

Assignment 1

Due date: Wednesday, February 8 at 11:55 pm

Learning Outcomes

In this assignment, you will get practice with:

- Creating classes and their methods
- Arrays and 2D arrays
- Working with objects that interact with one another
- Conditionals and loops
- Implementing linear algebra operations
- Creating a subclass and using inheritance
- Programming according to specifications

Introduction

Linear algebra is an important subject in mathematics, but also in computer science and a variety of other areas. One of the fundamental structures in linear algebra is the matrix: a 2-dimensional table of numbers organized in rows and columns.

$$\begin{bmatrix} 8 & 14 & 2 \\ 5 & 3 & 12 \\ 1 & 7 & 9 \\ 15 & 6 & 4 \end{bmatrix}$$

Figure 1. A matrix with four rows and three columns.

Matrices are useful in many different areas to represent data in a table-like structure. A benefit of using matrices to store the data is that they have several standard operations for making computations on the data. For example, multiplying two matrices together is an important operation in many different applications including computer graphics, solving linear equations, modelling of data (money, population, etc.), and much more.

Another structure in linear algebra is called a **vector**. Vectors are special types of matrices that have only 1 row or only 1 column, meaning they are 1-dimensional rather than 2-dimensional. So vectors are similar to arrays and matrices are similar to arrays of arrays, or 2-dimensional arrays.

Matrices and vectors are also used in structures called Markov chains to make predictions about the future based on historic patterns. This approach requires that there are a finite number of independent "states" of which one or more must be active at all times, and that there are known probabilities of transitioning from one state to itself or to another state. For example, suppose you have exactly 3 possible states in your life: sleeping, eating, and working. You can transition from sleeping to sleeping, sleeping to eating, sleeping to working, eating to sleeping, and so on. You can never be stateless nor in a state other than the specified ones for this

Assignment 1

CS 1027 Computer Science Fundamentals II

model. If this is confusing, do not worry. This assignment is not really about Markov chains, but we give you some background information so you can see how the code you will work on could be used in applications.

The probabilities of the state-to-state transitions are based on historic data and they are stored in a matrix called a **transition matrix**. The states are stored in a vector called the **state vector**. The state vector might store the value 1 in one of its entries and the remaining entries could have value 0, but it also could contain different values in all its entries as long as these values add up to 1 (i.e. [0.3, 0.6, 0.1]).

Given the initial state of a system modelled with a Markov chain, we can predict what the state will be after one time unit (i.e. day, or month, or year depending on the system being modelled) by multiplying the transition matrix by the state vector. We can predict what the state will be after two time units by multiplying the transition matrix by itself and then multiplying that resulting matrix by the state vector. We can predict what the state will be after n time units by raising the transition matrix to the n^{th} power (multiplying it by itself n times) and then multiplying the resulting matrix by the state vector, thus allowing us to predict future states simply with the state vector and transition matrix.

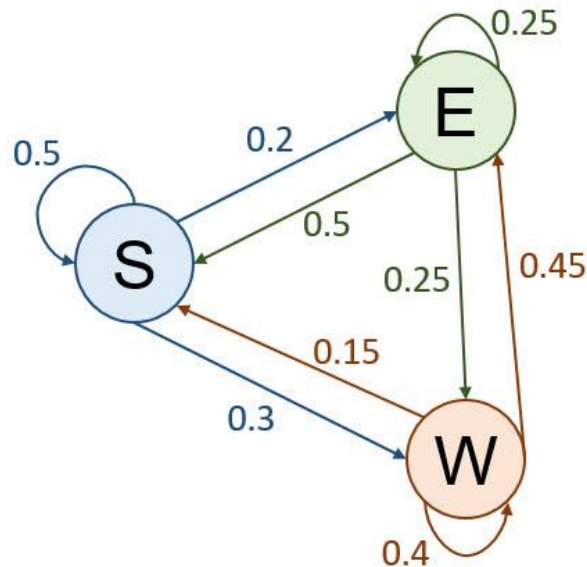


Figure 2. An example of a Markov chain with 3 possible states, S (sleeping), E (eating), and W (working), and the probabilities of transitioning between states.

$$\begin{bmatrix} S \text{ to } S & S \text{ to } E & S \text{ to } W \\ E \text{ to } S & E \text{ to } E & E \text{ to } W \\ W \text{ to } S & W \text{ to } E & W \text{ to } W \end{bmatrix} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.5 & 0.25 & 0.25 \\ 0.15 & 0.45 & 0.4 \end{bmatrix}$$

Figure 3. The transition matrix representing the probabilities shown in the Markov chain diagram from Figure 2.

