

MATLAB OVERVIEW
=====

WHAT IS MATLAB?

MATLAB is a software package available via your Leland account. You can also buy a student or professional version for your PC or MAC. MATLAB is designed for the manipulation and visualization of matrices, and analysis of large amounts of data.

HOW DO I START MATLAB?

To use MATLAB, first log into your Leland account using either `bramble.stanford.edu` or `hedge.stanford.edu`. Not only do they run MATLAB much faster than `cardinal`, but if you log on to `cardinal.stanford.edu` you may be knocked off of MATLAB. Once you have logged on to your Leland account and your `msande351` directory, type "matlab", e.g.,

```
bramble2:~/msande351> matlab
```

MATLAB responds with:

```
      < M A T L A B >  
      Copyright 1984-2007 The MathWorks, Inc.  
      Version 7.5.0.338 (R2007b)  
      August 9, 2007
```

To get started, type one of these: `helpwin`, `helpdesk`, or `demo`.
For product information, visit www.mathworks.com.

```
>>
```

The above `>>` is the MATLAB prompt. A MATLAB command is executed by typing it at this prompt and then striking the ENTER key. What follows is the MATLAB reply. In the sequel the lines with `>>` are command lines and the rest are MATLAB replies.

HOW DO I QUIT?

Type "quit" from the command line, e.g.,

```
>> quit
```

```
bramble2:~/msande351>
```

NAVIGATING MATLAB

Typing "help general" will give information on general purpose commands useful for navigating MATLAB. UNIX shell commands may also be executed by putting a ! before the command. For example, to edit a file "words.txt" using EMACS while running MATLAB type "!emacs words.txt".

MATRICES

MATLAB was originally written as a matrix manipulation program, and therefore tends to try to deal with everything as a matrix. Although it is possible to input equations, assign variables, and use a lot of mathematical functions, to make efficient use of MATLAB, it is necessary to use matrices. To enter a small matrix, type

```
>> [1 2 3; 4 5 6]
```

with rows separated by semicolons and MATLAB responds with

```
ans =  
     1     2     3  
     4     5     6
```

This matrix can also be entered in its usual form with carriage returns replacing the semicolons.

```
>> [1  2  3  
    4  5  6]
```

```
ans =  
     1     2     3  
     4     5     6
```

When entering a matrix, elements are separated by a space, rows are separated by a semicolon and the matrix is enclosed in square brackets.

SUPPRESSING OUTPUT

After typing a MATLAB command, the output of that command is displayed. To execute a command and not display the output, follow the command with a semicolon.

```
>> [1  2  3  
    4  5  6];
```

VARIABLES and WORKSPACE

When you create a matrix, you can assign it a variable consisting of characters and numbers--the first a character, e.g., r2D2, x32, Y5, A, etc. To display the value of a variable, type its name.

```
>> A = [1  2  3  
        4  5  6];
```

```
>> A
```

```
A =  
     1     2     3  
     4     5     6
```

MATLAB stores variables in an initially empty workspace. Once a variable has been assigned it is stored in the workspace and may be referred to again until you redefine or "clear" it. For example

```
>> clear VAR1 VAR2 ....
```

removes the variables VAR1, VAR2, ... from the workspace. To get a list of variables in the workspace type "who".

```
>> who
```

Your variables are:

```
A
```

Type "whos" for a more detailed description

```
>> whos
```

Name	Size	Bytes	Class
A	2x3	48	double array

Grand total is 6 elements using 48 bytes

VECTOR FUNCTIONS

EQUALLY SPACED ELEMENTS. The colon `:` operation can be used to produce vectors with equally spaced elements, e.g., with unit increments as in

```
>> 3:7
```

```
ans =  
    3    4    5    6    7
```

or with arbitrary increments, say, in increments of `-2`, as in

```
>> 7:-2:3
```

```
ans =  
    7    5    3
```

CONVOLUTION. If `A` and `B` are vectors, `conv(A,B)` returns their convolution.

```
>> A = 1:3 ;
```

```
>> B = 4:5 ;
```

```
>> conv(A,B)
```

```
ans =  
    4   13   22   15
```

See the section MORE on CONVOLUTIONS near the end of this document for more details.

MATRIX OPERATIONS AND FUNCTIONS, AND SOLVING MATRIX EQUATIONS

TRANSPOSE. `A'` returns the transpose of a matrix `A`.

