Introduction

In CS 9A, you learn to program in Matlab. You are assumed to have had sufficient previous programming experience in a high-level language like Java, C, or Scheme, and to be familiar with applications of matrix processing.

Course material consists of quizzes, which test your knowledge of language and low-level conceptual details, and programming assignments, which exercise your overall command of the language. This volume supplies a framework for the course. It contains the following:

Study modules. Each module focuses on a particular programming topic. It provides references to textbook material describing the topic, and suggests exercises for self-study. The study modules reference the following texts.

Mastering MATLAB 7, Hanselman and Littlefield (Prentice Hall, 2005).

Learning MATLAB 7, The Mathworks (supplied with the student version of MATLAB)

Programming assignments. Each one has a header page (this tells you the title and related topics) that is followed by the actual assignment.

Sample quiz questions, with solutions. These help you prepare for the quizzes.
Structure of quizzes and programs in CS 9A

The following table outlines the relationship between quizzes and programs. There are four quizzes you must take and five sets of programs you must write. All the material for a particular group must be completed before material in the next group; however, quizzes and programs within a group may be done in any order.

<table>
<thead>
<tr>
<th>group</th>
<th>programs</th>
<th>quizzes</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Orientation</td>
<td>Introduction to array and matrix operations</td>
</tr>
<tr>
<td></td>
<td>Array manipulation</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>Flow-of-control project</td>
<td>Advanced array and matrix operations</td>
</tr>
<tr>
<td>c</td>
<td>Image manipulation</td>
<td>Plotting and function handles</td>
</tr>
<tr>
<td></td>
<td>Cell arrays and structures</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>Symbolic mathematics toolbox exercises</td>
<td>Symbolic math operations</td>
</tr>
</tbody>
</table>

Note that this breakdown is different from what’s required to satisfy course deadlines. For information about deadlines, consult the “Information and Regulations” document.

Quizzes focus on Matlab details, while the programming assignments provide a context that pulls the details together. In our other self-paced programming courses, whether a student should complete a program before or after the corresponding quiz depends on his or her personal learning style; some students learn better “bottom-up” by familiarizing themselves with details before going on to apply them, while others benefit from the “big picture” provided by the programming assignment. Matlab, however, has a very large set of builtin functions, and we don’t come close to covering them all in CS 9A. Someone trying to learn all the details of a given segment before trying to use them will, we think, waste a lot of time. We therefore suggest that in each segment you start by working on the programming assignment(s). Once you make sufficient progress on the programs, move on to studying for the quiz.
Orientation

Goals
In this activity, you familiarize yourself with software to compute deadline penalties in the self-paced courses and supply us with some administrative information that makes it easier to contact you.

Readings
The “Information and Regulations” pamphlet.

Problem
The “Orientation” assignment will be distributed at the Self-Paced Center. You must complete it to enable any of your other work to be recorded. Complete it as early in the semester as possible.
Program—Array manipulation

This programming assignment introduces you to array processing and plotting in Matlab.

Readings

*Mastering MATLAB 7*, chapters 2 through 4, sections 5.1 through 5.8 and 5.12, 26.1 through 26.3, chapter 37, and sections 38.1 and 38.2.

*Learning MATLAB 7*, chapter 3, pages 4-1 through 4-17, 4-21 through 4-25, 4-28 through 4-30, 5-38 through 5-50, and chapter 8.

If you are unfamiliar with the EECS instructional systems, you should also download and read the document “Before you begin …” available at the self-paced course web site.

Related quizzes

Introduction to array and matrix operations

Programming assignment

Do both the programming assignments described on the following pages.
Array manipulation 1—Correlation coefficient

Problem

Write a MATLAB M-file that reads two arrays of equal size from the user and determines their correlation coefficient. The correlation coefficient is a number between –1 and 1; coefficient values close to –1 or 1 mean that there is a linear relation between corresponding data values in the two arrays. To find the correlation coefficient between values in arrays xvalues and yvalues, do the following.

- Convert the values to standard units. That is, for each value in xvalues, subtract the mean of the xvalues values and divide by the xvalues standard deviation, and for each value in yvalues, subtract the mean of the yvalues values and divide by the yvalues standard deviation.

- The average of the products of corresponding standard unit scores gives the correlation coefficient.*

Here’s an example. The correlation coefficient is 0.40.

<table>
<thead>
<tr>
<th>xvalues values (mean = 4, SD = 2)</th>
<th>yvalues values (mean = 7, SD = 4)</th>
<th>xvalues values in standard units</th>
<th>yvalues values in standard units</th>
<th>products</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>-1.5</td>
<td>-0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>-0.5</td>
<td>0.5</td>
<td>-0.25</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.5</td>
<td>-1.5</td>
<td>-0.75</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>1.5</td>
<td>1.5</td>
<td>2.25</td>
</tr>
</tbody>
</table>

Miscellaneous requirements

Every line in your M-file should be an assignment to a scalar or an array. The first two lines should call the input function; others should carry out the computation. You should assign the correlation coefficient to a variable named answer in the last statement of the M-file.

Matlab provides std and corrcoef functions that implement somewhat different computations. Don’t use these.

Test your M-file on the above values as well as on a pair of arrays that produces a correlation coefficient of –0.5. Be prepared to verify your computation to a tutor.

* Source: Statistics, by Freedman, Pisani, and Purves (Norton, 1978). Some authors describe a difference computation for the correlation coefficient, the result of dividing the sum of the products of corresponding standard unit scores by N–1 instead of by N, where N is the number of scores in each list. This obviously produces a different value.
Checklist

A correctly working M-file, each of whose lines is an assignment statement as described, and whose last line is an assignment to the variable `answer`.

Avoidance of the built-in `std` and `corrcoef` functions.

A printed listing of your command window, displaying the specified test cases and results.

A printed listing of your M-file, with each line accompanied by comments that describe its purpose and its inputs.
Array manipulation 2—Generalized dice probabilities

Background

As children, many of us loved to play Yahtzee and Backgammon and other games with dice. As adults we play Dungeons and Dragons (and other role-playing games) and learn about dice that have more than six sides. Now that we have the benefit of Matlab, we’d like to be able to ask questions about the probability distribution of the sum of \( n \) different dice, each with \( s \) sides. There’s a wonderful mathematical connection between this question, the coefficients of polynomials, Pascal’s triangle, the Sierpinski Sieve, Galton boards, binomial distributions, and bell curves.

Let’s start with the simplest possible non-trivial case. If we have a two-sided die (a coin!), we would expect that the probability of it landing on the “1” side (heads) is 0.5, the same as the probability of it landing on the “2” (tails) side. Similarly, if we roll a single six-sided die (\( s=6 \)), we would expect a \( \frac{1}{6} \) probability that it would land on any of the sides 1 to 6.

What if we throw two 2-sided dice? What would the probability distribution of the sum of all of the dice be? Well, there are four outcomes (but only three sums!), as highlighted in the table below. (You should fill in the blanks.) As a sanity check, we recall the sum of the probabilities of all outcomes must be unity.