Assignment 1

CS 1027 Computer Science Fundamentals II

Suppose you are currently working. The state vector would be:

$$[\text{Sleeping} \quad \text{Eating} \quad \text{Working}] = [0 \quad 0 \quad 1]$$

The first two entries of the state vector are zero, because you are not sleeping (so the probability that you are sleeping is zero) and you are not eating.

We can predict what your next state will be after one unit of time, i.e. in an hour, by multiplying the state vector by the transition matrix. This operation produces a vector representing the probability of being in each state. Notice in Figure 4 that the resulting state vector is the same as the bottom row of the transition matrix – this is because the state vector had 0's in the first two entries and a 1 in the last entry. In state vectors that have different values in all their entries (not just one 1 and some 0's), the resulting vector will not necessarily be identical to a row from the transition matrix.

$$[0 \quad 0 \quad 1] \times \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.5 & 0.25 & 0.25 \\ 0.15 & 0.45 & 0.4 \end{bmatrix} = [0.15 \quad 0.45 \quad 0.4]$$

Figure 4. The calculation of probabilities of your states after one time unit.

We can also predict what your state will be in two units of time, i.e. in two hours, by multiplying the transition matrix by itself and then multiplying the state vector by that produced matrix:

$$[0 \quad 0 \quad 1] \times \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.5 & 0.25 & 0.25 \\ 0.15 & 0.45 & 0.4 \end{bmatrix} \times \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.5 & 0.25 & 0.25 \\ 0.15 & 0.45 & 0.4 \end{bmatrix} = [0.36 \quad 0.323 \quad 0.318]$$

Figure 5. The calculation of probabilities of your states after two time units.

Classes to Implement

For this assignment, you must implement three (3) Java classes: **Matrix**, **Vector**, and **MarkovChain**. Follow the guidelines for each one below.

In these classes, you may implement more private (helper) methods if you want. However, you may not implement more public methods **except** `public static void main(String[] args)` for testing purposes (this is allowed and encouraged). You may not add instance variables other than the ones specified in these instructions nor change the variable types or accessibility (i.e. making a variable `public` when it should be `private`). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

Assignment 1

Matrix.java

This class is used to represent a matrix containing double (i.e. decimal) numbers.

The class must have exactly (no more and no less) the following private instance variables:

- private int numRows
- private int numCols
- private double[][] data

The class must have the following public methods:

- public Matrix (int r, int c): [constructor](#)
 - Initialize the instance variables numRows and numCols and initialize data to have r rows and c columns
- public Matrix (int r, int c, double[] linArr): [constructor](#)
 - Same as above, but now you have to populate the 2-dimensional array data with the elements in linArr
 - Note that linArr is a 1-dimensional array whereas data is 2-dimensional, so you have to map the items from linArr to data. The first values from linArr must be copied in left to right order across the first row of data, the following values of linArr must then be copied to the second row, and so on. For example, [1, 2, 3, 4] would be put into a 2 by 2 array with [1, 2] in the first row and then [3, 4] in the next row.
- public int getNumRows()
 - Return the number of rows in the matrix
- public int getNumCols()
 - Return the number of columns in the matrix
- public double[][] getData()
 - Return the 2-dimensional array containing the matrix data
- public double getElement(int r, int c)
 - Return the value from data at row r and column c
- public void setElement(int r, int c, double value)
 - store value at row r and column c of data
- public void transpose()
 - Transpose the matrix and update the instance variables to store the transposed matrix as the new instance of 'this' matrix (* see explanation below about how to transpose a matrix *)
- public Matrix multiply (double scalar)
 - Create a new Matrix object with the same dimensions as 'this' matrix.
 - Multiply each value in 'this' matrix by the given scalar and insert the resulting values into the new Matrix object and return it (* see explanation below *)
- public Matrix multiply (Matrix other)

