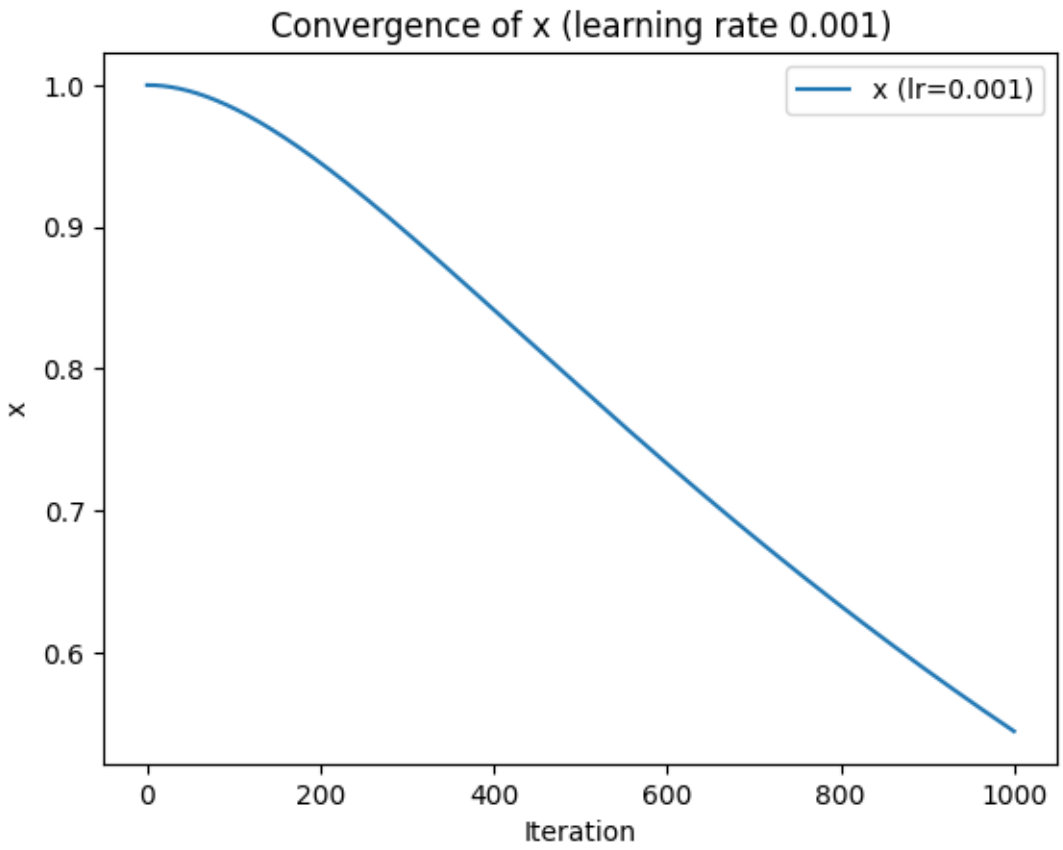
```
    xs, ys, fs = [x], [y], [f(x, y)]
    for k in range(max_iters):
        g = grad(x, y)
        x_new = x - lr * g[0]
        y_new = y - lr * g[1]
        xs.append(x_new)
        ys.append(y_new)
        fs.append(f(x_new, y_new))
        if np.linalg.norm([x_new - x, y_new - y]) < tol:
            x, y = x_new, y_new
            break
        x, y = x_new, y_new
    return np.array(xs), np.array(ys), np.array(fs)
```