SCALAR OPERATIONS. MATLAB supports scalar addition and multiplication of matrices. For example, if `A` is a matrix, then `A+2` (resp., `A*2`) returns the matrix formed by adding (resp., multiplying) each element of `A` to (resp., by) `2`.

MATRIX OPERATIONS. MATLAB supports the usual matrix operations, viz., `+`, `-` and `*`, the last being ordinary matrix multiplication.

DIVISION AND SOLVING MATRIX EQUATIONS. If `A` and `B` are matrices with the same number of rows, then `A \ B` returns the solution `X` of the matrix linear equations $AX = B$. Similarly, if `A` and `B` are matrices with the same number of columns, then `B / A` returns the solution `Y` of the matrix linear equations $YA = B$. MATLAB uses different techniques for solving linear equations depending on the shape of `A`. If `A` is square and singular, MATLAB produces an error message even when the system $AX = B$ has a solution. The workaround for this is to add a dummy row of zeros to `A` and to `B`. MATLAB will solve the resulting system. For example, suppose

```
>> A = [.6 .4  
        .4 .6];  
  
>> B = [ 1  1].
```

The solutions of the equations $AX = B'$ and $YA = B$ are

```
>> X = A \ B'  
  
X =  
    1.0000  
    1.0000  
  
>> Y = B / A  
  
Y =  
    1.0000    1.0000
```

But MATLAB fails to solve the system $AX = B$ below even though it has a solution.

```
>> A = [0  0  
        1  0];  
  
>> B = [0  1]';  
  
>> X = A \ B
```

Warning: Matrix is singular to working precision.

```
X =  
    NaN  
    NaN
```

The workaround is to append a row of zeros to A and B. Then MATLAB solves the equivalent resulting system, but warns that the matrix has rank 1.

```
>> A = [A  
        0  0];  
  
>> B = [B  
        0];  
  
>> X = A \ B
```

Warning: Rank deficient, rank = 1 tol = 6.6613e-016.

```
X =  
    1  
    0
```

POWERS. Use \wedge for powers, e.g., $2^3 = 8$, $4^{.5} = 2$. This works for square matrices too. For example, if A is a square matrix, A^3 returns the cube $A*A*A$ of A and $A^{0.5}$ returns the square root of A, i.e., the matrix whose square is A. If A is also non-singular, this works for negative integer powers of A too, e.g., $A^{(-2)}$ returns the square of the inverse of A. Alternately, $\text{inv}(A)$ also returns the inverse of A.

MAX and MIN. If A is a vector, $\text{max}(A)$ returns the largest element of A. If A is a matrix, $M = \text{max}(A)$ returns the row vector M containing the maximum element in each column of A. Also, $[M,I] = \text{max}(A)$ returns both M and the row indices I of the maximum elements. If there are several such indices, the index is the smallest one.

```
>> A = [1 3 0  
        0 4 0];
```

```
>> [M,I] = max(A)
```

```
M =  
    1    4    0
```

```
I =  
    1    2    1
```

The function `min(A)` works analogously after replacing `max` by `min` everywhere.

`SUM` and `CUMSUM`. If `A` is a vector, `sum(A)` returns the sum of the elements of `A` and `cumsum(A)` returns the partial sums of elements of `A`. If `A` is a matrix, `sum(A)` returns the row vector that is the column sums of `A` and `cumsum(A)` returns the matrix each of whose columns is the column-vector of partial sums of the corresponding column of `A`.

```
>> A = [ 1 2  
        3 4];
```

```
>> sum(A)
```

```
ans =  
     4     6
```

```
>> cumsum(A)
```

```
ans =  
     1     2  
     4     6
```

`PRODUCT`. If `A` is a vector, `prod(A)` returns the product of the elements of `A`. If `A` is a matrix, `prod(A)` returns the row vector of products of the elements in each column.

```
>> prod(A)
```

```
ans =  
     3     8
```

`ARRAY OPERATIONS`. An array operation, i.e., an operation that is performed element-wise, is done by inserting a `."` in front of the operation (only for `*`, `^`, `/` and `\`), e.g.,

```
>> [1 2 3] .^ 3
```

```
ans =  
     1     8    27
```

```
>> [1 2 3] .* [0 -1 2]
```

```
ans =  
     0    -2     6
```

`SPECIAL MATRICES` (`EMPTY`, `IDENTITY`, `ZEROS`, `ONES`, `RANDOM`).