<table>
<thead>
<tr>
<th>die #1</th>
<th>die #2</th>
<th>sum</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

What if we threw two 6-sided dice? What would the probability distribution of the sums be then? It gets harder to make the table. Fortunately, there’s a built-in Matlab function that can help us with this, namely conv:

```plaintext
>> help conv
CONV Convolution and polynomial multiplication.
C = CONV(A, B) convolves vectors A and B. The resulting
vector is length LENGTH(A)+LENGTH(B)-1.
If A and B are vectors of polynomial coefficients, convolving
them is equivalent to multiplying the two polynomials.

>> coin = [0.5 0.5]; % Distribution of a 2-sided die (coin)
>> conv(coin, coin) % Note the same answer as our table above!
ans =
0.2500 0.5000 0.2500
```

Please note that when we threw an \( n \)-sided dice for \( m \) time we need to apply the convolution function for \((n-1)\) times.

---

* See mathworld.wolfram.com/GaltonBoard.html for a starting point.
Problem

You are to determine and plot the probability distribution of the sum of one through five 101-sided dice. To normalize the five cases, subtract the median value from each of the sums so that the values (and resulting plots) are all centered about 0. E.g., for the coin problem, the sums were 2, 3 and 4 and median value was 3, so the normalized sums would be –1, 0 and 1. You will generate one large $n \times 6$ two-dimensional array, where $n$ is the number of different outcomes for five 101-sided dice. The first column in this matrix should be the normalized sums; the remaining columns list the probabilities of forming those sums with one die, with two dice, and so on up to five dice.

<table>
<thead>
<tr>
<th>normalized sums</th>
<th>probabilities for one 101-sided die</th>
<th>probabilities for two 101-sided dice</th>
<th>probabilities for three 101-sided dice</th>
<th>probabilities for four 101-sided dice</th>
<th>probabilities for five 101-sided dice</th>
</tr>
</thead>
<tbody>
<tr>
<td>–250</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>a very small value</td>
</tr>
<tr>
<td>–249</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>a very small value</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>–1</td>
<td>0.009900990099009901</td>
<td>0.009900990099009901</td>
<td>0.00742501463165</td>
<td>0.00660001300591</td>
<td>0.00592992664476</td>
</tr>
<tr>
<td>0</td>
<td>0.009900990099009901</td>
<td>0.009900990099009901</td>
<td>0.00742598522179</td>
<td>0.00660098359606</td>
<td>0.0059305329928</td>
</tr>
<tr>
<td>1</td>
<td>0.009900990099009901</td>
<td>0.00990296049407</td>
<td>0.00742501463165</td>
<td>0.00660001300591</td>
<td>0.00592992664476</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>250</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>a very small value</td>
</tr>
</tbody>
</table>

Produce a 2D plot similar to what’s displayed in Figure 1. It will have all the probabilities on it simultaneously in different colors, a legend indicating which each one is, and each curve centered on 0 (i.e., normalized). The x-axis should have the values from your leftmost column above. Put a title on your plot and label the axes. A sample plot appears on the next page.

Miscellaneous requirements

Since you have yet to learn how to define functions and pass in arguments, but we still want you to write good quality code, you should define two constants (these would have been your function parameters), $n$ and $s$, and a dictionary that explains what each means. For example:

```matlab
% DICTIONARY of constants we’ll use
n = 5;  % n is the number of dice we have
s = 101;  % s is the number of sides on each dice
```

One common pattern in Matlab is first to generate all the data and then to analyze it with different plots. You should do the same thing here: break your program into two different parts, the first to build the array, the second to generate the plot.

Changing one line of your code (the “s=101” one) to assign $s$ another odd value should produce a valid array and plot. For now, it’ll be OK to assume $n$ won’t change.

At this stage, your code should not include any loops, if or for constructs. You should comment your code appropriately and save it in an M-file.
Checklist

A correctly working M-file, containing commented definitions for \( n \) and \( s \) and uses for them as specified.

Avoidance of loops and conditional constructs.

A printed listing of your labeled, titled plot.

A printed listing of your M-file, with comments that describe its purpose and its inputs along with internal comments as appropriate.
Quiz—Introduction to array and matrix operations

Goals
This quiz tests you on Matlab operations for creating and manipulating vectors and matrices, and on the data types most commonly used for vector/matrix elements. Questions focus on the colon and semicolon operators, the quote operator to produce the transpose of a vector, operators for subscripting and doing scalar and vector arithmetic, and special variables such as eps, inf, and NaN.

Readings
Mastering MATLAB 7, chapters 2 through 4, sections 5.1 through 5.8 and 5.12.
Learning MATLAB 7, pages 4-1 through 4-17 and 4-21 through 4-25.

Sample questions for the “Introduction to array and matrix operations” quiz
1. Give two different ways to construct the matrix

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

2. Give two different ways to compute, for a vector \( A = [a_1 \ a_2 \ \ldots \ a_n] \), the value \( a_1a_1 + a_2a_2 + \ldots + a_na_n \).

3. What is produced by the following Matlab code?

```matlab
a = [1:-1:-2, 1:2:3]
b = a*2
```

4. Suppose that vectors \( A \) and \( B \) contain the following values.

\[
A = [2 \ -1 \ 5 \ 0]
B = [3 \ 2 \ -1 \ 4]
\]

What is produced by the following Matlab code?

```matlab
C = A./B
C = 2*A + A.^B
C = 2.^B + A
C = 2*B/3.*A
```
Solutions to sample questions for the “Introduction to array and matrix operations” quiz

1. Some possible answers:
   
   \[
   \begin{bmatrix}
   1:3; 4:6; 7:9 \\
   1:3:9; 2:3:9; 3:3:9
   \end{bmatrix}
   \]
   
   \[
   [1,2,3; 4,5,6; 7,8,9]
   \]
   
   \[
   [0:2; 3:5; 6:8] + \text{ones}(3,3)
   \]
   
   \[
   \text{reshape } ([1:9], 3, 3)'
   \]

2.

   \[
   \text{sum}(a.*a)
   \]
   
   \[
   a*a'
   \]

3. Output from Matlab is as follows.
   
   \[
   a =
   \begin{bmatrix}
   1 & 0 & -1 & -2 & 1 & 3
   \end{bmatrix}
   \]
   
   \[
   b =
   \begin{bmatrix}
   2 & 0 & -2 & -4 & 2 & 6
   \end{bmatrix}
   \]

4. Output from Matlab is as follows.
   
   \[
   C =
   \begin{bmatrix}
   0.6667 & -0.5000 & -5.0000 & 0
   \end{bmatrix}
   \]
   
   \[
   C =
   \begin{bmatrix}
   12.0000 & -1.0000 & 10.2000 & 0
   \end{bmatrix}
   \]
   
   \[
   C =
   \begin{bmatrix}
   10.0000 & 3.0000 & 5.5000 & 16.0000
   \end{bmatrix}
   \]
   
   \[
   C =
   \begin{bmatrix}
   4.0000 & -1.3333 & -3.3333 & 0
   \end{bmatrix}
   \]
**Program—Flow of control project**

This project introduces you to the use of flow of control constructs (for, while, if), multidimensional arrays, functions, and 2D visualizations. It comprises two programming assignments, Life1D and Heat Transfer.

