Matrix Project

Due: February 21st, 11:59pm

1 Introduction

In this project, you will use MATLAB to implement some basic matrix and vector operations, some less basic matrix and vector operations, and then you will learn about a really cool application of matrices and vectors.

2 Design Recipe

You are required to follow the CS4 Design Recipe for each function that you create; the first step is to create comprehensive test cases for each of the functions you create.

Your test cases involving numeric matrix and vector results should check that they are within 1e-10 of what you expect using the L_2 norm (described below). Below is an example assert statement for comparing two matrices.

```
assert(myNorm(A-B)<1e-10,'Test 4) A neq B!!')
```

Place all your test cases and comments that explain them in a single file called matrix.m. Indicate each section of the file using `%% Problem N`. The write up for each function include a part A, containing test cases and a part B with a print out of your function. e.g.,

```
%% Problem 1: L2 Norm

% A) Test cases
assert(myNorm(zeros(4,4))==0, 'Zeros are zero')
...

% B) Function Listing(s)
dbtype myNorm.m
dbtype myNormalize.m
```

3 Permitted Built-In Functions

The true power of MATLAB comes from its extensive library of built-in functions. However, for this project one of the focuses is on iteration, so you may only use the following built-in functions (type `help function_name` into MATLAB if you don't know how a function works).

1. All arithmetic operations; that is, $\hat{} + - .* ./$ etc
2. All relational operators: $==, <, >, <=, >=$
3. All logical operators: &, &&, |, ||, , `xor`, etc
4. Horizontal and vertical concatenation; that is, [A A] and [A; A].
5. The colon operator (:) and linspace.
6. `ceil`
7. `floor`
8. `sqrt`
9. `eye`
10. `all`
11. `length`
12. `size`
13. `zeros`
14. `det`
15. `assert`
16. `error`
17. `disp, sprintf, fprintf,etc`
18. `if-elseif-else`
19. `for, while`

4 How to download and hand in this project

To download this project, open a Brown CS terminal window and run `cs4_install matrix`.

To hand in this project, move all of your solution files into the matrix directory, cd into the directory (use ls to make sure all of your files are there!) and run `cs4_handin matrix`.

5 What Is a Vector?

A vector is a list of numbers. For example, $(1, 2)$ or $(5, 6, 4, 3, 2, 2, 5)$ are vectors. Vectors can be used for many applications. We can think of a 2-element vector as the (x, y) coordinates of a point in a 2D coordinate plane, or a 3-element vector as the (x, y, z) coordinates of a point in the 3D coordinate plane. Similarly, we can conceptualize an n-element vector as the $(x_1, x_2, \cdots, x_n)$ coordinates of a point in an n-dimensional space. In MATLAB, vectors are are always stored in either row $(1 \times n)$ or column $(n \times 1)$ format.

6 L_2 Norm

The norm of an n-element vector (for our purposes) is the root of the sum of the squares of the elements in the vector:

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

A unit vector is a vector with a norm of 1. To convert a vector into a unit vector, we divide each element in the vector by that vector's norm. This is known as normalizing the vector, i.e. $x/\|x\|$ is a unit vector that points in the same direction as x.

The L_2 norm is also used to study matrices. It is calculated by summing the squared elements of the matrix and then taking the square root.

Problem 1: L_2 Norm

Implement the calculation of the vector norm and the normalization of a vector by filling out the `myNorm` and `myNormalize` functions. The `myNorm` function should accept a vector or matrix and return the the L_2 norm, and the `myNormalize` function should accept a vector and return a normalized version of that vector. Your `myNormalize` function MUST use your `myNorm` function.

For this problem, and all the others require you to create a function, be sure to follow the CS4 Design Recipe (i.e., design analytic test cases first, code your function body, and then re-test/fix the code, and add tests.)

MATLAB has a built-in function `norm()`, that you can use as an additional check of your work, but obviously, do not use it in your function implementations.

7 The Dot Product

Vectors are easy to add and subtract. The sum of two vectors is

$$(x_1, x_2, ...x_n) + (y_1, y_2, ...y_n) = (x_1 + y_1, x_2 + y_2, ...x_n + y_n)$$

and the difference is defined in the same way. However, there is more than one way to multiply two vectors. One such way is the dot product of the vectors. The dot product is the sum of the products of each element in the vectors. The dot product can be expressed as

$$(x_1, x_2, \cdot, x_n) \cdot (y_1, y_2, \cdots, y_n) = \sum_{i=1}^{n} x_i y_i$$

Note that the dot product is only defined for vectors of the same length, and that the dot product of 2 vectors is a scalar (a single number).