The matrix `A = []` is empty.

The function `eye(n)` returns the identity matrix of order `n`.

The function `zeros(m,n)` returns the $m \times n$ matrix of zeros.
The function `zeros(n)` returns the $n \times n$ matrix of zeros.
The function `ones(m,n)` returns the $m \times n$ matrix of ones
The function `ones(n)` returns the $n \times n$ matrix of ones.
The function `rand(m,n)` returns an $m \times n$ matrix of uniform random numbers in $(0,1)$.
The function `randn(m,n)` returns an $m \times n$ matrix of standard normal random numbers.

DUPLICATING COLUMNS. Often a column vector will need to be subtracted from each column of a matrix. Rather than use a loop, use a row vector of ones to create a matrix with the column vector duplicated in each of its columns. Then subtract this matrix from the original matrix.

For example, if you want to subtract

```
>> v = [2
        5
        7];
```

from the matrix

```
>> M = [1  2  3
        4  5  6
        7  8  9];
```

use the command

```
>> M - v * ones(1,3)
```

```
ans =
    -1     0     1
    -1     0     1
     0     1     2
```

RESHAPING A MATRIX INTO A COLUMN. You can reshape a matrix into a column with the operator `:.` Given the matrix `A`, `A(:)` is a column vector with the columns of `A` stacked in order on top of one another, e.g.,

```
>> A = [1  3
        2  4];
```

```
>> A(:)
```

```
ans =
     1
     2
     3
     4
```

FLIP A MATRIX LEFT AND RIGHT. You can reverse the order of the columns of a matrix `A` with the function `fliplr(A)`, e.g.,

```
>> A = [1  3
        2  4];
```

```
>> fliplr(A)
```

```
ans =
     3     1
     4     2
```

MATRIX of MATRICES. Here are some examples of matrices of matrices.

```
>> A = [1 2 3; 4 5 6; 7 8 9];
```

```
>> b = [10 11 12];
```

```
>> [A  
    b]
```

```
ans =  
     1     2     3  
     4     5     6  
     7     8     9  
    10    11    12
```

```
>> [A b']
```

```
ans =  
     1     2     3    10  
     4     5     6    11  
     7     8     9    12
```

```
>> [A b'  
    b 1]
```

```
ans =  
     1     2     3    10  
     4     5     6    11  
     7     8     9    12  
    10    11    12     1
```

HELP and LIST OF MATRIX FUNCTIONS THAT MATLAB SUPPORTS

Typing "help" gives a list of all the areas for which there are built-in functions. One of these areas is matlab/matfun - Matrix functions - numerical linear algebra. Typing

```
>> help matlab/matfun
```

gives a list of the matrix functions that MATLAB supports. Here are some examples.

MORE MATRIX FUNCTIONS

```
norm(A) - Matrix or vector norm of A.  
rank(A) - Matrix rank of A.  
det(A) - Determinant of a square matrix A.  
expm(A) - Matrix exponential of a square matrix A  
eig(A) - Eigenvalues of a square matrix A  
.  
.
```

To get information about the function rank, type

```
>> help rank
```

RANK Matrix rank.

RANK(A) provides an estimate of the number of linearly independent rows or columns of a matrix A.

RANK(A,tol) is the number of singular values of A that are larger than tol.

RANK(A) uses the default $\text{tol} = \max(\text{size}(A)) * \text{norm}(A) * \text{eps}$.

In general to find out about the predefined function or command "name", type

```
>> help name
```

FORMAT COMMAND

The FORMAT command allows you to format the output, e.g., display numbers as two-digit decimals or as fractions. To see what the options are, type

```
>> help format
```

RELATIONAL OPERATORS

MATLAB supports six relational operators:

```
< Less than  
<= Less than or equal to  
> Greater than  
>= Greater than or equal to  
== Equal to  
~= Not equal to
```

A relational operator returns one if its argument is true and zero otherwise, and operates element-by-element on a matrix. Here is an example.

```
>> A = [2 4  
        1 5];
```

```
>> B = [5 4;  
        2 3];
```

```
>> (A <= B)
```

```
ans =  
     1     1  
     1     0
```