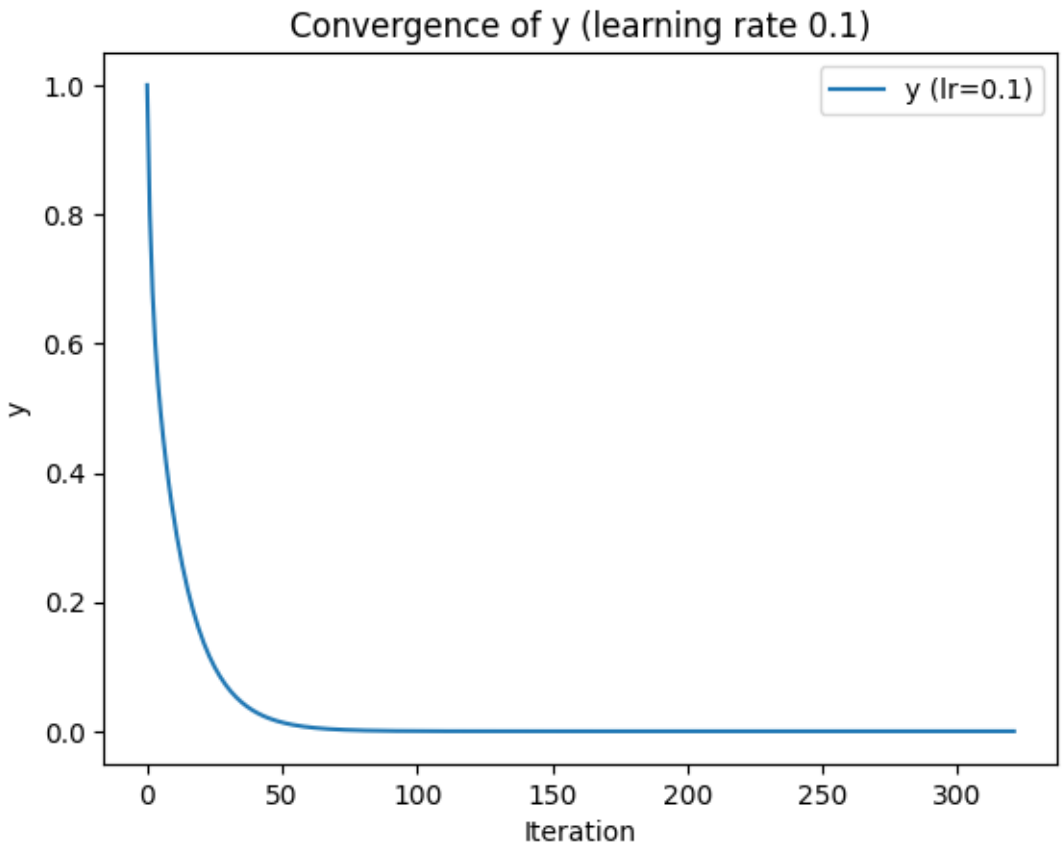
[7]:
```
# Run with different learning rates

lrs = [0.1, 0.01, 0.001]
results = {}

for lr in lrs:
    xs, ys, fs = gradient_descent(1.0, 1.0, lr, max_iters=1000, tol=1e-12)
    results[lr] = {"x": xs, "y": ys, "f": fs}
```

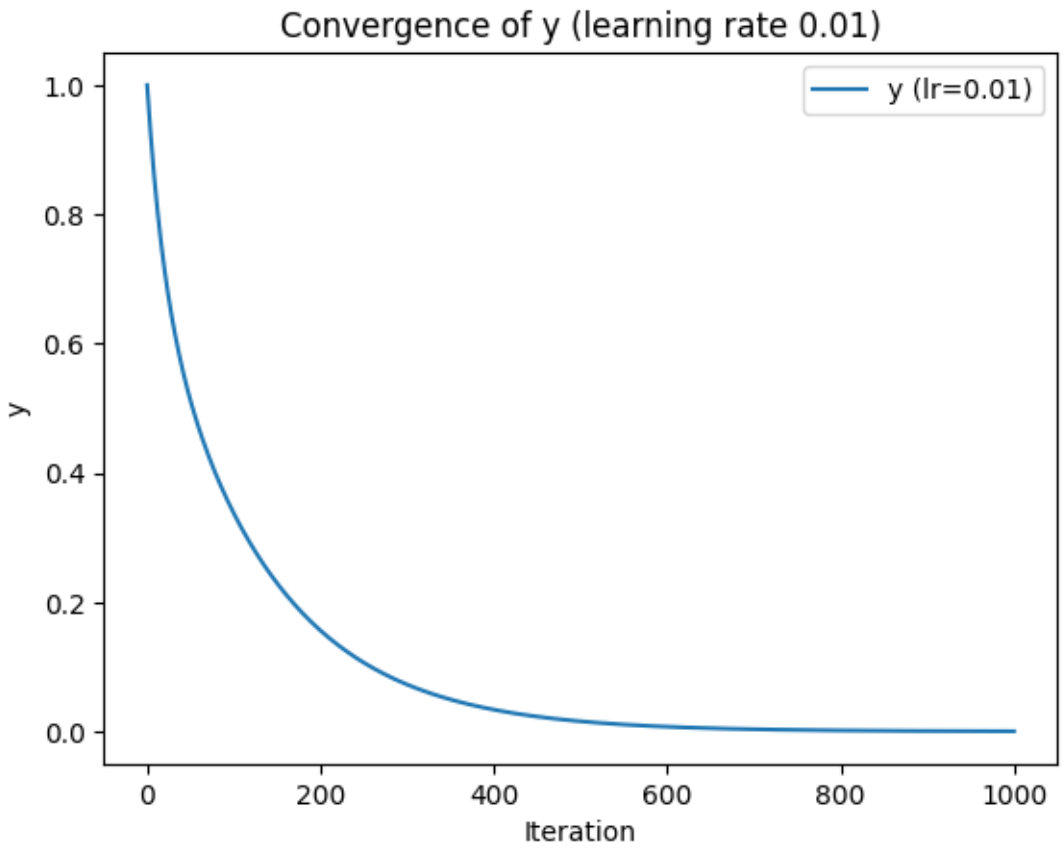
[8]:
```
# Plotting convergence

# Convergence of x
for lr in lrs:
    xs = results[lr]["x"]
    plt.figure()
    plt.plot(range(len(xs)), xs, label=f"x (lr={lr})")
    plt.xlabel("Iteration")
    plt.ylabel("x")
    plt.title(f"Convergence of x (learning rate {lr})")
    plt.legend()
    plt.show()

# Convergence of y

for lr in lrs:
    ys = results[lr]["y"]
    plt.figure()
    plt.plot(range(len(ys)), ys, label=f"y (lr={lr})")
    plt.xlabel("Iteration")
    plt.ylabel("y")
    plt.title(f"Convergence of y (learning rate {lr})")
    plt.legend()
    plt.show()
```

