Systems of Linear Equations

Welcome to the first assignment of the Course 1. From the lecture videos you learned about the systems of linear equations and the approach to find their solutions using row reduction.

After this assignment you will be able to:

- Use `NumPy` package to set up the arrays corresponding to the system of linear equations
- Evaluate the determinant of a matrix and find the solution of the system with `NumPy` linear algebra package
- Perform row reduction to bring matrix into row echelon form
- Find the solution for the system of linear equations using row reduced matrix

Table of Contents

Packages

Run the following cell to load the packages you'll need.

```
import numpy as np
```

Load the unit tests defined specifically for this notebook.

```
import w2_unittest
```

1 - System of Linear Equations and Corresponding `NumPy` Arrays

Matrices can be used to solve systems of equations. But first, you need to represent the system using matrices. Given the following system of linear equations:

$$\left\{\begin{array}{c} 2x_1 - x_2 + x_3 + x_4 = 6, \\ x_1 + 2x_2 - x_3 - x_4 = 3, \\ -x_1 + 2x_2 + 2x_3 + 2x_4 = 14, \\ x_1 - x_2 + 2x_3 + x_4 = 8, \end{array}\right. \quad (1)$$

you will construct matrix A, where each row represents one equation in the system and each column represents a variable x_1, x_2, x_3, x_4. The free coefficients from the right sides of the equations you will put into vector b.

Exercise 1

Construct matrix A and vector b corresponding to the system of linear equations (1).

```
### START CODE HERE ###
A = np.array([
        [2,-1,1,1],
        [1,2,-1,-1],
        [-1,2,2,2],
        [1,-1,2,1]
    ], dtype=np.dtype(float))
b = np.array([6,3,14,8], dtype=np.dtype(float))

### END CODE HERE ###

# Test your solution
w2_unittest.test_matrix(A, b)
```

```
 All tests passed
```

2 - Solution for the System of Equations with `NumPy` Linear Algebra Package

A system of four linear equations with four unknown variables has a unique solution if and only if the determinant of the corresponding matrix of coefficients is not equal to zero. `NumPy` provides quick and reliable ways to calculate the determinant of a square matrix and also to solve the system of linear equations.

Exercise 2

Find the determinant d of matrix A and the solution vector x for the system of linear equations (1).

```
### START CODE HERE ###
# determinant of matrix A
d = np.linalg.det(A)

# solution of the system of linear equations
# with the corresponding coefficients matrix A and free coefficients b
x = np.linalg.solve(A,b)
### END CODE HERE ###

print(f"Determinant of matrix A: {d:.2f}")

print(f"Solution vector: {x}")
```

```
Determinant of matrix A: -17.00
Solution vector: [2. 3. 4. 1.]
```

Expected Output

```
Determinant of matrix A: -17.00
Solution vector: [2. 3. 4. 1.]
```

```
# Test your solution
w2_unittest.test_det_and_solution_scipy(d,x)
```

```
 All tests passed
```

3 - Elementary Operations and Row Reduction

Even though the contemporary packages allow to find the solution with one line of the code, performing required algebraic operations manually helps to build foundations which are necessary for deep understanding of the machine learning algorithms.

Here you will solve the system of linear equations algebraically using row reduction. It involves combination of the equations using elementary operations, eliminaring as many variables as possible for each equation. There are three valid operations which can be performed to bring the system of equations to equivalent one (with the same solutions):

- Multiply any row by non-zero number
- Add two rows and exchange one of the original rows with the result of the addition
- Swap rows

Exercise 3

Set up three functions corresponding to the discussed above elementary operations.

```
### START CODE HERE ###
def MultiplyRow(M, row_num, row_num_multiple):
    # .copy() function is required here to keep the original matrix
without any changes
    M_new = M.copy()
    # exchange row_num of the matrix M_new with its multiple by
row_num_multiple
    # Note: for simplicity, you can drop check if  row_num_multiple
has non-zero value, which makes the operation valid
    M_new[row_num] = M_new[row_num] * row_num_multiple
    return M_new

def AddRows(M, row_num_1, row_num_2, row_num_1_multiple):
    M_new = M.copy()
    # multiply row_num_1 by row_num_1_multiple and add it to the
row_num_2,
    # exchanging row_num_2 of the matrix M_new with the result
    M_new[row_num_2] += M_new[row_num_1] * row_num_1_multiple
    return M_new

def SwapRows(M, row_num_1, row_num_2):
    M_new = M.copy()
    # exchange row_num_1 and row_num_2 of the matrix M_new
    M_new[row_num_1] = M_new[row_num_1] + M_new[row_num_2]
    M_new[row_num_2] = M_new[row_num_1] - M_new[row_num_2]
    M_new[row_num_1] = M_new[row_num_1] - M_new[row_num_2]
    return M_new
### END CODE HERE ###
```