Problem 2: Dot Product

Implement the dot product by filling out the `myDot` function. The `myDot` function should accept 2 vectors and return the dot product of those vectors. If the 2 vectors are not the same length, your function use the MATLAB `error` command to print out an appropriate error message.

MATLAB has a built-in function `dot()`, which you can use as an additional check of your work, but obviously do not use it in your function implementation.

8 What is a Matrix?

A matrix is two-dimensional array of numbers. For example,

$$\begin{pmatrix} 1 & 5 & 1 \\ 2 & 2 & 4 \\ 0 & 1 & 0 \end{pmatrix}$$

or

$$\begin{pmatrix} 1 & 5 & 1 \\ 2 & 2 & 4 \end{pmatrix}$$

are matrices. Matrices have a wide variety of applications, such as solving systems of equations, storing data, and applying functions. Vectors are special cases of matrices. A **column vector** is a matrix with a single column. A **row vector** is a matrix with a single row. A matrix with the same number of rows as columns is a **square matrix**.

9 Matrix Multiplication

Like vectors, matrices are easy to add and subtract. For example,

$$\begin{pmatrix} 1 & 4 & 1 \\ 2 & 4 & 4 \\ 0 & 1 & 8 \end{pmatrix} + \begin{pmatrix} 0 & 5 & 2 \\ 2 & 6 & 4 \\ 1 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 1 & 9 & 3 \\ 4 & 10 & 8 \\ 1 & 3 & 14 \end{pmatrix}$$

Also like vectors, only matrices with identical dimensions (the same number of rows and columns) can be subtracted or added to each other. However, matrix multiplication (as opposed to element-wise multiplication) is more complicated. For matrices A and B, multiplying AB requires the number of columns in A to match the number of rows in B. Note: AB is also sometimes written $A*B$.

$$\begin{pmatrix} 1 & 4 & 1 \\ 2 & 4 & 4 \end{pmatrix} \begin{pmatrix} 0 & 5 \\ 2 & 6 \\ 1 & 2 \end{pmatrix}$$

is defined, but the product

$$\begin{pmatrix} 0 & 5 \\ 2 & 6 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 4 & 1 \\ 2 & 4 & 4 \\ 2 & 4 & 6 \end{pmatrix}$$

is not defined.

Formally, the matrix product AB, where A is $m \times n$ and B is $n \times p$ is as follows:

$$(AB)_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$

$A * B$ is an $m \times p$ matrix.

Note: The above equation is equivalent to the dot product of the vector formed from row i of A and the vector formed from column j of B. Therefore, we can also define element (i, j) of the matrix product A*B as the dot product of the i^{th} row of A and j^{th} column of B.

Also note that the multiplication of a row vector with a column vector is a dot product.

$$\begin{pmatrix} 0 & 5 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} = 20$$

Also, note that matrix multiplication is not commutative:

$$\begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} \begin{pmatrix} 0 & 5 & 5 \end{pmatrix} = \begin{pmatrix} 0 & 5 & 5 \\ 0 & 10 & 10 \\ 0 & 10 & 10 \end{pmatrix}$$

Problem 3: Matrix Multiplication

Note: In MATLAB, the product of 2 matrices A and B is simply A*B, using the asterisk symbol to indicate multiplication. You may use this as an additional check of your work. Obviously, do not use this in your function implementation.

Implement matrix multiplication by filling out the `myMatrixMult` function. This function should accept 2 matrices and return their product. Your function should use your `myDot` function. If the product of the input matrices does not exist, your function should use `error` to indicate that there is a problem.

Before you write any code, try to compute the following products by hand and check your work in MATLAB in order to ensure that you understand the algorithm.

$$\begin{pmatrix} 1 & 6 & 0 \\ 0 & 4 & 4 \\ 0 & 2 & 2 \end{pmatrix} \begin{pmatrix} 5 & 6 & 3 \\ 0.5 & 8 & 1 \\ 1 & 3 & 14 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 16 & 1 \end{pmatrix} \begin{pmatrix} 1 & 6 & 0 \\ 5 & 3 & 1 \\ 0 & 2 & 8 \end{pmatrix}$$

10 Matrix Transpose

The transpose of a matrix A (denoted A^t) is the matrix formed by switching the rows and columns. That is, element $A^t_{i,j} = A_{j,i}$. For example,

$$\begin{pmatrix} 1 & 4 & 1 \\ 2 & 4 & 7 \\ 2 & 8 & 4 \end{pmatrix}^T = \begin{pmatrix} 1 & 2 & 2 \\ 4 & 4 & 8 \\ 1 & 7 & 4 \end{pmatrix}$$

A matrix that is equal to its own transpose is called a **symmetric matrix**.

