

71

MATLAB[®]

	71.1	Matrices, Submatrices, and Multidimensional Arrays	71-1
	71.2	Matrix Arithmetic	71-3
	71.3	Built-In MATLAB Functions	71-4
	71.4	Special Matrices	71-5
	71.5	Linear Systems and Least Squares	71-7
	71.6	Eigenvalues and Eigenvectors	71-9
	71.7	Sparse Matrices	71-9
	71.8	Programming	71-11
	71.9	Graphics	71-14
	71.10	Symbolic Mathematics in MATLAB	71-17
	71.11	Graphical User Interfaces	71-19
Steven J. Leon		References	71-22
<i>University of Massachusetts Dartmouth</i>			

MATLAB[®] is generally recognized as the leading software for scientific computation. It was originally developed in the 1970s by Cleve Moler as an interactive Matrix Laboratory with matrix routines based on the algorithms in the LINPACK and EISPACK software libraries. In the original 1978 version everything in MATLAB was done with matrices, even the graphics. MATLAB has continued to grow and expand from the classic 1978 Fortran version to the current version, MATLAB 7, which was released in May 2004. Each new release has included significant improvements. The graphics capabilities were greatly enhanced with the introduction of Handle Graphics and Graphical User Interfaces in version 4 (1992). A sparse matrix package was also included in version 4. Over the years, dozens of toolboxes (application libraries of specialized MATLAB files) have been added in areas such as signal processing, statistics, optimization, symbolic math, splines, and image processing. MATLAB's matrix computations are now based on the LAPACK and BLAS software libraries. MATLAB is widely used in linear algebra courses. Books such as [Leo06], [LHF03], and [HZ04] have extensive sets of MATLAB exercises and projects for the standard first course in linear algebra. The book by D.J. Higham and N.J. Higham [HH03] provides a comprehensive guide to all the basic features of MATLAB.

71.1 Matrices, Submatrices, and Multidimensional Arrays

Facts:

1. The basic data elements that MATLAB uses are matrices. Matrices can be entered into a MATLAB session using square brackets.
2. If A and B are matrices with the same number of rows, then one can append the columns of B to the matrix A and form an augmented matrix C by setting $C = [A, B]$. If E is a matrix with the same number of columns as A , then one can append the rows of E to A and form an augmented matrix F by setting $F = [A; E]$.

- Row vectors of evenly spaced entries can be generated using MATLAB's `:` operator.
- A submatrix of a matrix A is specified by $A(u, v)$ where u and v are vectors that specify the row and column indices of the submatrix.
- MATLAB arrays can have more than two dimensions.

Commands:

- The number of rows and columns of a matrix can be determined using MATLAB's `size` command.
- The command `length(x)` can be used to determine the number of entries in a vector x . The `length` command is equivalent to the command `max(size(x))`.
- MATLAB's `cat` command can be used to concatenate two or more matrices along a single dimension and it can also be used to create multidimensional arrays. If two matrices A and B have the same number of columns, then the command `cat(1, A, B)` produces the same matrix as the command `[A; B]`. If the matrices B and C have the same number of rows, the command `cat(2, B, C)` produces the same matrix as the command `[B, C]`. The `cat` command can be used with more than two matrices as arguments. For example, the command `cat(2, A, B, C, D)` generates the same matrix as the command `[A, B, C, D]`. If A_1, A_2, \dots, A_k are $m \times n$ matrices, the command $S = \text{cat}(3, A_1, A_2, \dots, A_k)$ will generate an $m \times n \times k$ array S .

Examples:

- The matrix

$$A = \begin{bmatrix} 1 & 2 & 4 & 3 \\ 3 & 5 & 7 & 2 \\ 2 & 4 & 1 & 1 \end{bmatrix}$$

is generated using the command

$$A = \begin{bmatrix} 1 & 2 & 4 & 3 \\ 3 & 5 & 7 & 2 \\ 2 & 4 & 1 & 1 \end{bmatrix}.$$

Alternatively, one could generate the matrix on a single line using semicolons to designate the ends of the rows

$$A = [1\ 2\ 4\ 3; 3\ 5\ 7\ 2; 2\ 4\ 1\ 1].$$

The command `size(A)` will return the vector $(3, 4)$ as the answer. The command `size(A,1)` will return the value 3, the number of rows of A , and the command `size(A,2)` will return the value 4, the number of columns of A .

- The command

$$x = 3 : 7$$

will generate the vector $x = (3, 4, 5, 6, 7)$. To change the step size to $\frac{1}{2}$, set

$$z = 3 : 0.5 : 7.$$

This will generate the vector $z = (3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7)$. The MATLAB commands `length(x)` and `length(z)` will generate the answers 5 and 9, respectively.

- If A is the matrix in Example 1, then the command `A(2,:)` will generate the second row vector of A and the command `A(:,3)` will generate the third column vector of A . The submatrix of elements that are in the first two rows and last two columns is given by `A(1:2, 3:4)`. Actually there is no need to use adjacent rows or columns. The command `C = A([1 3], [1 3 4])` will generate the submatrix

$$C = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 1 & 1 \end{bmatrix}.$$

4. The command

$$S = \text{cat}(3, [5\ 1\ 2; 3\ 2\ 1], [1\ 2\ 3; 4\ 5\ 6])$$

will produce a $2 \times 3 \times 2$ array S with

$$S(:, :, 1) = \begin{array}{ccc} 5 & 1 & 2 \\ 3 & 2 & 1 \end{array}$$

and

$$S(:, :, 2) = \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6. \end{array}$$

71.2 Matrix Arithmetic

The six basic MATLAB operators for doing matrix arithmetic are: $+$, $-$, $*$, $^$, \backslash , and $/$. The matrix left and right divide operators, \backslash and $/$, are described in Section 71.5. These same operators are also used for doing scalar arithmetic.

Facts:

1. If A and B are matrices with the same dimensions, then their sum and difference are computed using the commands: $A + B$ and $A - B$.
2. If B and C are matrices and the multiplication BC is possible, then the product $E = BC$ is computed using the command

$$E = B * C.$$

3. The k th power of a square matrix A is computed with the command A^k .
4. Scalars can be either real or complex numbers. A complex number such as $3 + 4i$ is entered in MATLAB as $3 + 4i$. It can also be entered as $3 + 4 * \text{sqrt}(-1)$ or by using the command $\text{complex}(3, 4)$. If i is used as a variable and assigned a value, say $i = 5$, then MATLAB will assign the expression $3 + 4 * i$ the value 23; however, the expression $3 + 4i$ will still represent the complex number $3 + 4i$. In the case that i is used as a variable and assigned a numerical value, one should be careful to enter a complex number of the form $a + i$ (where a real) as $a + 1i$.
5. MATLAB will perform arithmetic operations element-wise when the operator is preceded by a period in the MATLAB command.
6. The conjugate transpose of a matrix B is computed using the command B' . If the matrix B is real, then B' will be equal to the transpose of B . If B has complex entries, then one can take its transpose without conjugating using the command B' .

