```
# Define the updated functions and Jacobian
def f1(x, y):
    return 1/((2-x)*(2-y)) - x

def f2(x, y):
    return 2/((3-2*x)*(3-y)) - y

def F(p):
    x, y = p
    return np.array([f1(x, y), f2(x, y)])

def J(p):
    x, y = p
    # Calculated from the Jacobian provided
    j11 = -1/((2-x)**2 * (2-y)) - 1
    j12 = -1/((2-x) * (2-y)**2)
    j21 = -4/((3-2*x)**2 * (3-y))
    j22 = -2/((3-2*x) * (3-y)**2) - 1
    return np.array([[j11, j12], [j21, j22]])

# Newton's method for a system of equations in two dimensions
def newton_method_2d(f, J, initial_guess, tolerance=1e-10, max_iterations=100):
    p = np.array(initial_guess)

    for _ in range(max_iterations):
        F_val = f(p)
        J_val = J(p)

        # Check if the determinant of the Jacobian is zero
        if np.linalg.det(J_val) == 0:
            raise ValueError("Jacobian determinant is zero")

        # Solve for the change
        delta = np.linalg.solve(J_val, -F_val)

        # Update the guess
        p = p + delta

        # Check for convergence
        if np.linalg.norm(delta) < tolerance:
            return p
    raise ValueError("Did not converge after maximum iterations")

initial_guess = (0.01, 0.01)
root = newton_method_2d(F, J, initial_guess)
# after a while...
> root = [0.36675873, 0.33056621]
```