EXTRACTING AND MODIFYING SUBMATRICES OF A MATRIX. To access the (i,j) element of a matrix A, type "A(i,j)", e.g.,

```
>> A = [1 2 3  
        4 5 6  
        7 8 9];
```

```
>> A(2,3)
```

```
ans =  
     6
```

Submatrices can be extracted using vectors, e.g.,

```
>> u = 1:2 ;
```

```
>> v = [1 3];
```

```
>> A(u,v)
```

```
ans =  
     1     3  
     4     6
```

gives the submatrix defined by the first two rows and the first and third columns. An entire row or column can be selected by using a colon : instead of an index or a vector. For example, to find the second column of A, type

```
>> A(:,2)
```

```
ans =  
    2  
    5  
    8
```

Submatrices can be extracted and modified using relational operators. If you modify a matrix, the entire modified matrix displays, e.g.,

```
>> x = [5 0 -4];
```

```
>> x(x < 0) = -x(x < 0)
```

```
x =  
    [5 0 4]
```

CONTROL EXPRESSIONS AND LOOPING

MATLAB supports if, for and while statements. The following examples show how to use these expressions

```
if x == 0  
    x = x + 1;  
elseif x == 1  
    x = x - 1;  
else  
    x = 0;  
end
```

```
for i = 1:10  
    j = 10 * i;  
end
```

```
i = 0;  
while i < 10  
    i = i + 1;  
end
```

AVOID LOOPS--ESPECIALLY NESTED LOOPS--VECTORIZE FOR SPEED!

MATLAB is designed for matrices. For this reason, it is MUCH FASTER to use matrices than loops. For example, if A is the 500 x 500 matrix of 2's, the matrix of its square roots can be found with the program.

```
>> for i = 1:500  
    for j = 1:500  
        A(i,j) = A(i,j) ^ .5;  
    end  
end
```

But the following computes these square roots much faster.

```
>> A = A .^ .5;
```

GRAPHS AND PLOTS

MATLAB supports 2-D and 3-D plots. To illustrate the former, let y be an $m \times n$ matrix. Then `plot(y)` graphs the n columns of y versus their row indices $1, \dots, m$.

Suppose also that x is an m -vector. Then `plot(x,y)` graphs the columns of y versus x .

```
>> x = -5:5;  
>> plot(x,x.^2)
```

plots the square of the elements of x versus x .

Titles, axes, axis labels, etc., of graphs can be altered with built-in functions.

There are two areas, `matlab/graph2d` and `matlab/graph3d`, containing built-in functions for graphs.

It is necessary to "activate" a figure to print it. If you are running MATLAB in a windows environment, click the figure with your mouse. If you are running MATLAB in a command-line environment and wish to activate figure 2 say, type

```
>> figure(2)
```

m-FILES (MATLAB PROGRAMS)

You can write MATLAB "programs" using m-files that are text files with the extension `.m`.

CASE of FILE NAMES. Though the case (upper or lower) of characters in file names is ignored by the standard PC and MAC operating systems, that is not so when running under UNIX. Under UNIX, make sure that the extension of m-files is `.m`, not `.M`, and that the capitalization of m-files you call agrees with the actual capitalization of those file names. Otherwise, you will get an error message.

COMMENTS. Use the symbol `"%"` to make comments to document the input, output and calculations in an m-file. Anything after a `%` is not seen as a command, e.g.,

```
>> c = 5 % The rest is a comment
```

```
c =  
    5
```

EXECUTION AND DIRECTORY. To execute the m-file `filename.m`, type `filename` at the MATLAB prompt. The file `filename.m` must be in the current directory. To display the contents of the current directory, use the `dir` command, e.g.,

```
>> dir  
  
. .  
jack.txt  
jill.m
```

To change to a different directory, use the cd command, e.g.,

```
>> cd d:\msande351  
  
.  
..  
airlin27.m  
booklim3.m  
finhor3r.m
```

There are two types of m-files, SCRIPT and FUNCTION.

SCRIPT. Scripts are the simplest type of m-file. They consist of a script of MATLAB commands that have no input or output. They operate on the existing data in the workspace and can create new data on which to operate. Any variables that scripts create remain in the workspace after the script finishes so you can use them for further computations. For example, suppose you create the m-file TwoA.m with the single line

```
B = 2 * A
```

When in MATLAB, create any matrix A, e.g.,

```
>> A = [1 2  
        3 4];
```

and run the m-file TwoA.m by typing

```
>> TwoA
```

MATLAB responds with

```
B =  
    2    4  
    6    8
```

If the matrix B was previously defined in the workspace, it is overwritten with the new matrix.