Commands:

1. The inverse of a nonsingular matrix C is computed using the command $\text{inv}(C)$.
2. The determinant of a square matrix A is computed using the command $\text{det}(A)$.

Examples:

1. If

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 5 & 1 \\ 2 & 3 \end{bmatrix}$$

the commands $A * B$ and $A \cdot B$ will generate the matrices

$$\begin{bmatrix} 9 & 7 \\ 23 & 15 \end{bmatrix}, \quad \begin{bmatrix} 5 & 2 \\ 6 & 12 \end{bmatrix}.$$

71.3 Built-In MATLAB Functions

The `inv` and `det` commands are examples of built-in MATLAB functions. Both functions have a single input and a single output. Thus, the command $d = \text{det}(A)$ has the matrix A as its input argument and the scalar d as its output argument. A MATLAB function may have many input and output arguments. When a command of the form

$$[A_1, \dots, A_k] = \text{fname}(B_1, \dots, B_n) \quad (71.1)$$

is used to call a function `fname` with input arguments B_1, \dots, B_n , MATLAB will execute the function routine and return the values of the output arguments A_1, \dots, A_k .

Facts:

1. The number of allowable input and output arguments for a MATLAB function is defined by a function statement in the MATLAB file that defines the function. (See section 71.8.) The function may require some or all of its input arguments. A MATLAB command of the form (71.1) may be used with j output arguments where $0 \leq j \leq k$. The MATLAB help facility describes the various input and output options for each of the MATLAB commands.

Examples:

1. The MATLAB function `pi` is used to generate the number π . This function is used with no input arguments.
2. The MATLAB function `kron` has two input arguments. If A and B are matrices, then the command `kron(A,B)` computes the Kronecker product of A and B . Thus, if $A = [1, 2; 3, 4]$ and $B = [1, 1; 1, 1]$, then the command $K = \text{kron}(A, B)$ produces the matrix

$$K = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix}$$

and the command $L = \text{kron}(B, A)$ produces the matrix

$$L = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{bmatrix}.$$

3. One can compute the QZ factorization (see section 71.6) for the generalized eigenvalue problem using a command

$$[E, F, Q, Z] = \text{qz}(A, B)$$

with two input arguments and four outputs. The input arguments are square matrices A and B and the outputs are quasitriangular matrices E and F and unitary matrices Q and Z such that

$$QAZ = E \quad \text{and} \quad QBZ = F.$$

The command

$$[E, F, Q, Z, V, W] = \text{qz}(A, B)$$

will also compute matrices V and W of generalized eigenvectors.

71.4 Special Matrices

The ELMAT directory of MATLAB contains a collection of MATLAB functions for generating special types of matrices.

Commands:

1. The following table lists commands for generating various types of special matrices.

Matrix	Command Syntax	Description
eye	eye(n)	Identity matrix
ones	ones(n) or ones(m, n)	Matrix whose entries are all equal to 1
zeros	zeros(n) or zeros(m, n)	Matrix whose entries are all equal to 0
rand	rand(n) or rand(m, n)	Random matrix
compan	compan(p)	Companion matrix
hadamard	hadamard(n)	Hadamard matrix
gallery	gallery(matname, $p1, p2, \dots$)	Large collection of special test matrices
hankel	hankel(c) or hankel(c, r)	Hankel matrix
hilb	hilb(n)	Hilbert matrix
invhilb	invhilb(n)	Inverse Hilbert matrix
magic	magic(n)	Magic square
pascal	pascal(n)	Pascal matrix
rosser	rosser	Test matrix for eigenvalue solvers
toeplitz	toeplitz(c) or toeplitz(c, r)	Toeplitz matrix
vander	vander(x)	Vandermonde matrix
wilkinson	wilkinson(n)	Wilkinson's eigenvalue test matrix

2. The command `gallery` can be used to access a large collection of test matrices developed by N. J. Higham. Enter `help gallery` to obtain a list of all classes of gallery test matrices.

Examples:

1. The command `rand(n)` will generate an $n \times n$ matrix whose entries are random numbers that are uniformly distributed in the interval $(0, 1)$. The command may be used with two input arguments

to generate nonsquare matrices. For example, the command `rand(3,2)` will generate a random 3×2 matrix. The command `rand` when used by itself with no input arguments will generate a single random number between 0 and 1.

2. The command

$$A = [\text{eye}(2), \text{ones}(2,3); \text{zeros}(2), 2 * \text{ones}(2,3)]$$

will generate the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 2 & 2 & 2. \end{bmatrix}$$

3. The command `toeplitz(c)` will generate a symmetric toeplitz matrix whose first column is the vector `c`. Thus, the command

$$\text{toeplitz}([1; 2; 3])$$

will generate

$$T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1. \end{bmatrix}$$

Note that in this case, since the `toeplitz` command was used with no output argument, the computed value of the command `toeplitz(c)` was assigned to the temporary variable `ans`. Further computations may end up overwriting the value of `ans`. To keep the matrix for further use in the MATLAB session, it is advisable to include an output argument in the calling statement.

For a nonsymmetric Toeplitz matrix it is necessary to include a second input argument `r` to define the first row of the matrix. If $r(1) \neq c(1)$, the value of $c(1)$ is used for the main diagonal. Thus, commands

$$c = [1; 2; 3], \quad r = [9, 5, 7], \quad T = \text{toeplitz}(c, r)$$

will generate

$$T = \begin{bmatrix} 1 & 5 & 7 \\ 2 & 1 & 5 \\ 3 & 2 & 1. \end{bmatrix}$$

The Toeplitz matrix generated is stored using the variable `T`.

4. One of the classes of gallery test matrices is circulant matrices. These are generated using the MATLAB function `circul`. To see how to use this function, enter the command

$$\text{help private} \backslash \text{circul}.$$

The help information will tell you that the `circul` function requires an input vector `v` and that the command

$$C = \text{gallery}(' \text{circul}', v)$$

will generate a circulant matrix whose first row is \mathbf{v} . Thus, the command

$$C = \text{gallery}('circul', [4, 5, 6])$$

will generate the matrix

$$C = \begin{bmatrix} 4 & 5 & 6 \\ 6 & 4 & 5 \\ 5 & 6 & 4 \end{bmatrix}$$

71.5 Linear Systems and Least Squares

The simplest way to solve a linear system in MATLAB is to use the matrix left divide operator.