Problem 4: Transpose

Implement the functions `myTranspose` and `myIsSymmetric`. `myTranspose` should accept a matrix as input and return its transpose, and `myIsSymmetric` should accept a matrix as input and return 1 if the matrix is symmetric and 0 otherwise.

The transpose of a matrix A in MATLAB is A'. You should use this to check your work. Obviously, do not use this command in your function implementations.

11 The Identity Matrix

The Identity Matrix is a square matrix with ones along the diagonal and zeros everywhere else. For example,

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \textit{and} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

are identity matrices. When we multiply a matrix (of the correct size) by the identity matrix, the matrix does not change. That is, $AI = A$ and $IA = A$. To construct an $n \times n$ identity matrix in MATLAB, use the command `eye(n)`.

12 Matrix Inverse and Gaussian Elimination

The inverse of a square matrix A (denoted A^{-1}) is the matrix such that $AA^{-1} = I$ and $A^{-1}A = I$. Not all matrices have inverses. In order to check whether a matrix A has an inverse, use the command `det(A)`. If `det(A)` $\neq 0$, then A is invertible. Otherwise, A is not invertible. Matrices that are not invertible are sometimes referred to as **singular**.

Matrix inverses provide a way to solve linear systems. For example, say we have the equation

$$Ax = b$$

where A is a matrix and x and b are vectors. If we know A^{-1}, we can multiply both sides of the equation to arrive at the solution, e.g.,

$$\begin{gathered} Ax = b \\ A^{-1}Ax = A^{-1}b \\ Ix = A^{-1}b \\ x = A^{-1}b \end{gathered}$$

However, finding a matrix inverse can be challenging. We will use an algorithm based on techniques with which you are probably familiar from high school.

12.1 Gaussian Elimination

You've probably already used Gaussian elimination to solve systems of equations, even if you haven't called it by that name. To find the inverse of a matrix, we're trying to solve an equation of the form

$$AX = I$$

where $A^{-1} = X$. In order to solve this, we want to find a set of 'legal' transformations (described below) that convert the above equation to the one below.

$$IX = X$$

As we perform these transformations on the original matrix, A is converted into the identity matrix, and the matrix that begins as the identity matrix on the right hand side becomes the inverse matrix.

To perform the transformation, we transform the equation by applying 'elementary row operations,' or operations that can be encoded as a matrix multiplication, to each side. These include the following operations:

1. Swapping rows
2. Multiplying a row by a scalar value
3. Adding a multiple of one row to another row

The Gaussian elimination algorithm starts with 2 matrices: the matrix A that you want to invert, and the identity matrix B. Perform elementary row operations on A until A is the identity matrix. Whenever you perform an operation on A, perform the same operation on B (for example, if you add 5*(row 1) of A to row 2 of A, then add 5*(row 1) of B to row 2 of B).

The code you write might adopt the following outline:

1. Perform row operations on both matrices so that the first column of A has only one nonzero entry.
2. Repeat this process for the second column, the third column, etc. until each column has only one nonzero entry.
3. Swap the rows of both matrices so that the first row of A has a nonzero entry in the first column, the second row has a nonzero entry in the second column, etc.
4. Convert A to the identity matrix by multiplying each row of both matrices by a scalar value.

Example:

$$A = \begin{pmatrix} 2 & 1 & -3 \\ 4 & -2 & -6 \\ -1 & -2 & 2 \end{pmatrix} B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Because we want to transform this into the identity matrix, we only want one nonzero entry in each column. Starting with the first column, let's have the first row keep its 2 as our nonzero entry. Then we need to add -2 times the first row to the second, and $\frac{1}{2}$ times the first row to the third, to make the other entries

in the first column zeros. Our matrix now looks like

$$A = \begin{pmatrix} 2 & 1 & -3 \\ 0 & -4 & 0 \\ 0 & -1.5 & 0.5 \end{pmatrix} B = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0.5 & 0 & 1 \end{pmatrix}$$