FUNCTION. Functions are m-files with the following properties:

- * the first noncomment line of a function m-file defines the function;
- * the function accepts input and returns output;
- * the function operate on variables in its own function workspace separate from the workspace you access from the MATLAB prompt.

Consider the function m-file TwoTimes.m that produces the same result as the m-file TwoA.m, viz.,

```
function B = TwoTimes(A) % The syntax of the first line of the file is to type  
                        % "function" followed by output = FunctionName(input). In  
                        % this example A is the input, TwoTimes is the function  
                        % name, and B is the output. The labels A and B of the  
                        % input and output are local to the function workspace.  
B = 2 * A;              % The remainder of the file calculates the function  
                        % output B from its input A
```

The function may then be used in MATLAB. Create any matrix C in MATLAB

```
>> C = [1 2  
        3 4];
```

and suppose the matrix B = [1 1] is in the workspace so

```
>> B  
B =  
    1    1
```

Call the function as if it were built-in, viz.,

```
>> D = TwoTimes(C);  
  
>> D  
D =  
    2    4  
    6    8
```

The function multiplied C by 2 as expected. Even though B is used inside the function file, B is not affected in the workspace. Replacing D and/or C above by A would have produced the same result.

```
>> B  
B =  
    1    1
```

EXPORTING AND IMPORTING DATA

Use the DIARY command to create a diary file that contains text input and output that MATLAB displays on the screen during a session. You can edit the file with a text editor. Type "help diary" at the command line for details.

The simplest method of importing data into MATLAB is to create an m-file containing the data using a text editor, a spreadsheet, etc.

VIEWING and PRINTING GRAPHICS in MATLAB at SWEET HALL or REMOTELY

This subsection explains how to view and print graphics in MATLAB using the workstations in Sweet Hall or remotely. The procedure is illustrated with the m-files for an airline overbooking problem, but the method is general.

The first step is to login to your Leland account on any of the workstations in Sweet Hall. If you do not have a Leland account, open one by typing open at the login prompt. Create a directory for the course by typing mkdir msande351. Copy the files airlin27.m, finhor3r.m and booklim3.m to this directory by using

```
cp /usr/class/msande351/*.* msande351
```

Below two methods of viewing and printing files will be given. The first method uses X Windows and is easiest if you are at Sweet Hall. The second method is needed if you log in remotely with Telnet. Both methods assume that you are in your msande351 directory at Leland.

X WINDOWS

1. Start up the X Window system by typing `x`. (However, most Unix Stations at Sweet Hall have been configured to start X Windows automatically upon login. The user will be asked a question such as "Do you want to start X window? (y/n)".) After a while, a "console" window and a "Local XTerm window" will appear on the screen. Select the console window by clicking on it with your mouse. Move the arrow to the Local XTerm window and type `matlab`.

2. At the MATLAB prompt, type `airlin27`. You will then be asked for the problem parameters, and after the execution of the program the data will be ready and the solution found. The file `airlin27.m` calls both `finhor3r.m` and `booklim3.m`. In separate graphics windows, you will see two figures. One plots the maximum expected flight revenue vs. reservations for each number of hours to flight time not exceeding the one you input. The other plots the optimal booking limits vs. hours to flight time. To get a printout of either figure, activate the window you want to print by clicking on it and then clicking `print` from the file menu. You can also save the figure as a file. Then go to the console window and type

```
lpr -Psweetn filename
```

where `n` is 1 or 2, depending on which printer is closer to you, and `filename` is the Postscript file (like `filename.ps`) created by the previous command. It may take a while to print. You can see the jobs in the queue for printing using `lpq`.

TELNET (USING SAMPSON AND PC LELAND)

1. Type `matlab`.

2. At the MATLAB prompt, type `airlin27`. As above, provide the problem parameters. You will not see the two figures, but they are in memory. To create a Postscript file of figure 1 in your `msande351` directory, type

```
print -f1 FilenameOfFigure.ps.
```

To create a Postscript file of figure 2, use the same command with `-f2` replacing `-f1` above and use a different file name with the `.ps` extension. Then ftp those Postscript figure files to your local machine. You can then print them to a Postscript printer by copying them to the printer from the DOS prompt of a PC. If you want to view the figures or want to use a printer that does not support Postscript, you can either convert them to pdf format using `ps2pdf` at Sweet Hall or Adobe Acrobat Distiller on a PC, or use Ghostview to interpret the Postscript files. In each case, you can then view/print the figure.