**Readings**

*Mastering MATLAB 7*, chapters 7 and 10, sections 11.1 through 11.4, 12.1 through 12.3, 12.6, and chapters 13 and 14. (Our solution to the first programming assignment also uses the bitget function described in section 15.2.)

*Learning MATLAB 7*, pages 4-15, 6-1 through page 6-7, 6-18 through 6-21, and 6-30.

**Related quizzes**

Advanced array and matrix operations

**Programming assignment**

Do both the programming assignments described on the following pages.
CS 9A programming assignment
Life 1D

Background
In 1983, Steven Wolfram introduced elementary cellular automata that had wonderful repetitive and chaotic properties. (These are described at http://mathworld.wolfram.com/ElementaryCellularAutomaton.html.) Starting with a single “live” cell in the center of the top row (with the rest of the row “dead”) and following a simple set of rules based on the cells above, further generations (rows) automatically evolve. If each next generation is displayed below the previous one, a beautiful 2D structure is created, as in the example above. Rules resemble those of Conway’s “Game of Life” (http://en.wikipedia.org/wiki/Conway’s_Game_of_Life), but in one dimension rather than two—hence the name “Life1D”.

Problem
In the file Life1D.m, provide the code for the function Life1D . Life1D takes two arguments, a positive integer n and an integer ruleNum between 0 and 255, inclusive:

- n specifies the number of rows to generate (we don’t count the first “seed row” which is all dead except for the central live cell);
- ruleNum specifies which of the 256 possible rules to use.

Life1D returns an (n+1) by (2n+1) uint8 array of 1’s and 0’s that represents n applications of rule ruleNum to the starting row. The data type uint8 is used because our data is only a 0 (dead) or 1 (live), and uint8 is the data type that uses the least storage (1 byte). The squares off the board should be considered “dead”; your arrays may be different than those on the Mathworld Elementary Cellular Automaton site because of this simpler boundary condition.

You can display your call to Life1D with the following command (assuming you wanted to view 50 generated rows of rule #30, the same rule as in the image above):

```
> bw = gray(2); colormap(bw(end:-1:1,:));
> imagesc(Life1D(50,30)); axis square off equal
```
**Miscellaneous requirements**

Provide comments for each of your functions that explain what its arguments and valid ranges are, and what its return value will be.

Test your code using the rules from Elementary Cellular Automata. Note that our boundary conditions are slightly different than those on the website so some of your images may differ. Test your code on at least rules numbered 30, 90, 110, 250 and 254. Your code should process all rule numbers (between 0 and 255, inclusive) correctly.

You may use either loops (while and for constructs) or vectorization.

Show your tutor a printout of several images with different rules, and a printed listing of your M-file.

**Checklist**

A satisfactory grade on this program requires the following:

- correct results for the Life1D simulation and several images, using at least rules numbered 30, 90, 110, 250 and 254
- printed output of the results of the simulation
- evidence of individual testing of functions
- no routine longer than twenty-four lines of code
- variable and subprogram names that reflect their use
- comments at the head of the main function (and any subroutine you wish to write) that explain its purpose
Background

In this problem, we work with a very simplified model of heat transfer. We will consider a square plate at the edges of which the temperature distribution is known, and determine both by computer simulation and by direct computation the temperature equilibrium of the object.

Problem

This is a two-part problem. The idea is to solve this problem in two ways, in order to be able to check your answer. The two parts may be done in separate programs, or in a single program; the latter is probably better, since you can make more clear that you are computing the answer with one step and checking your answer with the other. Obviously the two answers should agree (within the limits of accuracy). If they don’t, there is a bug in your program.

In both parts, we conceptually represent the interior of the square plate by a 3-by-3 double array, each cell of which contains the temperature in degrees Kelvin of the corresponding interior portion of the plate. Your programs will start by initializing a matrix that represents the temperature at points on the exterior (sixteen values) and the interior (nine values) of the plate. Do this either by executing an assignment statement or by reading its values from a file. Note that the temperatures at the exterior are constant, and the temperatures at the interior are (arbitrary) initial values.

Problem, part I

The program for part I will then enter a loop that simulates heat loss or gain by the plate over a period of time, according to the following rule:

The temperature in cell \([r, c]\) of the 3-by-3 array at state \(t\) is the average of the temperatures in the four cells \([r−1, c], [r, c−1], [r, c+1],\) and \([r+1, c]\) at state \(t−1\). (Cells \([0, c], [r, 0], [4, c],\) and \([r, 4]\) represent points on the exterior of the plate.)
This loop should run until the maximum changes in temperature in all cells of the interior of the plate in a particular unit of time are less than 0.05 degree (you may wish to make this difference smaller to improve the accuracy of the simulation). It should then print the array of (equilibrium) temperature values, along with the number of time units the process took to reach this point.

You will need a copy of the plate to implement the process described above correctly, since if you use only one copy you will be computing values at state \( t \) from other values in state \( t-1 \).

**Problem, part II**

The program for part II will compute the equilibrium temperatures by solving the system of equations that results from the following rule:

The temperature in cell \( [r, c] \) of the plate must be the average of the temperatures in the four adjacent cells \([r-1, c], [r, c-1], [r, c+1], \text{and} [r+1, c] \).

There will be nine such equations, one for each cell in the array. Your program will create a 9x9 double matrix to contain the coefficients of the equations, along with a 9-element double array to contain their right-hand-sides. It will then return the solution of the system of equations.

**Example**

The matrix below represents a plate for which the temperatures on the exterior points are known and the temperatures on the interior are not. (Note that the numbers given here are arbitrary! Your program should be able to handle any set of numbers for the exterior.)

\[
\begin{pmatrix}
0.0 & 4.0 & 8.0 & 12.0 & 16.0 \\
12.0 & x_{11} & x_{12} & x_{13} & 20.0 \\
20.0 & x_{21} & x_{22} & x_{23} & 24.0 \\
20.0 & x_{31} & x_{32} & x_{33} & 28.0 \\
20.0 & 24.0 & 28.0 & 28.0 & 32.0
\end{pmatrix}
\]

Your program for part I will read the values both for the exterior of the plate and for the interior (the \( x_{ic} \) above). The temperatures you supply for the \( x_{ic} \) will initialize the simulation; they should be such that they will convince a tutor that your simulation works. Note that the exterior temperatures don’t change during the simulation.

