Lab 2 Solution key

1.

```
import numpy as np
def check_linear_independence(vectors):
    # Stack vectors as columns
    A = np.column_stack(vectors)
    num_vectors = len(vectors)
    rank = np.linalg.matrix_rank(A)

    if rank == num_vectors:
        return True, "The vectors are linearly independent."
    else:
        return False, "The vectors are linearly dependent."


# Example Usage
v1 = np.array([1, 2, 3])
v2 = np.array([0, 1, 4])
v3 = np.array([5, 6, 0])

is_independent, message = check_linear_independence([v1, v2, v3])
print(message)
```

2.

```
import sympy as sp
def find_spanning_set(vectors):
    # Create SymPy Matrix with vectors as columns
    A = sp.Matrix(vectors).T
    # Compute Row Reduced Echelon Form (RREF)
    rref_matrix, pivot_cols = A.rref()

    # The spanning set (basis of column space) consists of pivot columns of original matrix
    spanning_set = [A.col(idx) for idx in pivot_cols]
    return spanning_set


# Example Usage
vectors = [[1, 2, 3], [2, 4, 6], [0, 1, 1]]
span = find_spanning_set(vectors)

print("Spanning set vectors (linearly independent basis):")
for v in span:
    sp.pprint(v)
```

3.

```
import sympy as sp

# 1. Input vectors and construct matrix with vectors as columns
vectors = [[1, 2, 1], [-2, -4, -2], [0, 3, 1], [1, 5, 2]]
A = sp.Matrix(vectors).T

print("Constructed Matrix A:")
sp.pprint(A)

# 2. Perform Row Reduced Echelon Form (RREF) and find pivot columns
rref_A, pivot_cols = A.rref()

print("\nRREF Matrix:")
sp.pprint(rref_A)
print("\nPivot Columns (0-indexed):", pivot_cols)

# 3. Find a basis for the column space
basis = [A.col(idx) for idx in pivot_cols]

print("\nBasis for the vector space:")
for b in basis:
    sp.pprint(b)
```

4.

```
import sympy as sp
def solve_linear_system():
    num_eqs = int(input("Enter number of equations: "))
    num_vars = int(input("Enter number of variables: "))

    # Define variable symbols (x1, x2, ...)
    vars = sp.symbols(f"x1:{num_vars + 1}")

    print("\nEnter coefficients and RHS for each equation (space-separated):")
    A_rows = []
    b_vals = []

    for i in range(num_eqs):
        row = list(
            map(
                float,
                input(
                    f"Eq {i+1} ({num_vars} coeffs + 1 RHS): "
                ).split(),
```

```
            )
        )
        A_rows.append(row[:-1])
        b_vals.append(row[-1])

    A = sp.Matrix(A_rows)
    b = sp.Matrix(b_vals)
    Augmented = A.row_insert(A.shape[0], sp.Matrix([[0] * num_vars]))  # Temp
    Augmented = A.col_insert(A.shape[1], b)

    rank_A = A.rank()
    rank_Aug = Augmented.rank()

    print("\n--- Solution Result ---")
    if rank_A < rank_Aug:
        print("Inconsistent system: No solution exists.")
    else:
        sol = sp.linsolve((A, b), vars)
        if rank_A == num_vars:
            print("Unique Solution:")
            print(sol)
        else:
            print("Parametric Solution (Infinitely many solutions):")
            print(sol)
solve_linear_system()
```