MORE on CONVOLUTIONS

CONVOLUTIONS are useful in many areas including the following.

- * inventory control: the expected storage and shortage cost function is the convolution of the storage and shortage cost function and the probability distribution of demand.
- * probability: the probability distribution of the sum of two independent random variables is the convolution of their probability distributions.
- * Markov chains: the sequence of expected population sizes in a Markov chain with immigration is the convolution of the immigration stream and the sequence of powers of the transition matrix.
- * product of polynomials: the vector of coefficients of a product of two polynomials is the convolution of their vectors of coefficients.

The CONVOLUTION $C = A \circ B$ of two finite sequences $A = (A(j))$ and $B = (B(j))$ is the sequence $C = (C(k))$ for which $C(k)$ is the sum of $A(k-i) * B(i)$ over i . If $A(a)$ and $B(b)$ are respectively the first elements of A and B , then the first element of C is $C(c) = A(a) * B(b)$ where $c = a + b$. In the sequel, the term convolution (or true convolution) is used as just defined.

The MATLAB CONVOLUTION $\text{conv}(A,B)$ is slightly different. MATLAB assumes that the first elements of A and B are $A(1)$ and $B(1)$ respectively. Then it follows from the above that $a = b = 1$ and $c = a + b = 2$, so the first element of C is $C(2)$. But MATLAB wants all sequences to have first index one. For this reason, the MATLAB convolution $\text{conv}(A,B)$ takes the true convolution $A \circ B$ and shifts it to the left one position so its first index is also $c - 1 = a + b - 1 = 1$.

CAN MATLAB DO TRUE CONVOLUTIONS $A \circ B$? Absolutely. To explain how, it is necessary to examine the relationship between true convolutions and MATLAB convolutions. Suppose a true convolution $C = A \circ B$ is desired. MATLAB in effect shifts the sequence A (resp., B) to the left $a-1$ (resp., $b-1$) positions, takes the true convolution and shifts the resulting convolution left one more position. The result of these operations by MATLAB is to shift the true convolution to the left $a-1 + b-1 + 1 = a + b - 1$ positions. Thus to obtain the true convolution from the MATLAB convolution, it suffices to shift the MATLAB convolution back to the right $a + b - 1$ positions. That's all there is to it. Of course in the above discussion, shifting $k < 0$ positions to the right (resp., left) is the same as shifting $-k > 0$ positions to the left (resp., right).

EXAMPLE. Suppose g is a storage-and-shortage cost vector with indices the interval $[-15:45]$ and pr is a demand probability vector with indices $[5:15]$. Then the true convolution $g \circ pr$ of g and pr has indices $[-15+5:45+15] = [-10:60]$ obtained by adding the intervals of indices for g and pr . The MATLAB convolution $\text{conv}(g,pr)$ instead has indices $[1:71]$. Thus, all that is required is to shift the MATLAB convolution back to the right $(-15 + 5 - 1) = -11$ positions, i.e., to the left 11 positions.

EXPECTED STORAGE AND SHORTAGE COST G . The expected storage-and-shortage cost vector G is the restriction of the convolution $g \circ pr$ to the interval $[0:50]$, i.e., the values of y for which $y - D$ is in the domain $[-15:45]$ for ALL values of the demand D . The interval $[0:50]$ is formed from $[-15:45]$ by adding the maximum demand 15 to the left end point and adding the minimum demand 5 to the right end point.

Why isn't G equal to the convolution on the entire interval $[-10,60]$? The answer is that the convolution does not correctly calculate G outside of the interval $[0:50]$ because the domain of g is restricted so that not all demands are accounted for outside that interval. For example, the value $(g \circ pr)(-10)$ of $g \circ pr$ at -10 is

$$(g \circ pr)(-10) = g(-10-5) * pr(5)$$

and is not equal to $G(-10)$ because the former includes the contribution of the demand equal to 5, but not any other values. To extend the definition of $G(y)$ to $y < 0$, it would be necessary to extend the domain of definition of g to include the values $g(y - 5), \dots, g(y - 15)$.

