```python
import numpy as np

DEBUG = False

def debug_print(*args):
    if DEBUG:
        print(*args)


def printRotationMatrix(R):
    print("[")
    for row in R:
        print("    ", np.round(row, 3))
    print("]")


def checkValidRotMatrix(R):
    # Check if the given matrix is a rotation matrix
    # Check if the determinant is 1
    det = np.linalg.det(R)
    if np.abs(det - 1) > 1e-6:
        debug_print("Determinant is not 1: ", det)
        return False

    # Check if the inverse is equal to the transpose
    R_inv = np.linalg.inv(R)
    R_T = np.transpose(R)
    if not np.allclose(R_inv, R_T):
        debug_print("Inverse is not equal to transpose")
        return False

    return True


"""
Given
    → x^2 + y^2 = c,
    → ax + by = d
Find x, y
"""
def solve_quadratic_and_linear(a, b, c, d):
    # x = (d - by) / a
    # → (d - by)^2 / a^2 + y^2 = c
    # → (d^2 - 2bdy + b^2y^2) / a^2 + y^2 = c
    # → d^2 - 2bdy + b^2y^2 + a^2y^2 = c*a^2
    # → (b^2 + a^2)y^2 - 2bdy + d^2 - c*a^2 = 0
    # → y = (bd +- sqrt(b^2d^2 - (b^2 + a^2)(d^2 - c*a^2))) / (b^2 + a^2)
    y = (
```

```
        (b*d + np.sqrt(b**2 * d**2 - (b**2 + a**2)*(d**2 - c*a**2))) / (b**2 + a**2),
        (b*d - np.sqrt(b**2 * d**2 - (b**2 + a**2)*(d**2 - c*a**2))) / (b**2 + a**2)
    )

    x = (
        (d - b*y[0]) / a,
        (d - b*y[1]) / a
    )

    return x, y


def RotationMatrixComplete(a12 = 0.892, a13 = 0.423, a31 = -0.186):
    # Given three entries of the rotation matrix
    # Find possible rotation matrices
    # R = [
    #     [a11,    0.892, 0.423],
    #     [a21,    a22  , a23  ],
    #     [-0.186, a32  , a33  ]
    # ]

    # 1. First find possible values of a11, using row 1 being
    #    unit norm vector.
    a11 = (np.sqrt(1 - a12**2 - a13**2), -np.sqrt(1 - a12**2 - a13**2))

    # 2. Next find possible values of a21, using column 1 being
    #    unit norm vector.
    a21 = (np.sqrt(1 - a31**2 - a11[0]**2), -np.sqrt(1 - a31**2 - a11[0]**2))

    a11_21_combined = [(a11[0], a21[0]), (a11[0], a21[1]), (a11[1], a21[0]), (a11[1], a21[1])]

    # 3. Use norm of row 2 = 1 to create one equation
    # → a22^2 + a23^2 = 1 - a21^2
    # Use orthogonality of row 2 and row 1 to create another equation
    # → a22*a12 + a23*a13 = -a21*a11
    # We can solve these two equations to find possible values of a22, a23
    a11_21_22_23_combined = []
    for a11_21 in a11_21_combined:
        a22, a23 = solve_quadratic_and_linear(a12, a13, 1 - a11_21[1]**2, -a11_21[1]*a11_21[0])
        a11_21_22_23_combined.append((a11_21[0], a11_21[1], a22[0], a23[0]))
        a11_21_22_23_combined.append((a11_21[0], a11_21[1], a22[1], a23[1]))

    # 4. Use norm of col2 and col3 = 1 to find a32, a33
    # → a12^2 + a22^2 + a32^2 = 1
    all_combined = []
    for a11_21_22_23 in a11_21_22_23_combined:
        a32 = np.sqrt(1 - a12**2 - a11_21_22_23[2]**2)
```

```
96          a33 = np.sqrt(1 - a13**2 - a11_21_22_23[3]**2)
97          all_combined.append((a11_21_22_23[0], a11_21_22_23[1], a11_21_22_23[2],
    a11_21_22_23[3], a32, a33))
98          all_combined.append((a11_21_22_23[0], a11_21_22_23[1], a11_21_22_23[2],
    a11_21_22_23[3], -a32, a33))
99          all_combined.append((a11_21_22_23[0], a11_21_22_23[1], a11_21_22_23[2],
    a11_21_22_23[3], a32, -a33))
100         all_combined.append((a11_21_22_23[0], a11_21_22_23[1], a11_21_22_23[2],
    a11_21_22_23[3], -a32, -a33))
101
102     # Compute all possible rotation matrices, print only those which are valid
103     print("Possible Rotation Matrices: ")
104     for a11_21_22_23_32_33 in all_combined:
105         a11, a21, a22, a23, a32, a33 = a11_21_22_23_32_33
106         R= np.array([
107             [a11, a12, a13],
108             [a21, a22, a23],
109             [a31, a32, a33]
110         ])
111         if checkValidRotMatrix(R):
112             printRotationMatrix(R)
113             print("")
114
115
116 if __name__ == "__main__":
117     RotationMatrixComplete()
118
119
```