In part II, you solve the following system of equations:

\[
\begin{align*}
x_{11} &= (12.0 + 4.0 + x_{12} + x_{21})/4 \\
x_{21} &= (20.0 + x_{11} + x_{22} + x_{31})/4 \\
x_{31} &= (20.0 + x_{21} + x_{32} + 24.0)/4 \\
x_{12} &= (x_{11} + 8.0 + x_{13} + x_{22})/4
\end{align*}
\]
\[
x_{22} = (x_{21} + x_{12} + x_{23} + x_{32})/4
\]
\[
x_{32} = (x_{31} + x_{22} + x_{33} + 28.0)/4
\]
\[
x_{13} = (x_{12} + 12.0 + 20.0 + x_{23})/4
\]
\[
x_{23} = (x_{22} + x_{13} + 24.0 + x_{33})/4
\]
\[
x_{33} = (x_{32} + x_{23} + 28.0 + 28.0)/4
\]

The \(x_{rc}\) are not initialized for this part; they are the solutions to the system of equations. The first step in this process is to transform the equations from the form above to the form \(Ax = b\). It helps to rewrite the above equations into a form more in keeping with the eventual matrices needed. We will make two changes: first, we will include all the variables (multiplying by zero when needed), and move the constant term to the right side of the expression. The result is

\[
\begin{align*}
1.0x_{11} &+ -x_{21}/4 + 0*x_{31} + -x_{12}/4 + 0*x_{22} + 0*x_{32} + 0*x_{13} + 0*x_{23} + 0*x_{33} = 16/4 \\
-x_{11}/4 &+ 1.0*x_{21} + -x_{31}/4 + 0*x_{12} + -x_{22}/4 + 0*x_{32} + 0*x_{13} + 0*x_{23} + 0*x_{33} = 20/4 \\
0*x_{11} &+ -x_{21}/4 + 1.0*x_{31} + 0*x_{12} + 0*x_{22} + -x_{32}/4 + 0*x_{13} + 0*x_{23} + 0*x_{33} = 44/4 \\
-x_{11}/4 &+ 0*x_{21} + 0*x_{31} + 1.0*x_{12} + -x_{22}/4 + 0*x_{32} + -x_{13}/4 + 0*x_{23} + 0*x_{33} = 8/4 \\
0*x_{11} &+ -x_{21}/4 + 0*x_{31} + -x_{12}/4 + 1.0*x_{22} + -x_{32}/4 + 0*x_{13} + -x_{23}/4 + 0*x_{33} = 0/4 \\
0*x_{11} &+ 0*x_{21} + -x_{31}/4 + 0*x_{12} + -x_{22}/4 + 1.0*x_{32} + 0*x_{13} + 0*x_{23} + -x_{33}/4 = 28/4 \\
0*x_{11} &+ 0*x_{21} + 0*x_{31} + -x_{12}/4 + 0*x_{22} + 0*x_{32} + 1.0*x_{13} + -x_{23}/4 + 0*x_{33} = 32/4 \\
0*x_{11} &+ 0*x_{21} + 0*x_{31} + 0*x_{12} + -x_{22}/4 + 0*x_{32} + -x_{13}/4 + 1.0*x_{23} + -x_{33}/4 = 24/4 \\
0*x_{11} &+ 0*x_{21} + 0*x_{31} + 0*x_{12} + 0*x_{22} + -x_{32}/4 + 0*x_{13} + -x_{23}/4 + 1.0*x_{33} = 56/4
\end{align*}
\]

If you compare the two sets of equations carefully, you will see that they are the same, but the second set is clearly more complicated. You may be wondering why go to the fuss. Well, if you remember your matrix multiplication, you can express that second set of equations as simply:

\[
\begin{array}{cccccccccc}
1.0 & -0.25 & 0.0 & -0.25 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & x_{11} = 4.0 \\
-0.25 & 1.0 & -0.25 & 0.0 & -0.25 & 0.0 & 0.0 & 0.0 & 0.0 & x_{21} = 5.0 \\
0.0 & -0.25 & 1.0 & 0.0 & 0.0 & -0.25 & 0.0 & 0.0 & 0.0 & x_{31} = 11.0 \\
-0.25 & 0.0 & 0.0 & 1.0 & -0.25 & 0.0 & -0.25 & 0.0 & 0.0 & x_{12} = 2.0 \\
0.0 & -0.25 & 0.0 & -0.25 & 1.0 & -0.25 & 0.0 & -0.25 & 0.0 & x_{22} = 0.0 \\
0.0 & 0.0 & -0.25 & 0.0 & -0.25 & 1.0 & 0.0 & 0.0 & -0.25 & x_{32} = 7.0 \\
0.0 & 0.0 & 0.0 & -0.25 & 0.0 & 0.0 & 1.0 & -0.25 & 0.0 & x_{13} = 8.0 \\
0.0 & 0.0 & 0.0 & 0.0 & -0.25 & 0.0 & -0.25 & 1.0 & -0.25 & x_{23} = 6.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.25 & 0.0 & -0.25 & 1.0 & x_{33} = 14.0
\end{array}
\]
The left-hand matrix consists only of the values needed to pick out the appropriate values of the coefficients. Thus that left-hand matrix will be always the same for any plate. It is the right-hand side that contains the constant values of the outer edge of the plate. Your program must compute that vector from the data values entered by the user.

In order to solve this system of equations, your program should initialize a matrix \( A \) and a vector \( B \) to values as just described. Two formats for commands to initialize \( A \) are shown below.

\[
A(1,:) = [ \text{values of first row} ]; \\
A(2,:) = [ \text{values of second row} ]; \\
\vdots \\
A(9,:) = [ \text{values of ninth row} ];
\]

\[
A = [ \text{values of first row}; \ldots \\
\text{values of second row}; \ldots \\
\vdots \\
\text{values of ninth row} ];
\]

The solution is then the result of the Matlab expression \( A \backslash B \).

**Optional (for fun only)**

One really neat visualization you can do is to animate the difference between the grid at each iteration of the iterative solution and the analytical solution. You should choose a colormap that has red for negative values, white in the middle for values around zero and blue for positive values. You can make a movie called frames with code that looks like this (assuming \( \text{iterations} \) is your variable for the # of iterations, and \( \text{threshold} \) is your threshold value):

```matlab
redwhitebluecmap=[1.00 0.00 0.00; 
                 1.00 0.25 0.25; 
                 1.00 0.50 0.50; 
                 1.00 0.75 0.75; 
                 1.00 1.00 1.00; 
                 1.00 1.00 1.00; 
                 0.75 0.75 1.00; 
                 0.50 0.50 1.00; 
                 0.25 0.25 1.00; 
                 0.00 0.00 1.00];

for i = 1:iterations+1  %% 3 iterations yields frame 1(initial),2,3,4
    imagesc( %%%YOUR-DIFFERENCE-GRID-HERE%%% ); %% Iteration-Analytical 
    colormap(redwhitebluecmap);        %% Set the colormap 
    axis square off equal;             %% Set the axes 
    caxis([-5*threshold,5*threshold]); %% Set colormap map (10 steps) 
    colorbar;                          %% Show a colorbar, add a title 
    title(sprintf('Temp grid convergence (iter-ans). Threshold %f, Iter-ation %d', threshold, i-1)); 
    frames(i) = getframe;              %% Save the frame 
end
```
You can then play your movie with the `movie` command (while it’s animating, feel free to stand and sing the national anthem if the colormap choice makes you feel overly patriotic):

```matlab
>> movie(frames)
```

**Miscellaneous requirements**

For full credit, you must divide this problem carefully into subproblems and represent them as subprograms. These will include a subroutine to fill the 9x9 coefficient matrix as shown above, and the 9-element right-hand-side vector based on the temperature values at the external points of the plate. You should also have a subroutine to compute the new temperatures at each step, and a logical function to determine if equilibrium has been reached. You won’t be able to store the new temperatures into the plate array immediately, since the averages for state \( t \) must be computed from the temperatures at state \( t-1 \).