Facts:

1. The symbol \backslash represents MATLAB's matrix left divide operator. One can compute the solution to a linear system $A\mathbf{x} = \mathbf{b}$ by setting

$$\mathbf{x} = A \backslash \mathbf{b}.$$

If A is an $n \times n$ matrix, then MATLAB will compute the solution using Gaussian elimination with partial pivoting. A warning message is given when the matrix is badly scaled or nearly singular. If the coefficient matrix is nonsquare, then MATLAB will return a least squares solution to the system that is essentially equivalent to computing $A^\dagger \mathbf{b}$ (where A^\dagger denotes the pseudoinverse of A). In this case, MATLAB determines the numerical rank of the coefficient matrix using a QR decomposition and gives a warning when the matrix is rank deficient.

If A is an $m \times n$ matrix and B is $m \times k$, then the command

$$C = A \backslash B$$

will produce an $n \times k$ matrix whose column vectors satisfy

$$\mathbf{c}_j = A \backslash \mathbf{b}_j \quad j = 1, \dots, k.$$

2. The symbol $/$ represents MATLAB's matrix right divide operator. It is defined by

$$B/A = (A' \backslash B')'.$$

In the case that A is nonsingular, the computation B/A is essentially the same as computing BA^{-1} , however, the computation is carried out without actually computing A^{-1} .

Commands:

The following table lists some of the main MATLAB commands that are useful for linear systems.

Function	Command Syntax	Description
rref	$U = \text{rref}(A)$	Reduced row echelon form of a matrix
lu	$[L, U] = \text{lu}(A)$	LU factorization
linsolve	$x = \text{linsolve}(A, b, \text{opts})$	Efficient solver for structured linear systems
chol	$R = \text{chol}(A)$	Cholesky factorization of a matrix
norm	$p = \text{norm}(X)$	Norm of a matrix or a vector
null	$U = \text{null}(A)$	Basis for the null space of a matrix
null	$R = \text{null}(A, 'r')$	Basis for null space rational form
orth	$Q = \text{orth}(A)$	Orthonormal basis for the column space of a matrix
rank	$r = \text{rank}(A)$	Numerical rank of a matrix
cond	$c = \text{cond}(A)$	2-norm condition number for solving linear systems
rcond	$c = \text{rcond}(A)$	Reciprocal of approximate 1-norm condition number
qr	$[Q, R] = \text{qr}(A)$	QR factorization
svd	$s = \text{svd}(A)$	Singular values of a matrix
svd	$[U, S, V] = \text{svd}(A)$	Singular value decomposition
pinv	$B = \text{pinv}(A)$	Pseudoinverse of a matrix

Examples:

- The null command can be used to produce an orthonormal basis for the nullspace of a matrix. It can also be used to produce a “rational” nullspace basis obtained from the reduced row echelon form of the matrix. If

$$A = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

then the command $U = \text{null}(A)$ will produce the matrix

$$U = \begin{bmatrix} -0.8165 & -0.0000 \\ 0.4082 & 0.7071 \\ 0.4082 & -0.7071 \\ -0.0000 & 0.0000 \end{bmatrix}$$

where the entries of U are shown in MATLAB's format short (with four-digit mantissas). The column vectors of U form an orthonormal basis for the nullspace of A . The command $R = \text{null}(A, 'r')$ will produce a matrix R whose columns form a simple basis for the nullspace.

$$R = \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

- MATLAB defines the numerical rank of a matrix to the number of singular values of the matrix that are greater than

$$\max(\text{size}(A)) * \text{norm}(A) * \text{eps}$$

where `eps` has the value 2^{-52} , which is a measure of the precision used in MATLAB computations. Let H be the 12×12 Hilbert matrix. The singular values of H can be computed using the command `s = svd(H)`. The smallest singular values are $s(11) \approx 2.65 \times 10^{-14}$ and $s(12) \approx 10^{-16}$. Since the value of `eps` is approximately 2.22×10^{-16} , the computed value of `rank(H)` will be the numerical rank 11, even though the exact rank of H is 12. The computed value of `cond(H)` is approximately 1.8×10^{16} and the computed value of `rcond(H)` is approximately 2.6×10^{-17} .

71.6 Eigenvalues and Eigenvectors

MATLAB's `eig` function can be used to compute both the eigenvalues and eigenvectors of a matrix.

Commands:

1. The `eig` command. Given a square matrix A , the command `e = eig(A)` will generate a column vector e whose entries are the eigenvalues of A . The command `[X, D] = eig(A)` will generate a matrix X whose column vectors are the eigenvectors of A and a diagonal matrix D whose diagonal entries are the eigenvalues of A .
2. The `eigshow` command. MATLAB's `eigshow` utility provides a visual demonstration of eigenvalues and eigenvectors of 2×2 matrices. The utility is invoked by the command `eigshow(A)`. The input argument A must be a 2×2 matrix. The command can also be used with no input argument, in which case MATLAB will take `[1 3; 4 2]/4` as the default 2×2 matrix. The `eigshow` utility shows how the image Ax changes as we rotate a unit vector x around a circle. This rotation is carried out manually using a mouse. If A has real eigenvalues, then we can observe the eigenvectors of the matrix when the vectors x and Ax are in the same or opposite directions.
3. The command `J = jordan(A)` can be used to compute the Jordan canonical form of a matrix A . This command will only give accurate results if the entries of A are exactly represented, i.e., the entries must be integers or ratios of small integers. The command `[X, J] = jordan(A)` will also compute the similarity matrix X so that $A = XJX^{-1}$.
4. The following table lists some additional MATLAB functions that are useful for eigenvalue related problems.

Function	Command Syntax	Description
<code>poly</code>	<code>p = poly(A)</code>	Characteristic polynomial of a matrix
<code>hess</code>	<code>H = hess(A)</code> or <code>[U, H] = hess(A)</code>	Hessenberg form
<code>schur</code>	<code>T = schur(A)</code> or <code>[U, T] = schur(A)</code>	Schur decomposition
<code>qz</code>	<code>[E, F, Q, Z] = qz(A, B)</code>	QZ factorization for generalized eigenvalues
<code>condeig</code>	<code>s = condeig(A)</code>	Condition numbers for the eigenvalues of A
<code>expm</code>	<code>E = expm(A)</code>	Matrix exponential

71.7 Sparse Matrices

A matrix is *sparse* if most of its entries are zero. MATLAB has a special data structure for handling sparse matrices. This structure stores the nonzero entries of a sparse matrix together with their row and column indices.