Here are two applications. First define the following in MATLAB. In each case the "M" is appended to a symbol to indicate that it is the MATLAB version thereof.

```
GM = conv(g,pr)
Y = [0:50]
YM = Y + 11
G = GM(YM)
```

* PLOT G ON Y. Use the MATLAB function $\text{plot}(Y,G)$

* FIND GLOBAL MINIMUM AND MINIMIZER OF G ON Y . Use the MATLAB formulas

$$[\text{minimum}, M] = \min(G)$$

$$\text{minimizer} = M - 1$$

The negative unit adjustment to the MATLAB minimizer M of G is to take account of the fact that MATLAB minimizes over $Y + 1$ whereas the minimum should be over Y .

MARKOV DECISION CHAINS

STANDARD FORM. The MATLAB formulations of Markov decision chains in MS&E 351 are specified in terms of the states $1, \dots, S$, the numbers $a(s)$ of actions in each state s , and the one-period reward $r(s,a)$ and the transition rates (or probabilities) $P(s,a,t)$ to states $t = 1, \dots, S$ when action a is chosen in state s . The MATLAB files represent this data in a standard form. The reason for having a standard form is to assure that formulations of problems are independent of the algorithms for solving them. The STANDARD FORM is as follows:

- * S is the number of states,
- * a is an S -row vector with $a(s)$ the number of actions in state s ,
- * A is an S -row vector with $A(s)$ the cumulative number of actions in states $1, \dots, s$,
- * r is an $A(S)$ -column vector of one-period rewards for each of the $A(S)$ state-action pairs "stacked" on top of one another, and
- * P is an $A(S) \times S$ matrix of transition rates for each of the $A(S)$ state-action pairs stacked on top of one another.

The above are input data except for A which is calculated as $A = \text{cumsum}(a)$. Notice that $A(S)$ is the total number of state-action pairs.

"Stacking" of the matrices r and P is done in lexicographic order of the state-action pairs (s,a) , i.e., the row of each matrix corresponding to the state-action pair (s,a) is above that for the pair (s',a') if either $s < s'$ or $s = s'$ and $a < a'$.

For example suppose there are three states, two actions in states 1 and 3, one action in state 2 and that the rewards and transition rates are as follows.

state-actions (s,a)	rewards r(s,a)	states t		
		1	2	3
		transition rates P(s,a,t)		
1,1	3	.5	0	.2
1,2	-2	0	.4	.4
2,1	0	0	0	1
3,1	-5	0	0	0
3,2	1	.2	.2	.3

In particular, taking action 2 in state 1 earns the one-period reward -2 and sends the system to states 1,2,3 with the respective rates 0,.4,.4 and stops with rate .2. Similarly taking action 1 in state 3 earns the one-period reward -5 and sends the system to the stopped state with rate 1.

You can put this problem in the workspace in standard form as follows.

```
>> S = 3;
>> a = [2 1 2];
>> A = cumsum(a)
A =
    2    3    5
```

```
>> r = [ 3
        -2
         0
        -5
         1];

>> P = [.5  0  .2
        0  .4  .4
        0  0  1
        0  0  0
        .2  .2  .3];
```

Note that MATLAB relabels the state-action pairs (s,a) as the integers 1,2,3,4,5 corresponding to the five rows of the last two matrices. This reindexing often makes converting problems to standard form in MATLAB a challenge.

CONVERSION OF NATURAL TO STANDARD FORM. When formulating a Markov decision chain in MATLAB, the most natural form of input data is usually different from the standard form used by the MS&E 351 algorithms. Then it is necessary to convert the natural form to standard form. The reshaping operator is often useful for this.

For example, consider a Markov decision chain with three states 1,2,3, two actions 1 and 2 in each state, and reward for taking action a in state s being $s * a$. One natural form of this problem might be as follows:

```
>> SS = [1:3]' % the state space

SS =
     1
     2
     3

>> NA = 1:2 % the actions in each state

NA =
     1     2

>> r = SS * NA % the rewards in natural form

r =
     1     2
     2     4
     3     6
```

so $r(s,a) = s * a$. But this matrix form of the rewards is not in standard form. To put the problem in that form, proceed as follows.

```
>> r = r';

>> r = r(:) % the rewards in standard form

r =
     1
     2
     2
     4
     3
     6
```

This r vector is now in standard form for use by the Markov-decision-chain algorithms.