In no case should you have a function that contains more than twenty-four lines of Matlab.

In this assignment you should assume temperatures measured in degrees Kelvin. Thus, negative temperatures are impossible (by the definition of the Kelvin scale and absolute zero). Do something reasonable with illegal data values.

As with earlier problems, test the subprograms separately, using data for which you can easily compute the correct answers by hand. Print out copies of your programs and test “drivers” so that tutors can verify that your individual routines work.

Make sure you can explain any difference between the solution values you produce from the two parts. If you make the difference criterion for part I a variable, you should be able to show that the differences between part I and II decrease for smaller differences between the averages.

You should strive to make your code as general as possible. As an example, you should not hardcode the 5x5 nature of the problem in your iterative solution. You may, however, hardcode it in your \( Ax = b \) analytical solution.

Use `format` statements to specify that no more than two decimal digits are to be printed for each value.
Checklist
A satisfactory grade on this program requires

correct results for simulation and for solution of systems of equations;
programs split into subprograms as specified:
  subroutine to fill the coefficient matrix;
  subroutine to update the simulation array;
  logical function to determine equilibrium;
subprograms as general as possible (e.g., no hardcoding of the 5x5 array sizes except in the Ax = b analytical solution);
evidence of individual testing of subprograms;
no routine longer than twenty-four lines of code;
variable and subprogram names that reflect their use;
systematic indentation of test and loop bodies;
comments at the head of the main program and subprograms that explain the purpose of each;
declaration of all variables in a “dictionary”;
negative temperatures detected and dealt with appropriately;
clean case analysis and simple loop structuring;
correct answers;
input of plate temperature values from a file;
nicely formatted output;
extplanation of differences between approximate and analytic solutions.
Quiz—Advanced array and matrix operations

Goals
This quiz tests you on Matlab operations for advanced creation and manipulation of vectors, matrices, and multidimensional matrices. Questions focus on matrix construction, manipulation, sorting, and searching. You will not be allowed to use loop or control constructs such as if, for, or while anywhere in this quiz.

Relevant functions include the following: linspace, reshape, repmat, meshgrid, rand, randperm, sort, logical, islogical, true, false, &, |, ~, any, all, isprime, mean, median, std, min, and max. Warning: you should be very familiar with all of these functions—study the help pages to make sure you know what each one does.

Readings
Mastering MATLAB 7, sections 5.8 through 5.12, chapter 6, sections 18.1 and 19.2.
Learning MATLAB 7, pages 4-26 through 4-27.
Sample questions for the “Advanced array and matrix operations” quiz

1. Give two different ways to construct a 1000-by-1000 matrix containing all zeros except for having ones all along the northeast-to-southwest diagonal. One solution should use eye and one should not.

   0 ... 0 1
   0 ... 1 0
   ... ... ... ...
   1 ... 0 0

2. Explain how to remove all of the non-perfect-square numbers from a matrix A. When you are done, the vector A should contain only perfect squares.

3. Explain how to construct a 5-by-5 matrix whose \([r,c]\) elements have the value \(z = (r^2 - c)\) for \(r\) evenly spaced between 0 and 4 and \(c\) evenly spaced between 1 and 2. Use linspace for specifying the rows and the : operator for specifying the columns.

4. Write an expression that returns true if all the elements in a matrix A are composite (not prime).

5. Suppose that vectors A and B contain the following values.

   \[
   \begin{align*}
   A &= [10 20 30 40] ; \\
   B &= [1 0 1 0] ;
   \end{align*}
   \]

   What is the value of A, B, and C after each expression below? Assume each expression is typed independently.

   \[
   \begin{align*}
   C &= A(B) ; \\
   C &= A(\neg B) ; \\
   C &= A(B \& A>20) ;
   \end{align*}
   \]
Solutions to sample questions for the “Introduction to array and matrix operations” quiz

1.

Three solutions that use eye

- `flipud(eye(1000))`
- `fliplr(eye(1000))`
- `rot90(eye(1000))`

Two solutions that don’t use eye

- `A=reshape(0:1000*1000-1,1000,1000)';`
- `A=(999*fix(A/999)==A);`
- `A(1,1)=0;`  
  `A(1000,1000)=0`  
  `A=zeros(1,998)';`  
  `B=[0 repmat(A,1,1000) zeros(1,999)];`  
  `reshape(B,1000,1000)`

2.

`A(fix(sqrt(A)).*fix(sqrt(A))~=A)=[]`

3.

`[r,c]=meshgrid(0:4,linspace(1,2,5));`
`r.*r-c`

4.

`all(~isprime(A))`

5.

```matlab
C = A(B);
A [10 20 30 40] unchanged
B [1 0 1 0] unchanged
C ??? Undefined function or variable 'C'
The expression caused an error! Even though B looks like a vector of logicals, it is in fact a vector of doubles, since that is the default type.

C = A(~B);
A [10 20 30 40] unchanged
B [1 0 1 0] unchanged
C [20 40]
The act of negating the double vector B converts it to a logical vector that can then be used with logical subscripting.

C = A(B & A>20);
A [10 20 30 40] unchanged
B [1 0 1 0] unchanged
C 30
ANDing a double vector B with a logical vector—the result of the > operator—produces a logical vector that can then be used with logical subscripting.
```
Program—Image manipulation

The goal of this project is to write a program to collect data from human subject perception experiments and to perform a graphical analysis of this data. In the process, you will gain experience with structures, function pointers, a Graphical User Interface (GUI), mouse events, and rudimentary image processing. Whew!

Readings

Mastering MATLAB 7, chapters 8 and 9; sections 12.7, 16.4, 26.6-26.8; chapter 28; and sections 29.1 through 29.3.

Learning MATLAB 7, pages 5-58, 5-59, 6-10 through 6-17, 6-26, and 6-27.

Related quizzes

Plotting and function handles; cell arrays and structures.