Commands:

1. The command `sparse` is used to generate sparse matrices. When used with a single input argument the command

$$S = \text{sparse}(A)$$

will convert an ordinary MATLAB matrix A into a matrix S having the sparse data structure. More generally, a command of the form

$$S = \text{sparse}(i, j, s, m, n, \text{nzmax})$$

will generate an $m \times n$ sparse matrix S whose nonzero entries are the entries of the vector s . The row and column indices of the nonzero entries are given by the vectors i and j . The last input argument nzmax specifies the total amount of space allocated for nonzero entries. If the allocation argument is omitted, by default MATLAB will set it to equal the value of $\text{length}(s)$.

- MATLAB's `spy` command can be used to plot the sparsity pattern of a matrix. In these plots the matrix is represented by a rectangular box with dots corresponding to the positions of its nonzero entries.
- The MATLAB directory SPARFUN contains a large collection of MATLAB functions for working with sparse matrices. The general sparse linear algebra functions are given in the following table.

MATLAB Function	Description
<code>eigs</code>	A few eigenvalues, using ARPACK
<code>svds</code>	A few singular values, using <code>eigs</code>
<code>luinc</code>	Incomplete LU factorization
<code>cholinc</code>	Incomplete Cholesky factorization
<code>normest</code>	Estimate the matrix 2-norm
<code>condest</code>	1-norm condition number estimate
<code>sprank</code>	Structural rank

All of these functions require a sparse matrix as an input argument. All have one basic output argument except in the case of `luinc`, where the basic output consists of the L and U factors.

- The SPARFUN directory also includes a collection of routines for the iterative solution of sparse linear systems.

MATLAB Function	Description
<code>pcg</code>	Preconditioned Conjugate Gradients Method
<code>bicg</code>	BiConjugate Gradients Method
<code>bicgstab</code>	BiConjugate Gradients Stabilized Method
<code>cgs</code>	Conjugate Gradients Squared Method
<code>gmres</code>	Generalized Minimum Residual Method
<code>lsqr</code>	Conjugate Gradients on the Normal Equations
<code>minres</code>	Minimum Residual Method
<code>qmr</code>	Quasi-Minimal Residual Method
<code>symmlq</code>	Symmetric LQ Method

If A is a sparse coefficient matrix and B is a matrix of right-hand sides, then one can solve the equation $AX = B$ using a command of the form $X = \text{fname}(A, B)$, where `fname` is one of the iterative solver functions in the table.

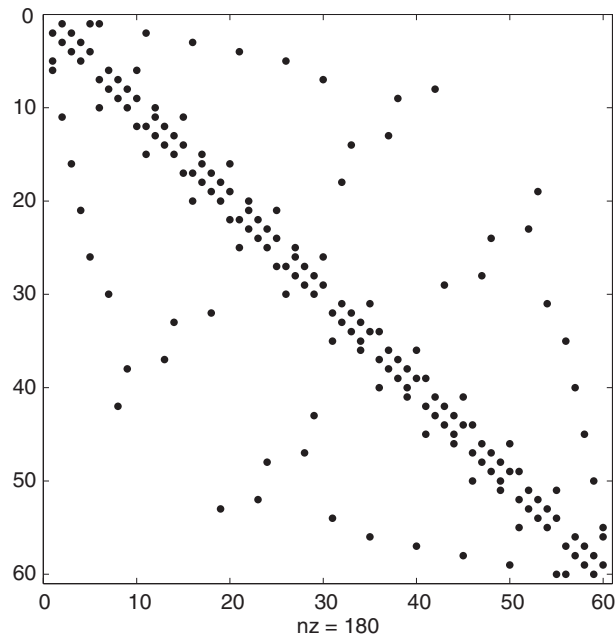
Examples:

- The command

$$S = \text{sparse}([25, 37, 8], [211, 15, 92], [4.5, 3.2, 5.7], 200, 300)$$

will generate a 200×300 sparse matrix S whose only nonzero entries are

$$s_{25,211} = 4.5, \quad s_{37,15} = 3.2, \quad s_{8,92} = 5.7.$$

FIGURE 71.1 `Spy(B)`.

2. The command `B = bucky` will generate the 60×60 sparse adjacency matrix B of the connectivity graph of the Buckminster Fuller geodesic dome and the command `spy(B)` will generate the spy plot shown in Figure 71.1.

71.8 Programming

MATLAB has built in all of the main structures one would expect from a high-level computer language. The user can extend MATLAB by adding on programs and new functions.

Facts:

1. MATLAB programs are called *M-files* and should be saved with a `.m` extension.
2. MATLAB programs may be in the form of *script* files that list a series of commands to be executed when the file is called in a MATLAB session, or they can be in the form of MATLAB functions.
3. MATLAB programs frequently include for loops, while loops, and if statements.
4. A function file must start with a function statement of the form

$$\text{function } [\text{oarg1}, \dots, \text{oargk}] = \text{fname}(\text{iarg1}, \dots, \text{iargj})$$

where `fname` is the name of the function, `iarg1, ..., iargj` are its input arguments, and `oarg1, ..., oargk` are the output arguments. In calling a MATLAB function, it is not necessary to use all of the input and output allowed for in the general syntax of the command. In fact, MATLAB functions are commonly used with no output arguments whatsoever.

5. One can construct simple functions interactively in a MATLAB session using MATLAB's `inline` command. A simple function such as $f(t) = t^2 + 4$ can be described by the character array (or string)

" $t^2 + 4$." The inline command will transform the string into a function for use in the current MATLAB session. Inline functions are particularly useful for creating functions that are used as input arguments for other MATLAB functions. An inline function is not saved as an m-file and consequently is lost when the MATLAB session is ended.

6. One can use the same MATLAB command with varying amounts of input and output arguments. MATLAB keeps track of the number of input and output arguments included in the call statement using the functions `nargin` (the number of input arguments) and `nargout` (the number of output arguments). These commands are used inside the body of a MATLAB function to tailor the computations and output to the specifications of the calling statement.
7. MATLAB has six relational operators that are used for comparisons of scalars or elementwise comparisons of arrays. These operators are:

Relational Operators	
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
~=	not equal

8. There are three logical operators as shown in the following table:

Logical Operators	
&	AND
	OR
~	NOT

These logical operators regard any nonzero scalar as corresponding to TRUE and 0 as corresponding to FALSE. The operator `&` corresponds to the logical AND. If a and b are scalars, the expression $a \& b$ will equal 1 if a and b are both nonzero (TRUE) and 0 otherwise. The operator `|` corresponds to the logical OR. The expression $a | b$ will have the value 0 if a and b are both 0 and otherwise it will be equal to 1. The operator `~` corresponds to the logical NOT. For a scalar a , the expression $\sim a$ takes on the value 1 (TRUE) if $a = 0$ (FALSE) and the value 0 (FALSE) if $a \neq 0$ (TRUE).