Assignment 1

CS 1027 Computer Science Fundamentals II

- If the number of columns of 'this' matrix is not equal to the number of rows of other matrix, return null immediately.
- If the above condition is not true, create a new Matrix object with the number of **rows** from 'this' and the number of **columns** from other.
- Multiply 'this' Matrix object by other and insert the resulting values into the new matrix object and return it. (* see explanation below *)
- public String toString()
 - If the matrix data array is empty, return "Empty matrix"
 - Otherwise, build a string containing the entire matrix following the format shown below and return that string (* see explanation below *)

Transposing a Matrix

Given a matrix M, transposing M into matrix N means reflecting all the elements from M along the diagonal to go into N. In other words, the first row of M becomes the first column of N, the second row of M becomes the second column of N, and so on, thus $N[i][j] = M[j][i]$ for all i, j indices. This means the number of rows and columns must be swapped.

Example:

$$\begin{bmatrix} 8 & 12 \\ 3 & 7 \\ 19 & 5 \end{bmatrix} \xrightarrow{\text{transpose}} \begin{bmatrix} 8 & 3 & 19 \\ 12 & 7 & 5 \end{bmatrix}$$

Matrix-Scalar Multiplication

When multiplying a matrix by a scalar (single number), each number in the matrix must be multiplied by that scalar. The resulting matrix is the same size as the original matrix and each number is just multiplied by that scalar's value.

Example:

$$\begin{aligned} & 3 \times \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 3 \times 3 & 3 \times 0 \\ 3 \times 2 & 3 \times 1 \end{bmatrix} \\ &= \begin{bmatrix} 9 & 0 \\ 6 & 3 \end{bmatrix} \end{aligned}$$

Matrix-Matrix Multiplication

Multiplying two matrices together can only be done if the number of columns of the first matrix (this) is equal to the number of rows of the second matrix (other). Consider 2 matrices A and B satisfying the above condition and let M be the matrix obtained by multiplying A and B. The

Assignment 1

CS 1027 Computer Science Fundamentals II

entry $M[i][j]$ is obtained by multiplying the i -th row of A with the j -th column of B . To multiply a row r with a column c we add the products $r[i] \times c[i]$ for all the values stored in r and c , so

$$M[i][j] = \sum_{l=0}^{n-1} r[l] \times c[l]$$

where n is the number of values in row r and column c . For the example below, the top-left value in the resulting matrix will be calculated by multiplying the top row of the first matrix $[3, 0]$ by the left column of the second matrix $[7, 4]$ (vertical vector) so we multiply 3 by 7 and then 0 by 4 and add them together to get 21. For the top-right element, we multiply the top row of the first matrix by the right column of the second matrix: $3 \times 2 + 0 \times 3 = 6$.

Example:

$$\begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 7 & 2 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 3 \times 7 + 0 \times 4 & 3 \times 2 + 0 \times 3 \\ 2 \times 7 + 1 \times 4 & 2 \times 2 + 1 \times 3 \end{bmatrix} = \begin{bmatrix} 21 & 6 \\ 18 & 7 \end{bmatrix}$$

Matrix toString Format

When returning the string for a non-empty matrix in the `toString()` method, you must format it in a specific way. Each value in the matrix must be converted to a string of exactly 8 characters (that includes numbers, decimal point, and spaces needed to complete 8 characters) in which the number of decimal digits is exactly 3. Start with an empty string S and add the string representation of the first value in the first row of the matrix; then concatenate to S the string representation of the second number in the first row, then the string for the third number and so on. After the last number in the first row, add to S a newline character `'\n'` so if you print S the next row will be shown below the first one. Then, concatenate to S the strings for the numbers of the second row of the matrix followed by `'\n'`; then concatenate the strings for the numbers in the third row, and so on. Do not add any manual spaces or tabs anywhere in the string. **Hint:** use the `String.format()` method and look up how to use it to specify the total spaces and the number of decimal points.