Programming assignment

Do the programming assignment described on the following pages.
CS 9A programming assignment
Image manipulation

Background

Researchers have used interactive computer experiments to test everything about people from color blindness to visual blind spots to response times under sleep deprivation. These experiments typically flash something on the screen and ask that the user indicate the moment they saw something change by clicking the mouse in the center of the region that changed. If everything is kept constant except for the small change in the display, one can measure how sensitive the user is to the particular type of change. You’re going to author one of these experiments. The really cool thing about this assignment is that once you’ve finished, you’ll have a template that will let you measure yourself against your friends in terms of response times and accuracy, as well as learn what kinds of visual changes are hardest to detect.

Problem

Your goal is to write software to allow you to run experiments that check a human subject’s ability to notice small distortions in grayscale images. In each trial, a small square window of an image will be distorted using one of a small number of distortion functions. The subject is shown this modified image (sometimes following the original image) and asked to click with the mouse at the center of the distortion patch. Your program will have the ability to vary the size of the patch as well as the distortion function; it will run several trials for each combination to see how these affect the results.

The main function for running the experiment is provided for you:

```matlab
result = RunExperiment(myImage, Functs, sizes, ntrials, show_orig)
```

RunExperiment is given an image, a cell array of function handles (one for each distortion), an array of sizes, the number of trials per function/size combination to run, and
a boolean (0 or 1) determining whether to show the original before the distorted image. It calls a function to generate the random order of the trials, distorts a randomly positioned window of the image and calls a function named TimeUser to show the image and time the subject’s response. The last parameter to RunExperiment, show_orig, is passed on to TimeUser.

RunExperiment returns a structure named result whose fields include the following:
- sizes, Funcs, and numTrials (supplied in the call to RunExperiment);
- size, func, clickPosition, elapsedTime, and windowPos (computed in each experiment and passed on to ConvertResults, a function you write to analyze the results).

For this assignment, you will provide several sets of functions:
- a function named GenerateTrials to generate a matrix that contains information about the trials;
- several distortion functions that replace pixels in a portion of the image;
- a function named TimeUser that runs a single trial, displaying an image and recording the user’s mouse click and reaction time;
- analysis functions that allow the results to be easily interpreted.

The GenerateTrials function

GenerateTrials takes three arguments: the number of distortion functions, the number of sizes, and the number of trials to run. It returns an n-by-2 matrix whose rows have the form [function_id size_id] and collectively contain numTrials copies of all [function_id size_id] pairs, appearing in random order. (Thus n = numSizes*numFunctions*numTrials.). The Matlab function repmat, meshgrid, and randperm will be useful for generating this matrix.

Here are two example calls. The first asks GenerateTrials to pair up 2 functions with 3 sizes with each combination represented exactly once. The second call wants each of these combinations represented exactly twice (numTrials = 2):

```
>> GenerateTrials(2,3,1)
ans =
   2   3
   2   1
   1   2
   2   2
   1   3
   1   1
>> GenerateTrials(2,3,2)
ans =
   2   3
   2   2
   2   3
   1   3
   1   3
   2   1
```
**Distortion functions**
Each distortion function will be called as follows, taking an image as argument and returning an image of the same size.

```
t_im = ImTrans___(im);
```

The blank in the function name should be replaced by one of the following:

- **Control.** Replace the pixels with all black or all white, depending on the mean color of the patch: if the patch is light, replace with black, otherwise with white. This should be very easy to spot, and should serve as a baseline, or *control* for the experiment.

- **Mean.** Replace the pixels with the mean color of the patch.

- **Invert.** Figure out the range of intensity values within the patch. Within this range, invert the intensity values. That is, the pixel that was the brightest before will get the intensity of the darkest, and vice versa.

- **Transpose.** Take the transpose of the square patch (i.e. flip along diagonal).

You may also provide extra distortion functions besides those in the list above.

**The TimeUser function**
The TimeUser function is called as follows.

```
[clickPosition, elapsedTime] = ...
    TimeUser(distortedImage, originalImage, show_orig)
```

TimeUser clears the figure or displays the originalImage (based on the show_orig flag), waits 1 second, then shows distortedImage and waits till the user clicks somewhere on the image with the mouse. When the click arrives, it records the [x y] position in clickPosition and the elapsed time in elapsedTime, and returns them. Use the same figure all the time; popping up a new figure each time gets really annoying for the user. Be sure to call axis off image after you call imagesc so that the axis tic marks are suppressed and the image has the correct aspect ratio. The Matlab functions imagesc, ginput, tic, and toc will be useful for writing TimeUser.

**Analysis**
The result structure returned by the experiment is not in the most convenient format for plotting. Write a function ConvertResults:

```
[tmatrix, dmatrix] = ConvertResults(res)
```

that takes in the res struct and returns tmatrix, containing elapsed-time information, and dmatrix, containing distances between the mouse click points and the rectangle centers. Each should be an S-by-F-by-N matrix, where S = the number of sizes, F = the number of distortion functions, and N = the number of trials. For example, tmatrix(2,3,4) is the time elapsed between showing the image and the mouse click for the second size value and third function type during the fourth trial. (This conversion can be performed without any loops, using the Matlab functions sub2ind, sort, permute and reshape!)
Now call a final helper routine DisplayAnalysis to do whatever plotting (error bars, scatters, 3-D plots, etc.) you feel appropriate to highlight the results in the most revealing way. You should declare a cell array to record the names of your functions (in the same order as they were listed in your cell array of functions you passed to RunExperiment); pass these names to DisplayAnalysis to use to label your charts.

Based on your data (collected from at least two different input images, one of random grey values and one of your choosing), devise and justify a model of what type of transformations are more or less difficult to notice. How did the input image affect the results? How does the size affect the results? Is the task easier if the original image is shown right before the distorted image?

**Miscellaneous requirements**

When you are in your last stage and ready to begin running your experiments, resize the “Figure 1” window to be larger (in raw pixel dimensions) than the original image. Fix that size for the duration of your tests.

You should comment your code appropriately and save it in M-files.

Show your tutor a printout of several plots with many different test scenarios (described below), and a printed listing of your M-file.

Run your experiment on at least three different volunteers and collect anonymized relevant information about them (age, gender, eyesight, years using a mouse, etc).
Checklist

A satisfactory grade on this program requires

- completion of the GenerateTrials, TimeUser, ConvertResults, and DisplayAnalysis functions, a main “driver” function that sets up your experiment, and any subroutines you found handy
- evidence of individual testing of all functions
- no routine longer than twenty-four lines of code
- variable and subprogram names that reflect their use
- comments at the head of all your functions that explain the purpose of each declaration of all variables in a “dictionary”
- printouts of the two original images you used, one of your own choosing and one a random grayscale image (in which each pixel takes a random value in a given range, say [0,1] or [0,255]).
- an explanation of why you chose the visualization you did (error bars, scatters, 3-D plots, etc.)
- printed output of your plots on the following test scenarios:
  - two images, one of your own choosing and one a random image
  - all four ImTrans___ functions
  - several different sizes of distortion patches
  - several different trials
- an answer to the questions posed in the “Analysis” section above.
runExperiment.m

function runExperiment (myImage, Funcs, windowSizes, numTrials)
    [times dists] = convertResults (dataGathered (myImage.X, Funcs, ...
        windowSizes, numTrials))
    
    Plot the times.
    