Check your code using the following test cell s:

```
A_test = np.array([
        [1, -2, 3, -4],
        [-5, 6, -7, 8],
        [-4, 3, -2, 1],
        [8, -7, 6, -5]
    ], dtype=np.dtype(float))
print("Original matrix:")
print(A_test)

print("\nOriginal matrix after its third row is multiplied by -2:")
print(MultiplyRow(A_test,2,-2))

print("\nOriginal matrix after exchange of the third row with the sum
of itself and first row multiplied by 4:")
print(AddRows(A_test,0,2,4))

print("\nOriginal matrix after exchange of its first and third rows:")
```

```
print(SwapRows(A_test,0,2))


Original matrix:
[[ 1. -2.  3. -4.]
 [-5.  6. -7.  8.]
 [-4.  3. -2.  1.]
 [ 8. -7.  6. -5.]]

Original matrix after its third row is multiplied by -2:
[[ 1. -2.  3. -4.]
 [-5.  6. -7.  8.]
 [ 8. -6.  4. -2.]
 [ 8. -7.  6. -5.]]

Original matrix after exchange of the third row with the sum of itself
and first row multiplied by 4:
[[  1.  -2.   3.  -4.]
 [ -5.   6.  -7.   8.]
 [  0.  -5.  10. -15.]
 [  8.  -7.   6.  -5.]]

Original matrix after exchange of its first and third rows:
[[-4.  3. -2.  1.]
 [-5.  6. -7.  8.]
 [ 1. -2.  3. -4.]
 [ 8. -7.  6. -5.]]
```

Expected Output

```
Original matrix:
[[ 1 -2  3 -4]
 [-5  6 -7  8]
 [-4  3 -2  1]
 [ 8 -7  6 -5]]

Original matrix after its third row is multiplied by -2:
[[ 1 -2  3 -4]
 [-5  6 -7  8]
 [ 8 -6  4 -2]
 [ 8 -7  6 -5]]

Original matrix after exchange of the third row with the sum of itself
and first row multiplied by 4:
[[  1  -2   3  -4]
 [ -5   6  -7   8]
 [  0  -5  10 -15]
 [  8  -7   6  -5]]

Original matrix after exchange of its first and third rows:
[[-4  3 -2  1]
```

```
 [-5  6 -7  8]
 [ 1 -2  3 -4]
 [ 8 -7  6 -5]]

# Test your solution
w2_unittest.test_elementary_operations(MultiplyRow, AddRows, SwapRows)
```

```
 All tests passed
```

Exercise 4

Apply elementary operations to the defined above matrix A, performing row reduction according to the given instructions.

Note: Feel free to add a return statement between the different matrix operations in the code to check on your results while you are writing the code (commenting off the rest of the function). This way you can see, whether your matrix operations are performed correctly line by line (don't forget to remove the return statement afterwards!).

```
def augmented_to_ref(A, b):
    ### START CODE HERE ###
    # stack horizontally matrix A and vector b, which needs to be
reshaped as a vector (4, 1)
    A_system = A_system = np.hstack((A, b.reshape(4, 1)))

    # swap row 0 and row 1 of matrix A_system (remember that indexing
in NumPy array starts from 0)
    A_ref = A_ref = SwapRows(A_system, 0, 1)

    # multiply row 0 of the new matrix A_ref by -2 and add it to the
row 1
    A_ref = AddRows(A_ref, 0, 1, -2)

    # add row 0 of the new matrix A_ref to the row 2, replacing row 2
    A_ref[2] += A_ref[0]

    # multiply row 0 of the new matrix A_ref by -1 and add it to the
row 3
    A_ref = AddRows(A_ref, 0, 3, -1)

    # add row 2 of the new matrix A_ref to the row 3, replacing row 3
    A_ref[3] += A_ref[2]

    # swap row 1 and 3 of the new matrix A_ref
    A_ref = SwapRows(A_ref, 1, 3)

    # add row 2 of the new matrix A_ref to the row 3, replacing row 3
    A_ref[3] += A_ref[2]
```

```
    # multiply row 1 of the new matrix A_ref by -4 and add it to the
row 2
    A_ref = AddRows(A_ref, 1, 2, -4)

    # add row 1 of the new matrix A_ref to the row 3, replacing row 3
    A_ref[3] += A_ref[1]

    # multiply row 3 of the new matrix A_ref by 2 and add it to the
row 2
    A_ref = AddRows(A_ref, 3, 2, 2)

    # multiply row 2 of the new matrix A_ref by -8 and add it to the
row 3
    A_ref = AddRows(A_ref, 2, 3, -8)

    # multiply row 3 of the new matrix A_ref by -1/17
    A_ref[3] = A_ref[3] * -1/17
    ### END CODE HERE ###

    return A_ref

A_ref = augmented_to_ref(A, b)

print(A_ref)
```

```
[[ 1.  2. -1. -1.  3.]
 [ 0.  1.  4.  3. 22.]
 [ 0.  0.  1.  3.  7.]
 [-0. -0. -0.  1.  1.]]
```