For matrices these operators are applied element-wise. Thus, if A and B are both $m \times n$ matrices, then $A \& B$ is a matrix of zeros and ones whose (i, j) entry is $a(i, j) \& b(i, j)$.

Examples:

1. Given two $m \times n$ matrices A and B , the command $C = A < B$ will generate an $m \times n$ matrix consisting of zeros and ones. The (i, j) entry will be equal to 1 if and only if $a_{ij} < b_{ij}$. If

$$A = \begin{bmatrix} -1 & 1 & 0 \\ 4 & -2 & 5 \\ 1 & -3 & -2 \end{bmatrix},$$

then command $A >= 0$ will generate

$$\begin{array}{l} \text{ans} = \\ 0 \ 1 \ 1 \\ 1 \ 0 \ 1 \\ 1 \ 0 \ 0. \end{array}$$

2. If

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & 2 \\ 0 & 3 \end{bmatrix},$$

then

$$A \& B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad A|B = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \sim A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

3. To construct the function $g(t) = 3 \cos t - 2 \sin t$ interactively set

$$g = \text{inline}('3 * \cos(t) - 2 * \sin(t)').$$

If one then enters $g(0)$ on the command line, MATLAB will return the answer 3. The command `ezplot(g)` will produce a plot of the graph of $g(t)$. (See Section 71.9 for more information on producing graphics.)

4. If the numerical nullity of a matrix is defined to be the number of columns of the matrix minus the numerical rank of the matrix, then one can create a file `numnull.m` to compute the numerical nullity of a matrix. This can be done using the following lines of code.

```
function k = numnull(A)
% The command numnull(A) computes the numerical nullity of A
[m,n] = size(A);
k = n - rank(A);
```

The line beginning with the `%` is a comment that is not executed. It will be displayed when the command `help numnull` is executed. The semicolons suppress the printouts of the individual computations that are performed in the function program.

5. The following is an example of a MATLAB function to compute the circle that gives the best least squares fit to a collection of points in the plane.

```
function [center,radius,e] = circfit(x,y,w)
% The command [center,radius] = circfit(x,y) generates
% the center and radius of the circle that gives the
% best least squares fit to the data points specified
% by the input vectors x and y. If a third input
% argument is specified then the circle and data
% points will be plotted. Specify a third output
% argument to get an error vector showing how much
% each point deviates from the circle.
if size(x,1) == 1 & size(y,1) == 1
    x = x'; y = y';
end
A = [2*x, 2*y, ones(size(x))];
b = x.^2 + y.^2;
c = A\b;
center = c(1:2)';
radius = sqrt(c(3) + c(1)^2 + c(2)^2);
```

```

if nargin > 2
    t = 0:0.1:6.3;
    u = c(1) + radius * cos(t);
    v = c(2) + radius * sin(t);
    plot(x,y,'x',u,v)
    axis('equal')
end
if nargin == 3
    e = sqrt((x - c(1)).^2 + (y - c(2)).^2) - radius;
end

```

The command `plot(x,y,'x',u,v)` is used to plot the original (x, y) data as discrete points in the plane, with each point designated by an “x,” and to also, on the same axis system, plot the (u, v) data points as a continuous curve. The following section explains MATLAB plot commands in greater detail.

71.9 Graphics

MATLAB graphics utilities allow the user to do simple two- and three-dimensional plots as well as more sophisticated graphical displays.

Facts:

1. MATLAB incorporates an objected-oriented graphics system called *Handle Graphics*. This system allows the user to modify and add on to existing figures and is useful in producing computer animations.
2. MATLAB's graphics capabilities include digital imaging tools. MATLAB images may be indexed or true color.
3. An indexed image requires two matrices, a $k \times 3$ colormap matrix whose rows are triples of numbers that specify red, green, blue intensities, and an $m \times n$ image matrix whose entries assign a colormap triple to each pixel of the image.
4. A true color image is one derived from an $m \times n \times 3$ array, which specifies the red, green, blue triplets for each pixel of the image.

Commands:

1. The `plot` command is used for simple plots of x - y data sets. Given a set of (x_i, y_i) data points, the command `plot(x,y)` plots the data points and by default sequentially draws line segments to connect the points. A third input argument may be used to specify a color (the default color for plots is black) or to specify a different form of plot such as discrete points or dashed line segments.
2. The `ezplot` command is used for plots of functions. The command `ezplot(f)` plots the function $f(x)$ on the default interval $(-2\pi, 2\pi)$ and the command `ezplot(f,[a,b])` plots the function over the interval $[a,b]$.
3. The commands `plot3` and `ezplot3` are used for three-dimensional plots.
4. The command `meshgrid` is used to generate an xy -grid for surface and contour plots. Specifically the command `[X,Y] = meshgrid(u,v)` transforms the domain specified by vectors u and v into arrays X and Y that can be used for the evaluation of functions of two variables and 3-D surface plots. The rows of the output array X are copies of the vector u and the columns of the output array Y are copies of the vector v . The command can be used with only one input argument in which case `meshgrid(u)` will produce the same arrays as the command `meshgrid(u,u)`.

5. The `mesh` command is used to produce wire frame surface plots and the command `surf` produces a solid surface plot. If $[X, Y] = \text{meshgrid}(u, v)$ and $Z(i, j) = f(u_i, v_j)$, then the command `mesh(X, Y, Z)` will produce a wire frame plot of the function $z = f(x, y)$ over the domain specified by the vectors u and v . Similarly the command `surf(X, Y, Z)` will generate a surface plot over the domain.
6. The MATLAB functions `contour` and `ezcontour` produce contour plots for functions of two variables.
7. The command `meshc` is used to graph both the mesh surface and the contour plot in the same graphics window. Similarly the command `surfc` will produce a surf plot with a contour graph appended.
8. Given an array C whose entries are all real, the command `image(C)` will produce a two-dimensional image representation of the array. Each entry of C will correspond to a small patch of the image. The image array C may be either $m \times n$ or $m \times n \times 3$. If C is an $m \times n$ matrix, then the colors assigned to each patch are determined by MATLAB's current colormap. If C is $m \times n \times 3$, a true color array, then no color map is used. In this case the entries of $C(:, :, 1)$ determine the red intensities of the image, the entries of $C(:, :, 2)$ determine green intensities, and the elements of $C(:, :, 3)$ define the blue intensities.
9. The `colormap` command is used to specify the current colormap for image plots of $m \times n$ arrays.
10. The `imread` command is used to translate a standard graphics file, such as a gif, jpeg, or tiff file, into a true color array. The command can also be used with two output arguments to determine an indexed image representation of the graphics file.