    Plot the distances.

dataGathered.m

function result = dataGathered (myImage, Funcs, windowSizes, numTrials)
    
    Runs the specified trials per function per window size of the experiment,
    
    returning a struct as described above.
    
    Arguments are
    
    myImage - image data array to use
    Funcs - cell array of function handles to distortion functions
    windowSizes - vector of window sizes
    numTrials - number of trials to run per function and window size
    
    trials = genTrials (Funcs, windowSizes, numTrials);
    figure (1);
    imagesc (myImage);
    colormap (gray);
    pause (5);
    result.windowSizes = windowSizes;
    result.Funcs = Funcs;
    result.numTrials = numTrials;
    s = size(myImage);
    for i = 1:size(trials,1)
        windowSize = [1 1] * windowSizes (trials (i,2));
        windowLowerLeft = round (rand (1,2) .* (s(1:2)-windowSize+1));
        distortedImage = myImage;
        % Extract subimage from myImage.
        subimage = myImage (windowLowerLeft(1)+(0:windowSize(1)-1), ...
            windowLowerLeft(2)+(0:windowSize(2)-1));
        distortedSubimage = _____ ;
        % Change subimage in distortedImage.
        distortedImage (windowLowerLeft(1)+(0:windowSize(1)-1), ...
            windowLowerLeft(2)+(0:windowSize(2)-1)) = distortedSubimage;
        [result.clickPosition(i,:), result.elapsedTime(i)] = ...
            timeUser(distortedImage, myImage);
        result.windowSize (i) = trials (i,2);
        result.func (i) = trials (i,1);
        result.windowPos (i,:) = [windowLowerLeft];
    end
Quiz—Plotting and function handles

Goals
This quiz tests you on use of the plot function and related graphing features, and on creation, storage, and use of function handles. For this quiz, we will focus on basic uses of plot:

• What is the effect of a given call to plot?
• What call to plot has the following effect?
• What pops a new window up?
• How do you produce several graphs in the same window?

The important aspects of function handles are their creation with @ and possibly with inline, and their application.

Readings
Mastering MATLAB 7, sections 12.7 and 26.1 through 26.7.
Learning MATLAB 7, pages 5-5-38 through 5-50 and 6-27.
Sample questions for the “Plotting and function handles” quiz

1. Define a cell array named `functs` that contains the two inline functions

\[
\begin{align*}
  f_1(x) &= 2 + x \\
  f_2(x) &= 6 - x
\end{align*}
\]

Do this completely within the Matlab command window; don’t assume that M-files exist for these functions, and don’t create the M-files yourself.

2. Give a plot command that graphs \( f_1 \) and \( f_2 \) applied to \( x = 2, 3, \) and \( 4 \). The desired output is shown below.

![Plot of functions](image)

3. Explain how you would create a plot of another function while still retaining the graph you created for exercise 2.

4. Draw the graph that is plotted by the call

```matlab
plot ([4 6 8; 4 3 2])
```

5. Write a function named `plotall` (to be included in an M-file) that takes as arguments a cell array of handles of 1-argument functions and a vector of arguments to which the functions are to be applied, and graphs the functions over the given vector. For example, if `functs` is your answer to exercise 1, the call

```matlab
plotall (functs, 2:4)
```

should produce the plot displayed in exercise 2.
Solutions to sample questions for the “Plotting and function handles” quiz

1.  \[
    \text{functs} = \{\text{inline}(’2+x’,’x’), \text{inline}(’6-x’,’x’)\}
    \]

2.  \[
    \text{plot(2:4,feval(f1,2:4),2:4,feval(f2,2:4))}
    \]

3.  Call \texttt{figure(2)} before plotting the second graph. (This assumes that your first graph is in figure 1.)

4.  When a matrix \(M\) is given to plot, one graph is plotted for each column in \(M\). The values in the column are the \(y\) values, and the \(x\) values are \(1:\#\text{rows}\). Thus for the matrix
    \[
    \begin{array}{ccc}
    4 & 6 & 8 \\
    4 & 3 & 2 \\
    \end{array}
    \]

    the points \((1,4)\) and \((2,4)\) appear in one graph, \((1,6)\) and \((2,3)\) appear in another, and \((1,8)\) and \((2,2)\) appear in another, giving

![](image)

5.  \[
    \text{function plotall (fs, vals);} \text{\hspace{1cm}}
    \text{hold on}\text{\hspace{1cm}}
    \text{for i=1:size(fs)}\text{\hspace{1cm}}
    \text{plot(vals,feval(fs{i},vals));}\text{\hspace{1cm}}
    \text{end}\text{\hspace{1cm}}
    \text{hold off}
    \]
Quiz—Cell arrays and structures

Goals
This quiz tests you on procedures for creating and accessing cell arrays, structures, structure arrays, and character strings. The conversion functions cell2struct and struct2cell will also be tested.

Readings
*Mastering MATLAB 7*, chapters 8 and 9.
*Learning MATLAB 7*, pages 6-8 through 6-17.

Sample questions for the “Cell arrays and structures” quiz

1. Create a structure named `s` with three fields: a character string named `courseName` with value “cs9a”, an integer named `unitCount` with value 1, and a 1-row matrix named `scores` whose elements are 33, 35, and 77.

2. Create a structure array named `courses` whose first element is `s` and whose second element has `courseName` = “cs9b”, `unitCount` = 1, and `scores` a matrix with the four elements 30, 40, 23, and 35.

3. Suppose the `courses` array has been correctly constructed. What's the result of typing each of the following to the MATLAB interpreter?
   
   ```matlab
   courses{1} % using braces
   courses(1) % using parentheses
   courses{1} % using brackets
   ```

4. In one expression, construct a 2x3 cell array named `A` whose first row contains the character string “abc”, a 2x2 matrix of all 1’s, and a 1x3 matrix containing 5, 3, and 1, and whose second row contains a 7, the character string “hello there”, and the structure array `s`.

5. In one expression (using the `max` function), find the largest element of the matrix in the last element of the first row of `A`. 
Solutions to sample questions for the “Cell arrays and structures” quiz

1. Either

s.courseName = 'cs9a'
s.unitCount = 1
s.scores = [33 35 37]

or

s = cell2struct ({{'cs9a' 1 [33 35 37]}});
        {'courseName' 'unitCount' 'scores'}, 2)

The 2 is necessary to specify that each column of the cell array is to be converted to a structure field.

