MATLAB ® / R Reference

February 11, 2008

David Hiebeler
Dept. of Mathematics and Statistics
University of Maine
Orono, ME 04469-5752
http://www.math.umaine.edu/faculty/hiebeler

I wrote the first version of this reference during the Spring 2007 semester, as I learned R while teaching my course "MAT400, Modeling & Simulation" at the University of Maine. The course covers population and epidemiological modeling, including deterministic and stochastic models in discrete and continuous time, along with spatial models. Half of the class meetings are in a regular classroom, and half are in a computer lab where students work through modeling & simulation exercises. When I taught earlier versions of the course, it was based on MATLAB only. In Spring 2007, some biology graduate students in the class who had learned R in statistics courses asked if they could use R in my class as well, and I said yes. My colleague Bill Halteman was a great help as I frantically learned R to stay ahead of the class. As I went, every time I learned how to do something in R for the course, I added it to this reference, so that I wouldn't forget it later. Some items took a huge amount of time searching for a simple way to do what I wanted, but at the end of the semester, I was pleasantly surprised that almost everything I do in MATLAB had an equivalent in R. I was also inspired to do this after seeing the "R for Octave Users" reference written by Robin Hankin.

This reference is organized into general categories. There is also a MATLAB index and an R index at the end, which should make it easy to look up a command you know in one of the languages and learn how to do it in the other (or if you're trying to read code in whichever language is unfamiliar to you, allow you to translate back to the one you are more familiar with). The index entries refer to the item numbers in the first column of the reference document, rather than page numbers.

Any corrections, suggested improvements, or even just notification that the reference has been useful will be appreciated. I hope all the time I spent on this will prove useful for others in addition to myself and my students. Note that sometimes I don't necessarily do things in what you may consider the "best" way in a particular language; I often tried to do things in a similar way in both languages. But if you believe you have a "better" way (either simpler, or more computationally efficient) to do something, feel free to let me know.

Acknowledgements: Thanks to Alan Cobo-Lewis and Isaac Michaud for correcting some errors, and Stephen Eglen for suggesting I include R's vectorized "`ifelse`".

Contents

1 Online help

No.	Description	MATLAB	R
1	Show help for a function (e.g. **sqrt**)	`help sqrt`, or `helpwin sqrt` to see it in a separate window	`help(sqrt)` or `?sqrt`
2	Show help for a built-in keyword (e.g. **for**)	`help for`	`help('for')` or `?'for'`
3	General list of many help topics	`help`	`library()` to see available libraries, or `library(help='base')` for very long list of stuff in base package which you can see help for
4	Explore main documentation in browser	`helpdesk`	`help.start()`
5	Search documentation for keyword or partial keyword (e.g. functions which refer to "binomial")	`lookfor binomial`	`help.search('binomial')`

2 Entering/building/indexing matrices

No.	Description	MATLAB	R
6	Enter a row vector $\vec{v} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$	`v=[1 2 3 4]`	`v=c(1,2,3,4)` or alternatively `v=scan()` then enter "1 2 3 4" and press Enter twice (the blank line terminates input)
7	Enter a column vector $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$	`[1; 2; 3; 4]`	`c(1,2,3,4)` (R does not distinguish between row and column vectors.)
8	Enter a matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	`[1 2 3 ; 4 5 6]`	To enter values by row: `matrix(c(1,2,3,4,5,6), nrow=2, byrow=TRUE)` To enter values by column: `matrix(c(1,4,2,5,3,6), nrow=2)`
9	Access an element of vector **v**	`v(3)`	`v[3]`
10	Access an element of matrix **A**	`A(2,3)`	`A[2,3]`
11	Access an element of matrix **A** using a single index: indices count down the first column, then down the second column, etc.	`A(5)`	`A[5]`
12	Build the vector [2 3 4 5 6 7]	`2:7`	`2:7`
13	Build the vector [7 6 5 4 3 2]	`7:-1:2`	`7:2`
14	Build the vector [2 5 8 11 14]	`2:3:14`	`seq(2,14,3)`
15	Build a vector containing n equally-spaced values between a and b inclusive	`linspace(a,b,n)`	`seq(a,b,length.out=n)` or just `seq(a,b,len=n)`
16	Build a vector of length k containing all zeros	`zeros(k,1)` (for a column vector) or `zeros(1,k)` (for a row vector)	`rep(0,k)`
17	Build a vector of length k containing the value j in all positions	`j*ones(k,1)` (for a column vector) or `j*ones(1,k)` (for a row vector)	`rep(j,k)`
18	Build an $m \times n$ matrix of zeros	`zeros(m,n)`	`matrix(0,nrow=m,ncol=n)` or just `matrix(0,m,n)`
19	Build an $m \times n$ matrix containing j in all positions	`j*ones(m,n)`	`matrix(j,nrow=m,ncol=n)` or just `matrix(j,m,n)`
20	$n \times n$ identity matrix I_n	`eye(n)`	`diag(n)`
21	"Glue" two matrices **a1** and **a2** (with the same number of rows) side-by-side	`[ a1 a2]`	`cbind(a1,a2)`
22	"Stack" two matrices **a1** and **a2** (with the same number of columns) on top of each other	`[ a1; a2]`	`rbind(a1,a2)`