Examples:

1. The graph of the function $f(x) = \cos(x) + \sin^2(x)$ on the interval $(-2\pi, 2\pi)$ can be generated in MATLAB using the following commands:

```
x = -6.3: 0.1: 6.3;
y = cos(x) + sin(x).^2;
plot(x,y)
```

The graph can also be generated using the `ezplot` command. (See Figure 71.2.)

```
f = inline('cos(x) + sin(x).^2')
ezplot(f)
```

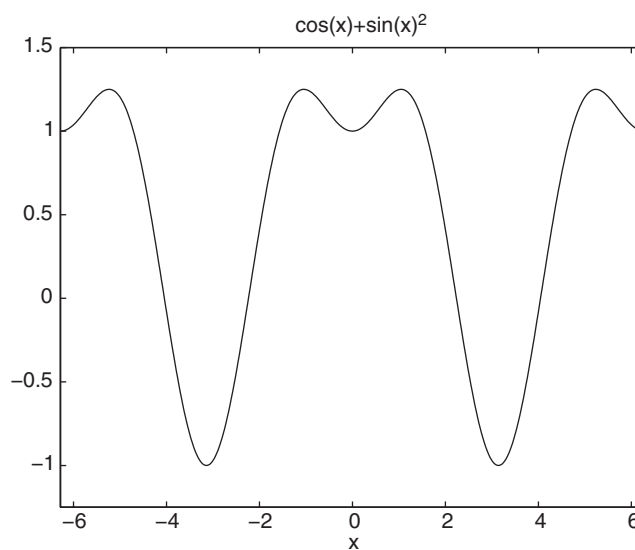


FIGURE 71.2

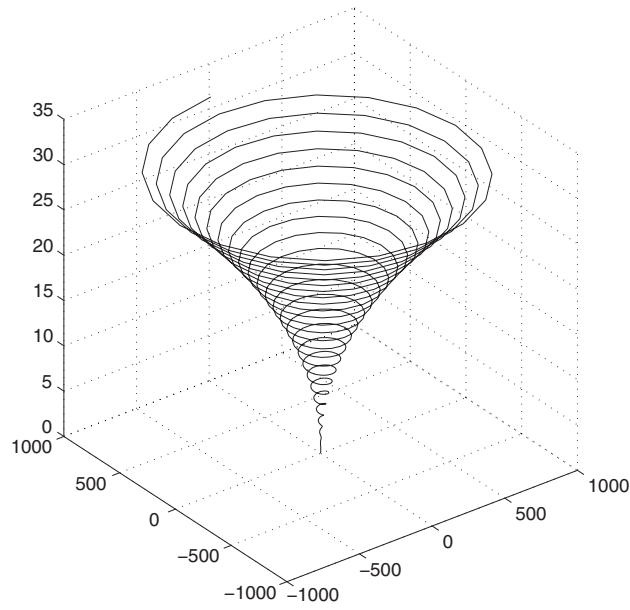


FIGURE 71.3

2. We can generate a three-dimensional plot using the following commands:

```
t = 0 : pi/50 : 10 * pi;
plot3(t.^2 .* sin(5 * t), t.^2 .* cos(5 * t), t)
grid on
axis square
```

These commands generate the plot shown in Figure 71.4.

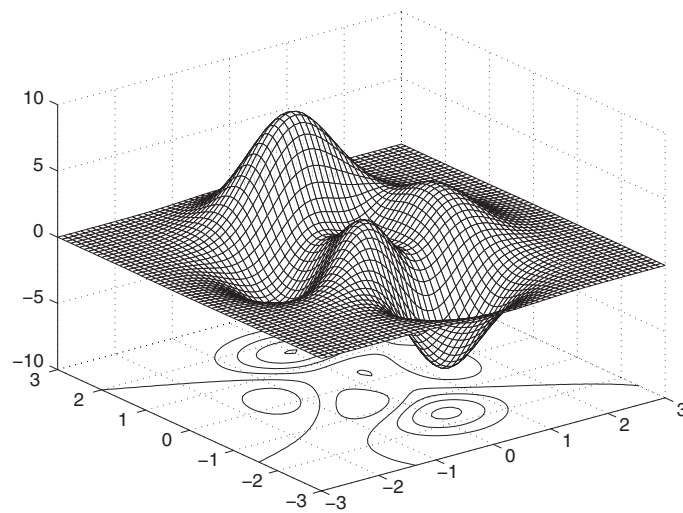


FIGURE 71.4

- MATLAB's `peaks` function is a function of two variables obtained by translating and scaling Gaussian distributions. The commands

```
[X,Y] = meshgrid(-3:0.1:3);
Z = peaks(X,Y);
meshc(X,Y,Z);
```

generate the mesh and contour plots of the peaks function. (See Figure 71.4.)

71.10 Symbolic Mathematics in MATLAB

MATLAB's Symbolic Toolbox is based upon the Maple kernel from the software package produced by Waterloo Maple, Inc. The toolbox allows users to do various types of symbolic computations using the MATLAB interface and standard MATLAB commands. All symbolic computations in MATLAB are performed by the Maple kernel. For details of how symbolic linear algebra computations such as matrix inverses and eigenvalues are carried out see Chapter 72.

Facts:

- MATLAB's symbolic toolbox allows the user to define a new data type, a symbolic object. The user can create symbolic variables and symbolic matrices (arrays containing symbolic variables).
- The standard matrix operations `+`, `-`, `*`, `^`, `'` all work for symbolic matrices and also for combinations of symbolic and numeric matrices. To add a symbolic matrix and a numeric matrix, MATLAB first transforms the numeric matrix into a symbolic object and then performs the addition using the Maple kernel. The result will be a symbolic matrix. In general if the matrix operation involves at least one symbolic matrix, then the result will be a symbolic matrix.
- Standard MATLAB commands such as `det`, `inv`, `eig`, `null`, `trace`, `rref`, `rank`, and `sum` work for symbolic matrices.
- Not all of the MATLAB matrix commands work for symbolic matrices. Commands such as `norm` and `orth` do not work and none of the standard matrix factorizations such as *LU* or *QR* work.
- MATLAB's symbolic toolbox supports variable precision floating arithmetic, which is carried out within the Maple kernel.

Commands:

- The `sym` command can be used to transform any MATLAB data structure into a symbolic object. If the input argument is a string, the result is a symbolic number or variable. If the input argument is a numeric scalar or matrix, the result is a symbolic representation of the given numeric values.
- The `syms` command allows the user to create multiple symbolic variables with a single command.
- The command `subs` is used to substitute for variables in a symbolic expression.
- The command `colspace` is used to find a basis for the column space of a symbolic matrix.
- The commands `ezplot`, `ezplot3`, and `ezsurf` are used to plot symbolic functions of one or two variables.
- The command `vpa(A,d)` evaluates the matrix *A* using variable precision floating point arithmetic with *d* decimal digits of accuracy. The default value of *d* is 32, so if the second input argument is omitted, the matrix will be evaluated with 32 digits of accuracy.