Expected Output

```
[[ 1  2 -1 -1  3]
 [ 0  1  4  3 22]
 [ 0  0  1  3  7]
 [ 0  0  0  1  1]]

# Test your solution
w2_unittest.test_augmented_to_ref(augmented_to_ref)

 All tests passed
```

4 - Solution for the System of Equations using Row Reduction

The solution can be found from the reduced form manually. From the last line you can find x_4, then you can calculate each of the x_3, x_2 and x_1 taking the elements of the matrix `A_ref[i,j]` and solving the linear equations one by one.

Exercise 5

From the last line of the reduced matrix `A_ref` find x_4. Then you can calculate each of the x_3, x_2 and x_1 taking the elements of the matrix `A_ref[i,j]` and solving the linear equations one by one.

```
### START CODE HERE ###

# find the value of x_4 from the last line of the reduced matrix A_ref
x_4 = A_ref[3, 3]

# find the value of x_3 from the previous row of the matrix. Use value
of x_4.
x_3 = A_ref[2, 4] -  A_ref[2, 3]* x_4

# find the value of x_2 from the second row of the matrix. Use values
of x_3 and x_4
x_2 = A_ref[1, 4] - A_ref[1, 2]*x_3 - A_ref[2, 3]*x_4

# find the value of x_1 from the first row of the matrix. Use values
of x_2, x_3 and x_4
x_1 = A_ref[0, 4] - A_ref[0, 1]*x_2 + x_3 + x_4
### END CODE HERE ###

print(x_1, x_2, x_3, x_4)
```

```
2.0 3.0 4.0 1.0
```

Expected Output

```
2 3 4 1

# Test your solution
w2_unittest.test_solution_elimination(x_1, x_2, x_3, x_4)
```

```
 All tests passed
```

Exercise 6

Using the same elementary operations as above you can reduce the matrix further to diagonal form, from which you can see the solutions easily.

```
def ref_to_diagonal(A_ref):
    ### START CODE HERE ###
    # multiply row 3 of the matrix A_ref by -3 and add it to the row 2
    A_diag = AddRows(A_ref, 3, 2, -3)

    # multiply row 3 of the new matrix A_diag by -3 and add it to the
row 1
    A_diag = AddRows(A_diag, 3, 1, -3)

    # add row 3 of the new matrix A_diag to the row 0, replacing row 0
    A_diag[0] += A_diag[3]

    # multiply row 2 of the new matrix A_diag by -4 and add it to the
row 1
    A_diag = AddRows(A_diag, 2, 1, -4)

    # add row 2 of the new matrix A_diag to the row 0, replacing row 0
    A_diag[0] += A_diag[2]

    # multiply row 1 of the new matrix A_diag by -2 and add it to the
row 0
    A_diag = AddRows(A_diag, 1, 0, -2)
    ### END CODE HERE ###

    return A_diag

A_diag = ref_to_diagonal(A_ref)

print(A_diag)
```

```
[[ 1.  0.  0.  0.  2.]
 [ 0.  1.  0.  0.  3.]
 [ 0.  0.  1.  0.  4.]
 [-0. -0. -0.  1.  1.]]
```

Expected Output

```
[[1 0 0 0 2]
 [0 1 0 0 3]
 [0 0 1 0 4]
 [0 0 0 1 1]]

# Test your solution
w2_unittest.test_ref_to_diagonal(ref_to_diagonal)
```

```
 All tests passed
```

Congratulations! You have finished first assignment in this Course.