Move on to the second column. This time, let -4 be the nonzero entry. Add $\frac{1}{4}$ times the middle row to the top row, and $-\frac{3}{8}$ times the middle row to the bottom row:

$$A = \begin{pmatrix} 2 & 0 & -3 \\ 0 & -4 & 0 \\ 0 & 0 & 0.5 \end{pmatrix} B = \begin{pmatrix} 0.5 & 0.25 & 0 \\ -2 & 1 & 0 \\ 1.25 & -0.375 & 1 \end{pmatrix}$$

For the last column, let 0.5 be the nonzero entry, and add 6 times the third row to the first row:

$$A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & 0.5 \end{pmatrix} B = \begin{pmatrix} 8 & -2 & 6 \\ -2 & 1 & 0 \\ 1.25 & -0.375 & 1 \end{pmatrix}$$

Now that the matrix is diagonal (the only nonzero elements are along the "diagonal"), we need to multiply each row by a scalar to reach the identity matrix. Multiply the first row by $\frac{1}{2}$, the second by $\frac{1}{4}$, and the third by 2:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 4 & -1 & 3 \\ 0.5 & -0.25 & 0 \\ 2.5 & -0.75 & 2 \end{pmatrix}$$

B is now equal to the inverse of our original matrix!

Problem 5: Inversion

In this section, you will fill out the function `myGaussInvert`. This function accepts a matrix A and returns A^{-1}. If A is singular, then your function should halt and print out the line 'This matrix is not invertible'. You can use any of the functions that you've written so far in this project. Feel free to write other helper functions if you feel it would be helpful, but this is not required.

In MATLAB, the command A^{-1} (or `inv(A)`) returns the inverse of A. You may use this function as an additional check of your work. Obviously, do not use this command in your function implementations.

This function is a little tricky to write. Be sure to test it with a variety of different matrices.

Before you write any code, try inverting the following matrix by hand. Check your work in MATLAB.

$$\begin{pmatrix} 9 & 10 & 3 \\ 0 & 7 & 6 \\ 2 & 1 & 0 \end{pmatrix}$$

13 Markov Chains

As a cool application of matrices, we are going to look at Markov Chains, which are a very popular and useful concept in applied mathematics.

13.1 What is a Markov Chain?

A Markov chain is a mathematical system that describes a sequence of possible transitions from one state to another on a state space (a set of states). It is a random process usually characterized as memoryless: the next state depends only on the current state and not on the sequence of events that preceded it. This specific kind of "memorylessness" is called the Markov property.

13.2 What is a Transition Matrix?

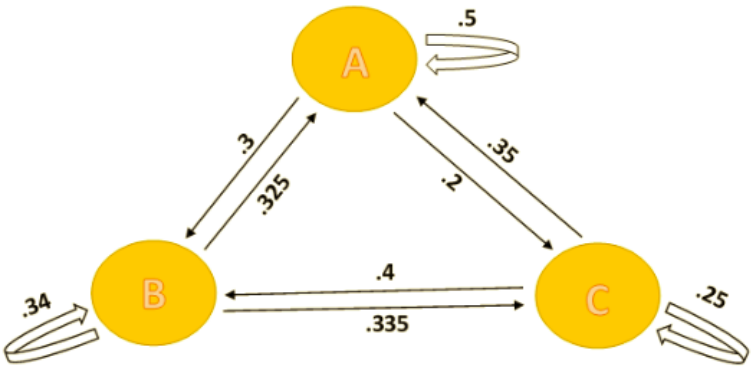
A transition matrix is a matrix that represents the probabilities that some given state in a state space will transition to another. Each column represents a given state and each row represents the the probabilities of going to other states. That is, $M_{i,j}$, the element in the i^{th} row and j^{th} column of the transition matrix M, is $P_{j,i}$, the probability that state j will transition to state i.

For example, in a transition matrix where the states are islands, if $M_{3,2} = \frac{1}{2}$ the probability that a person at island 2 will move to island 3 is $\frac{1}{2}$.

13.3 What is a State Vector?

A state vector is a column vector that represents a state probability distribution. For example, the vector

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$



would indicate that the state space consists of 4 states, and that the current state is state 1. Similarly, the vector

$$\begin{pmatrix} 0 \\ 0.5 \\ 0.5 \end{pmatrix}$$

would indicate that the state space consists of 3 states, and that the current state is equally likely to be state 2 or state 3. Note that since a state vector describes a probability distribution, the elements in a state vector must sum to 1.

13.4 Using Transition Matrices

Consider a Markov chain described by the transition matrix M. If the current state vector is v, then the next state vector will be the product Mv. The proof is left as an exercise to the reader.

13.5 Example

Let's look at a simple example, described by with the above graph. In CS4 world, there are three towns: A, B, and C.