Examples:

- The command

```
t = sym('t')
```

transforms the string 't' into a symbolic variable t. Once the symbolic variable has been defined, one can then perform symbolic operations. For example, the command

$$\text{factor}(t^2 - 4)$$

will result in the answer

$$(t - 2) * (t + 2).$$

2. The command

$$\text{syms } a \ b \ c$$

creates the symbolic variables a, b, and c. If we then set

$$A = [a, \ b, \ c; \ b, \ c, \ a; \ c, \ a, \ b]$$

the result will be the symbolic matrix

$$A = \begin{bmatrix} a, & b, & c \\ c, & a, & b \\ b, & c, & a \end{bmatrix}.$$

Note that for a symbolic matrix the MATLAB output is in the form of a matrix of row vectors with each row vector enclosed by square brackets.

3. Let A be the matrix defined in the previous example. We can add the 3×3 Hilbert matrix to A using the command $B = A + \text{hilb}(3)$. The result is the symbolic matrix

$$B = \begin{bmatrix} a + 1, & b + 1/2, & c + 1/3 \\ c + 1/4, & a + 1/5, & b + 1/6 \\ b + 1/7, & c + 1/8, & a + 1/9 \end{bmatrix}.$$

To substitute 2 for a in the matrix A we set

$$A = \text{subs}(A, a, 2).$$

The matrix A then becomes

$$A = \begin{bmatrix} 2, & b, & c \\ c, & 2, & b \\ b, & c, & 2 \end{bmatrix}.$$

Multiple substitutions are also possible. To replace b by b + 1 and c by 5, one need only set

$$A = \text{subs}(A, [b, c], [b + 1, 5]).$$

4. If a is declared to be a symbolic variable, the command

$$A = [1 \ 2 \ 1; \ 2 \ 4 \ 2; \ 0 \ 0 \ a]$$

will produce the symbolic matrix

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 6 \\ 0 & 0 & a \end{bmatrix}.$$

The eigenvalues 0, 5, and a are computed using the command `eig(A)`. The command

$$[X, D] = \text{eig}(A)$$

generates a symbolic matrix of eigenvectors

$$X = \begin{bmatrix} -2, & 1/2 * (a + 8)/(-2 + 3 * a), & 1 \\ 1, & & 1, & 2 \\ 0, & 1/2 * a * (a - 5)/(-2 + 3 * a), & 0 \end{bmatrix}$$

and the diagonal matrix

$$D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 5 \end{bmatrix}.$$

When $a = 0$ the matrix A will be defective. One can substitute 0 for a in the matrix of eigenvectors using the command

$$X = \text{subs}(X, a, 0).$$

This produces the numeric matrix

$$X = \begin{bmatrix} -2 & -2 & 1 \\ 1 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}.$$

5. If we set $z = \exp(1)$, then MATLAB will compute an approximation to e that is accurate to 16 decimal digits. The command `vpa(z)` will produce a 32 digit representation of z , but only the first 16 digits will be accurate approximations to the digits of e . To compute e more accurately one should apply the `vpa` function to the symbolic expression `'exp(1)'`. The command `z = vpa('exp(1)')` produces an answer $z = 2.7182818284590452353602874713527$, which is accurate to 32 digits.

71.11 Graphical User Interfaces

A graphical user interface (GUI) is a user interface whose components are graphical objects such as pushbuttons, radio buttons, text fields, sliders, checkboxes, and menus. These interfaces allow users to perform sophisticated computations and plots by simply typing numbers into boxes, clicking on buttons, or by moving slidebars.

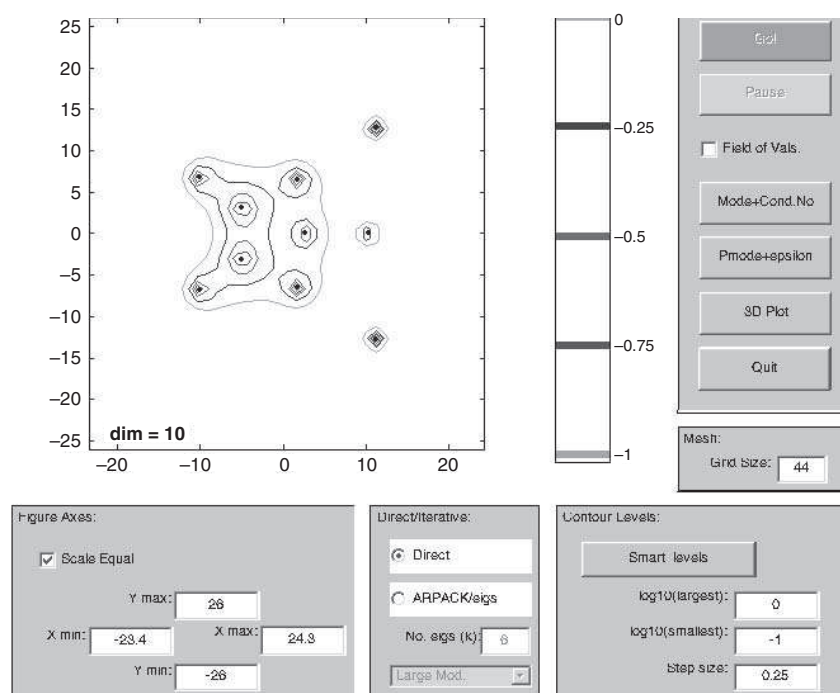


FIGURE 71.5 Eigtool GUI.

Commands:

1. The command guide opens up the MATLAB GUI Design Environment. This environment is essentially a GUI containing tools to facilitate the creation of new GUIs.

Examples:

1. Thomas G. Wright of Oxford University has developed a MATLAB GUI, eigtool, for computing eigenvalues, pseudospectra, and related quantities for nonsymmetric matrices, both dense and sparse. It allows the user to graphically visualize the pseudospectra and field of values of a matrix with just the click of a button.

The *epsilon-pseudospectrum* of a square matrix A is defined by

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} \mid z \in \sigma(A + E) \text{ for some } E \text{ with } \|E\| \leq \epsilon\}. \quad (71.2)$$

In Figure 71.5 the eigtool GUI is used to plot the ϵ -pseudospectra of a 10×10 matrix for $\epsilon = 10^{-k/4}$, $k = 0, 1, 2, 3, 4$.