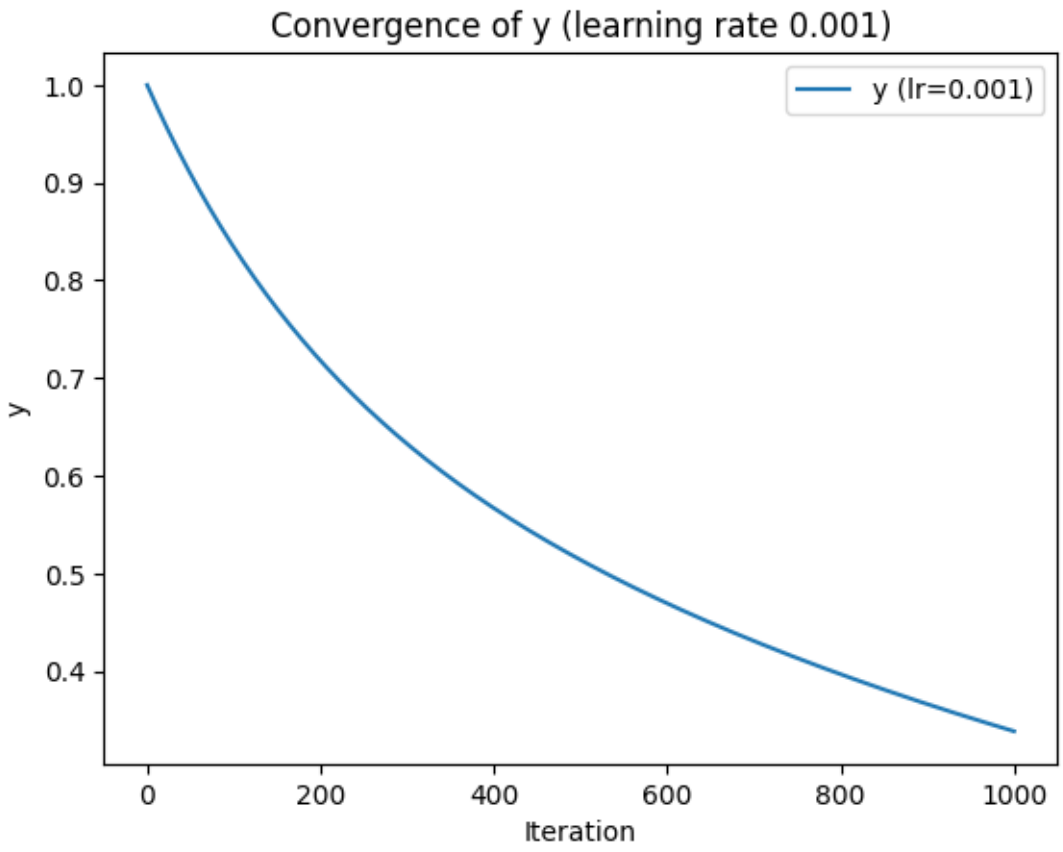

Convergence of x (learning rate 0.1)
x (lr=0.1)
1.0
0.8
0.6
0.4
0.2
0.0
x
0
50
100
150
200
250
300
Iteration


Convergence of x (learning rate 0.01)
x (lr=0.01)
1.0
0.8
0.6
0.4
0.2
0.0
x
0
200
400
600
800
1000
Iteration


Convergence of x (learning rate 0.001)
x (lr=0.001)
1.0
0.9
0.8
0.7
0.6
x
0
200
400
600
800
1000
Iteration


Convergence of y (learning rate 0.1)
y (lr=0.1)
1.0
0.8
0.6
0.4
0.2
0.0
y
0
50
100
150
200
250
300
Iteration


Convergence of y (learning rate 0.01)
y (lr=0.01)
1.0
0.8
0.6
0.4
0.2
0.0
y
0
200
400
600
800
1000
Iteration




[9]:
```
# Summarizing results

summary_rows = []
for lr in lrs:
    xs, ys, fs = results[lr]["x"], results[lr]["y"], results[lr]["f"]
    summary_rows.append({
        "learning_rate": lr,
        "iterations": len(xs)-1,
        "x_final": xs[-1],
        "y_final": ys[-1],
        "f_final": fs[-1]
    })

summary_df = pd.DataFrame(summary_rows)
summary_df
```

[9]:

| | learning_rate | iterations | x_final | y_final | f_final |
|---|---|---|---|---|---|
| 0 | 0.100 | 321 | 9.769133e-12 | 6.037656e-12 | 5.037721e-23 |
| 1 | 0.010 | 1000 | 5.470641e-04 | 3.381042e-04 | 1.579787e-07 |
| 2 | 0.001 | 1000 | 5.443489e-01 | 3.384308e-01 | 1.569377e-01 |