Example:


This is how the string produced by this method might look like when printed:

Assignment 1

CS 1027

Computer Science Fundamentals II

5.750	8.000	2.990
6.130	10.900	4.250
7.500	0.003	12.500



Each value takes up 8 total spaces
and shows 3 decimal places

Vector.java

This class is used to represent a Vector. Recall that vectors are special kinds of matrices so this class **must inherit** from the Matrix class. In the explanation above, vectors are defined as having either one row or one column; but in this assignment we will create them to be horizontal, that is they have 1 row.

No instance variables are needed in this class.

The class must have the following public methods:

- public Vector (int c): **constructor**
 - Use super() to call the Matrix constructor and send in 1 as the value of r
- public Vector (int c, double[] linArr): **constructor**
 - Use super() to call the Matrix constructor and send in 1 as the value of r
- public double getElement (int c)
 - Return the value at row 0 and column c (since it only has 1 row)

MarkovChain.java

This class represents a Markov Chain to produce a matrix of predicted future states using a state vector and a transition matrix.

The class must have these private instance variables:

- private Vector stateVector;
- private Matrix transitionMatrix;

The class must have the following public methods:

- public MarkovChain (Vector sVector, Matrix tMatrix): **constructor**
 - Initialize the instance variables with the given parameters
- public boolean isValid ()
 - Check if the instance variables are valid for a Markov chain problem and return a Boolean to indicate its validity (true if valid, false if invalid):

To be valid, the transition matrix must be a square (equal number of rows and columns) and that number must also match the number of columns in the state vector. Further, the sum of values in the state vector must equal 1.0 (*see note below) and the sum of values in **each** row of the transition matrix must equal 1.0 (*see note below)

- public Matrix computeProbabilityMatrix (int numSteps)
 - First, call isValid() to see if this is a valid Markov chain. If not, return `null` immediately.
 - Compute the probability matrix of this Markov chain after the given numSteps by multiplying the transition matrix by itself numSteps-1 times and multiplying the state vector by the resulting matrix. **Note that you must multiply the state vector by the matrix and not the other way around.** Return the resulting vector (note that it will technically be a Matrix object, not a Vector object, but it will be a 1-dimensional matrix with the same dimensions as the state vector).

*Note: due to possible roundoff errors, checking if the sum is equal to 1.0 should not be done using `==` but rather checking if the value is very close to 1.0. For simplicity, you can check if the sum value is between 0.99 and 1.01.

Provided files

TestMatrixVector.java and TestMarkov.java are tester files to **help** check if your java classes are implemented correctly. TestMatrixVector.java tests your Matrix.java and Vector.java files, and TestMarkov.java tests your MarkovChain.java file.

Similar files will be incorporated into Gradescope's auto-grader. **Passing all the tests within these files does not necessarily mean that your code is correct in all cases.**

Marking Notes

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?
- Does the code run properly on Gradescope (even if it runs on Eclipse, it is **up to you** to ensure it works on Gradescope to get the test marks)
- Does the program produces compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty.
- Including a "package" line at the top of a file will receive a penalty.

Remember **you must do** all the work on your own. **Do not copy** or even look at the work of another student. All submitted code will be run through similarity-detection software.

Submission (due Wednesday, February 8 at 11:55 pm)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see [these instructions](#) on submitting on Gradescope.

Rules

- Please only submit the files specified below.
- Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.**
- You are expected to perform additional testing (create your own test harness class to do this) to ensure that your code works for a variety of cases. We are providing you with some tests but we may use additional tests that you haven't seen for marking.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code as many times as you wish, however, re-submissions after the assignment deadline will receive a late penalty.

Files to submit

- Matrix.java
- Vector.java

Assignment 1

CS 1027
Computer Science Fundamentals II

- MarkovChain.java

Grading Criteria

Total Marks: 20

15 marks	Passing Tests: 15
5 marks	Non-functional Specifications (code readability, comments, correct variables and functions, etc.)