For further information, see Chapter 16 and also references [Tre99] and [WT01].

2. The NSF-sponsored ATLAST Project has developed a large collection of MATLAB exercises, projects, and M-files for use in elementary linear algebra classes. (See [LHF03].) The ATLAST M-file collection contains a number of programs that make use of MATLAB's graphical user interface features to present user friendly tools for visualizing linear algebra. One example is the ATLAST cogame utility where students play a game to find linear combinations of two given vectors with the objective of obtaining a third vector that terminates at a given target point in the plane. Students can play the game at any one of four levels or play it competitively by selecting the two person game option. (See Figure 71.6.) At each step of the game a player must enter a pair of coordinates. MATLAB then plots the corresponding linear combination as a directed line segment. The game terminates when

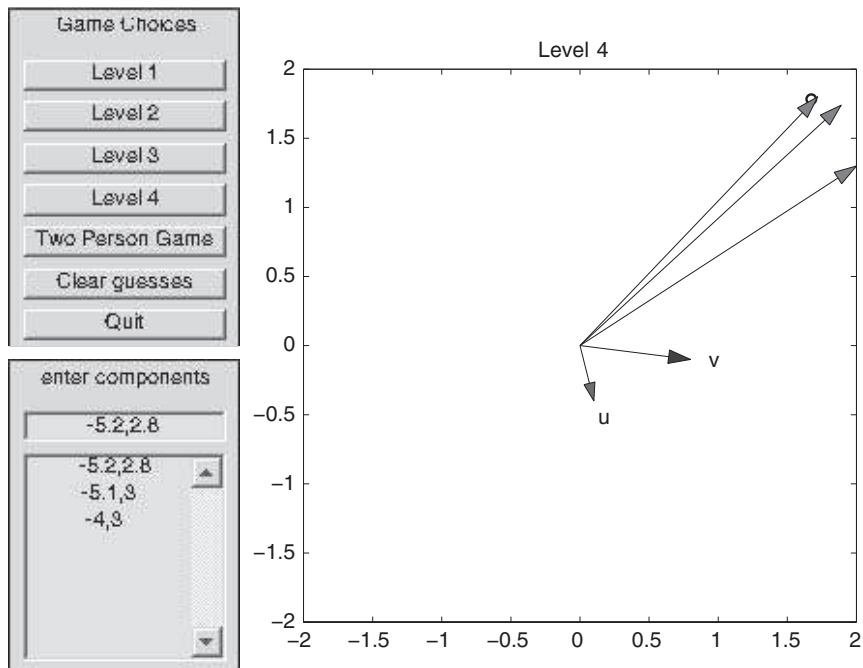


FIGURE 71.6 ATLAST Coordinate Game.

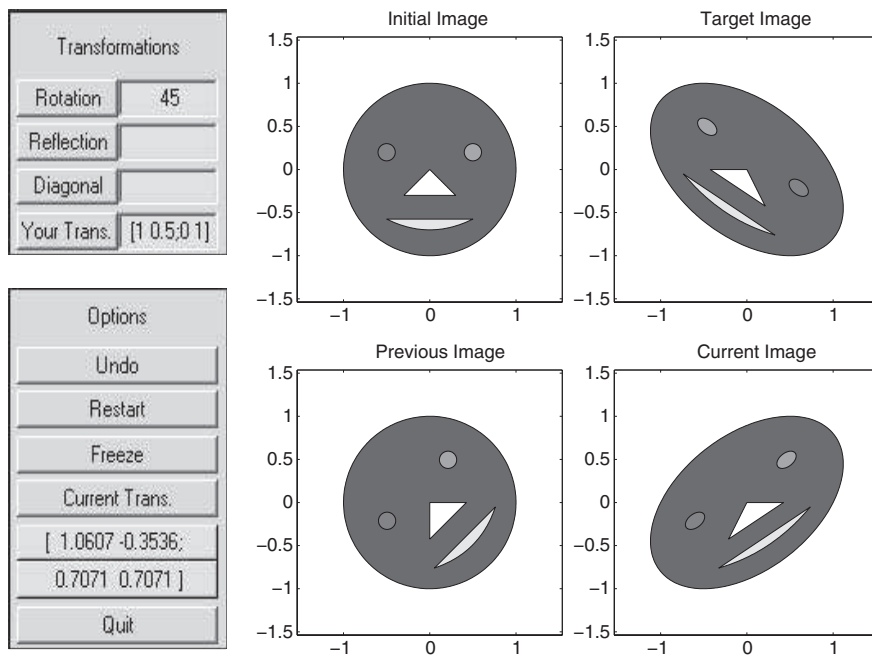


FIGURE 71.7 ATLAST Transformation Utility.

the tip of the plotted line segment lies in the small target circle. A running list of the coordinates entered in the game is displayed in the lower box to the left of the figure. The cogame GUI is useful for teaching lessons on the span of vectors in R^2 and for teaching about different bases for R^2 .

3. The ATLAST transform GUI helps students to visualize the effect of linear transformations on figures in the plane. With this utility students choose an image from a list of figures and then apply various transformations to the image. Each time a transformation is applied, the resulting image is shown in the current image window. The user can then click on the *current transformation* button to see the matrix representation of the transformation that maps the original image into the current image. In Figure 71.7 two transformations were applied to an initial image. First a 45° rotation was applied. Next a transformation matrix $[1, 0; 0.5, 1]$ was entered into the “Your Transformation” text field and the corresponding transformation was applied to the lower left image with the result being displayed in the Current Image window on the lower right. To transform the Current Image into the Target Image directly above it, one would need to apply a reflection transformation.

References

- [HH03] D.J. Higham and N.J. Higham. *MATLAB Guide, 2nd ed.* Philadelphia, PA.: SIAM, 2003.
- [HZ04] D.R. Hill and David E. Zitarelli. *Linear Algebra Labs with MATLAB, 3rd ed.* Upper Saddle River, NJ: Prentice Hall, 2004.
- [Leo06] S.J. Leon. *Linear Algebra with Applications, 7th ed.* Upper Saddle River, NJ: Prentice Hall, 2006.
- [LHF03] S.J. Leon, E. Herman, and R. Faulkenberry. *ATLAST Computer Exercises for Linear Algebra, 2nd ed.* Upper Saddle River, NJ: Prentice Hall, 2003.
- [Tre99] N.J. Trefethen. Computation of pseudospectra. *Acta Numerica*, 8, 247–295, 1999.
- [WT01] T.G. Wright and N.J. Trefethen. Large-scale computation of pseudospectra using ARPACK and eigs. *SIAM J. Sci. Comp.*, 23(2):591–605, 2001.