1. Of the people who visit town A on some day, 50% of them will stay in A on the next day, 30% of them will visit town B on the next day, and 20% of them will visit town C on the next day
2. Of the people who visit town B on some day, 34% of them will stay in B on the next day, 32.5% of them will visit town A on the next day, and 33.5% of them will visit town C on the next day

3. Of the people who visit town C on some day, 25% of them will stay in C on the next day, 35% of them will visit town A on the next day, and 40% of them will visit town B on the next day

The transition matrix M for this Markov chain is

$$\begin{pmatrix} .5 & .325 & .35 \\ .3 & .34 & .4 \\ .2 & .335 & .25 \end{pmatrix}$$

If today, half of the people are at town A and half of the people are at town C, then the current state vector is

$$\begin{pmatrix} 0.5 \\ 0 \\ 0.5 \end{pmatrix}$$

and tomorrow's state vector is

$$\begin{pmatrix} .5 & .325 & .35 \\ .3 & .34 & .4 \\ .2 & .335 & .25 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.425 \\ 0.350 \\ 0.225 \end{pmatrix}$$

Problem 6: Markov Chains

Answer the following questions as comments at the bottom of your `matrix.m` file. Continue to use sectioning to separate problems.

Why?

Problem 6.1

If v is the state vector for day 1, prove (informally) that Mv is the state vector for day 2. HINT: Use the definition for matrix multiplication.

Problem 6.2

If v is the state vector for day 1, and Mv is the state vector for day 2, what is the state vector for day 3? How about for day 8? Prove it.

Weather

Now, we will build a weather prediction model that uses Markov chains to predict tomorrow or next week's weather based on today's weather.

Problem 6.3: Creating a Transition Matrix

A transition matrix can be used to represent changes to the weather from a given day to the next. We will assume there are four types of possible weather in CS4 world: sunny, cloudy, rainy, and snowy. Say that the following probabilities exist:

1. If it is sunny today, there is a 60% chance it will be sunny tomorrow, a 10% chance it will be rainy tomorrow, a 20% chance it will be cloudy tomorrow, and a 10% chance it will be snowy tomorrow.
2. If it is rainy today, there is a 30% chance it will be sunny tomorrow, a 20% chance it will be rainy tomorrow, a 30% chance it will be cloudy tomorrow, and a 20% chance it will be snowy tomorrow.
3. If it is cloudy today, there is a 20% chance it will be sunny tomorrow, a 30% chance it will be rainy tomorrow, a 30% chance it will be cloudy tomorrow, and a 20% chance it will be snowy tomorrow.
4. If it is snowy today, there is a 10% chance it will be sunny tomorrow, a 30% chance it will be rainy tomorrow, a 30% chance it will be cloudy tomorrow, and a 30% chance it will be snowy tomorrow.

Build a transition matrix for the weather of the CS4 world. What are the dimensions of the transition matrix? Why? Why do each of the columns of a transition matrix always need to sum to 1?

Problem 6.4: Predicting tomorrow's weather

Say that today is Monday, and it is definitely sunny today. That is, today's state vector is

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

What will Tuesday's (tomorrow's) state vector be? Create a function called `myWeatherReport` that takes in a weather prediction state vector and the number of days in the future, and prints out a forecast as it were a weatherperson on TV (i.e. "In 2 days there is a 20% chance of rain, a 50% chance of sun, a 30% chance of clouds and no chance of snow"). HINT: Use `fprinf`

Problem 6.5: Predicting the weather in 3 days

What will Thursday's state vector be? Use `myWeatherReport` to print out your forecast.

Problem 6.6: Yesterday's weather

For the next 2 problems, say that today's state vector is

$$\begin{pmatrix} .295 \\ .230 \\ .270 \\ .205 \end{pmatrix}$$

What was yesterday's state vector? HINT: Solve Ax = b.

Problem 6.7: Predicting the Weather in 10, 20, and 30 days

What will the state vector describing the weather be in 10 days? How about 20 days? 30 days? How do the results look? Why do you think this happened?

Problem 6.8: Steady State

In section 6.7 we observed a pattern in future weather. Read the Wikipedia page about Markov Chains (`http://en.wikipedia.org/wiki/Markov_chain#Steady-state_analysis_and_limiting_distributions`) and explain what a steady state (stationary) distribution is.

Problem 6.9: Non-Steady State

Can you think of a 4 x 4 transition matrix and an initial state vector that never converge to a stationary vector?