2. Either

s2.courseName = 'cs9b'
s2.unitCount = 1
s2.scores = [30 40 23 35]
courses = [s s2]

or

courses(1) = s
courses(2).courseName = 'cs9b'
courses(2).unitCount = 1
courses(2).scores = [30 40 23 35]

3. Braces are used with cell arrays; courses is a structure array. Thus an error message results: “Cell contents reference from a non-cell array object.”

   courses(1) accesses the first structure in the array.

   Brackets are used to construct matrices. Thus an error message results: “Unbalanced or misused parentheses or brackets.”

4. 

A={'abc' [1 1; 1 1] [5 3 1]; 7 'hello there' courses}

5. 

max(a{1,3})
Program—Symbolic mathematics

This programming assignment introduces you to MATLAB’s Symbolic Math Toolbox.

Readings

*Mastering MATLAB 7* does not contain material on this topic.

*Learning MATLAB 7*, chapters 9 and 10.

“Symbolic Math Toolbox: Printable Documentation”, accessible from the Help window in your MATLAB programming environment.

Background

This assignment has been partially adapted from lab exercises designed by Professor Eric Woolgar at the University of Alberta.

Here are some examples of relevant MATLAB commands

\[
\begin{align*}
x &= \text{sym}('x') & \text{defines } x \text{ to be a symbolic object that can then be treated as a mathematical variable.} \\
syms ('a', 'b', 'c', 'd') & \text{creates several symbolic objects at once; syms('a') is the same as } a = \text{sym('a')}.
\end{align*}
\]

\[\text{syms } a \ b \ c \ d \] same as \[\text{syms ('a', 'b', 'c', 'd')}\].

\[y = \text{sym}('1/2')\] creates the symbolic object \(y\) and then gives it the definite value \(1/2\), so \(y\) is treated as a mathematical constant.

\[M = [a \ b; \ c \ d]\] creates a symbolic matrix, if \(a\), \(b\), \(c\), and \(d\) are symbolic objects.

\[\text{ezplot (y, [a b])}\] plots the symbolic expression \(y\) (given as a function of the symbolic object \(x\)) on the interval from \(a\) to \(b\).

\[\text{factor (y)}\] factors the symbolic expression \(y\) (given as a function of the symbolic object \(x\)).

\[S = \text{solve (eqn1, eqn2, ...)}\] solves a system of equations in symbolic variables \(x1, x2, ...\) and stores the solution in \(S\).

\[S.x1\] returns the value that \(x1\) takes in the solution \(S\).

\[\text{simplify (S.x1)}\] simplifies the expression for \(x1\).

Related quizzes

Symbolic mathematics.

Programming assignment

Do all of the exercises described on the following pages.
Symbolic root finding

1. Plot the polynomial \( y = x^3 - 3x + 2 \). Factor \( y \) and find all values of \( x \) such that \( y = 0 \). Remember when defining \( y \) to first define \( x \) as symbolic. Also remember that every multiplication must be represented by an asterisk (*) and that exponents are preceded by the hat (^) symbol.

Solving systems of equations

To solve an equation or a system of equations in the Symbolic Toolbox, rewrite each equation if necessary so that the right-hand side is 0. Give the left-hand side of each equation a name. For single equations, apply the `solve( )` command directly to this name. For systems of equations, first apply the `solve( )` command and then ask for the value of each unknown variable in turn.

For example, to solve \( x^2 - 3x + 1 = 5 \), we can rewrite it as \( x^2 - 3x - 4 = 0 \). Then:

```matlab
syms x
eqn = x^2 - 3*x - 4
solve (eqn)
```

Try it yourself. Notice you get two roots, of course.

Next, let’s try the inhomogeneous system

\[
\begin{align*}
x_1 + 2x_2 &= 3 \\
4x_1 + x_2 &= -1
\end{align*}
\]

Now the syntax is more complicated:

```matlab
syms x1 x2
eqn1 = x1 + 2*x2 - 3
eqn2 = 4*x1 + x2 + 2
S = solve (eqn1, eqn2) % You may use any symbol you prefer instead of S.
S.x1 % This returns the value of x1 in the solution.
S.x2 % This returns the value of x2 in the solution.
```

2. Use the Symbolic Toolbox to solve the system

\[
\begin{align*}
x_1 + 2x_2 + 2x_3 &= 0 \\
2x_1 + 3x_2 + 4x_3 &= 1 \\
-x_1 + x_2 + x_3 &= -6
\end{align*}
\]
Working with symbolic matrices

This exercise involves what Knuth (The Art of Computer Programming, Volume 1: Fundamental Algorithms) refers to as the \( n \times n \) combinatorial matrix:

\[
\begin{bmatrix}
x + y & y & \cdots & y \\
y & x + y & \cdots & y \\
\cdots & \cdots & \cdots & \cdots \\
y & y & \cdots & x + y
\end{bmatrix}
\]

All of its elements are \( y \) except those on the main diagonal, which are \( x+y \).

3. Find an expression for the sum of all \( n^2 \) elements of the inverse of the combinatorial matrix of size \( n \). (Hint: it’s a simple expression in \( n \), \( x \), and \( y \).) To do this, use functions in the Symbolic Toolbox and regular matrix functions to construct matrices of sizes 3, 4, 5, … and infer a pattern.

Checklist

A printed sequence of Matlab commands for each of the three exercises that produces a solution to the exercise.
Quiz—Symbolic math operations

Goals
This quiz tests you on Matlab’s facility for *symbolic* math.

Readings
*Mastering MATLAB 7* does not contain material on this topic.
*Learning MATLAB 7*, chapters 9 and 10.

Sample questions for the “Symbolic math operations” quiz
1. How would you find out the symbolic value of $y=ax^2+bx+2^{1/2}$ when $x=2$?
2. Where does the curve $y=ax^2+bx+2^{1/2}$ intersect the line $y=2^{1/2}$?
3. Build an $n$-by-2 infinite precision table of the counting numbers and their inverses. The first column should be increasing values of $n$ (starting from 1) and the second column should be values of $1/n$. Here is the answer for $n=3$:

   \[
   \text{ans} =
   \begin{bmatrix}
   1 & 1 \\
   1 & 1/2 \\
   1 & 1/3
   \end{bmatrix}
   \]
Solutions to sample questions for the “Symbolic math operations” quiz

1. 
   syms x y a b           % Declare symbolic variables  
   f = a*x^2 + b*x + sqrt(2);  % Declare symbolic function 
   subs(f,2)               % Evaluate at x (default) = 2 
   ans = 
   4*a+2*b+c

2. 
   solve(f-sqrt(2)) % y=f(x)=2^1/2 implies that f(x)-2^1/2 = 0  
   ans =  
   0  
   -b/a

3. 
   % n is defined already  
   n_ones = ones(n,1)       % Generate a column of ones  
   [ n_ones n_ones ./ sym([1:n]') ]       % [1 1/n]
Notes from tutoring sessions
Notes from tutoring sessions
Notes from tutoring sessions
Notes from tutoring sessions
Notes from tutoring sessions