No.	Description	MATLAB	R
23	Column 2 of matrix **A**	`A(:,2)`	`A[,2]`
24	Row 7 of matrix **A**	`A(7,:)`	`A[7,]`
25	All elements of **A** as a vector, column-by-column	`A(:)` (gives a column vector)	`c(A)`
26	Rows 2–4, columns 6–10 of **A** (this is a 3×5 matrix)	`A(2:4,6:10)`	`A[2:4,6:10]`
27	A 3×2 matrix consisting of rows 7, 7, and 6 and columns 2 and 1 of A (in that order)	`A([7 7 6], [2 1])`	`A[c(7,7,6),c(2,1)]`
28	Given a single index **ind** into an $m \times n$ matrix **A**, compute the row **r** and column **c** of that position (also works if **ind** is a vector)	`[r,c] = ind2sub(size(A), ind)`	`r = ((ind-1) %% m) + 1` `c = floor((ind-1) / m) + 1`
29	Given the row **r** and column **c** of an element of an $m \times n$ matrix **A**, compute the single index **ind** which can be used to access that element of **A** (also works if **r** and **c** are vectors)	`ind = sub2ind(size(A), r, c)`	`ind = (c-1)*m + r`
30	Given equal-sized vectors **r** and **c** (each of length k), set elements in rows and columns of matrix **A** equal to 12. That is, k elements of A will be modified.	`inds = sub2ind(size(A),r,c);` `A(inds) = 12;`	`inds = cbind(r,c)` `A[inds] = 12`

2.1 Cell arrays and lists

No.	Description	MATLAB	R

31	Build a vector **v** of length **n**, capable of containing different data types in different elements (called a *cell array* in MATLAB , and a *list* in R)	`v = cell(1,n)` In general, `cell(m,n)` makes an $m \times n$ cell array. Then you can do e.g.:
		`v{1} = 12`
		`v{2} = 'hi there'`
		`v{3} = rand(3)`
			`v = vector('list',n)` Then you can do e.g.:
			`v[[1]] = 12`
			`v[[2]] = 'hi there'`
			`v[[3]] = matrix(runif(9),3)`

32	Extract the i^{th} element of a cell/list vector **v**	`w = v{i}` If you use regular indexing, i.e. `w = v(i)`, then **w** will be a 1×1 cell matrix containing the contents of the i^{th} element of **v**.	`w = v[[i]]` If you use regular indexing, i.e. `w = v[i]`, then **w** will be a list of length 1 containing the contents of the i^{th} element of **v**.
33	Set the name of the i^{th} element in a list.	(MATLAB does not have names associated with elements of cell arrays.)	`names(v)[3] = 'myrandmatrix'` Use `names(v)` to see all names, and `names(v)=NULL` to clear all names.

3 Computations

3.1 Basic computations

No.	Description	MATLAB	R
34	$a+b$, $a-b$, ab, a/b	`a+b`, `a-b`, `a*b`, `a/b`	`a+b`, `a-b`, `a*b`, `a/b`
35	$\sqrt{a}$	`sqrt(a)`	`sqrt(a)`
36	a^b	`a^b`	`a^b`
37	$\|a\|$ (note: for complex arguments, this computes the modulus)	`abs(a)`	`abs(a)`
38	e^a	`exp(a)`	`exp(a)`
39	$\ln(a)$	`log(a)`	`log(a)`
40	$\log_2(a)$, $\log_{10}(a)$	`log2(a)`, `log10(a)`	`log2(a)`, `log10(a)`
41	$\sin(a)$, $\cos(a)$, $\tan(a)$	`sin(a)`, `cos(a)`, `tan(a)`	`sin(a)`, `cos(a)`, `tan(a)`
42	$\sin^{-1}(a)$, $\cos^{-1}(a)$, $\tan^{-1}(a)$	`asin(a)`, `acos(a)`, `atan(a)`	`asin(a)`, `acos(a)`, `atan(a)`
43	$\sinh(a)$, $\cosh(a)$, $\tanh(a)$	`sinh(a)`, `cosh(a)`, `tanh(a)`	`sinh(a)`, `cosh(a)`, `tanh(a)`
44	$\sinh^{-1}(a)$, $\cosh^{-1}(a)$, $\tanh^{-1}(a)$	`asinh(a)`, `acosh(a)`, `atanh(a)`	`asinh(a)`, `acosh(a)`, `atanh(a)`
45	n MOD k (modulo arithmetic)	`mod(n,k)`	`n %% k`
46	Round to nearest integer	`round(x)`	`round(x)` (Note: R uses IEC 60559 standard, rounding 5 to the even digit — so e.g. `round(0.5)` gives 0, not 1.
47	Round down to next lowest integer	`floor(x)`	`floor(x)`
48	Round up to next largest integer	`ceil(x)`	`ceiling(x)`

Note: the various functions above (logarithm, exponential, trig, abs, and rounding functions) all work with vectors and matrices, applying the function to each element, as well as with scalars.

3.2 Complex numbers

No.	Description	MATLAB	R
49	Enter a complex number	`1+2i`	`1+2i`
50	Modulus (magnitude)	`abs(z)`	`abs(z)` or `Mod(z)`
51	Argument (angle)	`angle(z)`	`Arg(z)`
52	Complex conjugate	`conj(z)`	`Conj(z)`
53	Real part of z	`real(z)`	`Re(z)`
54	Imaginary part of z	`imag(z)`	`Im(z)`

3.3 Matrix/vector computations

No.	Description	MATLAB	R
55	Matrix multiplication AB	`A * B`	`A %*% B`
56	Element-by-element multiplication of A and B	`A .* B`	`A * B`
57	Transpose matrix $\mathbf{A}$	`A'` (This is actually the complex conjugate transpose; use `A.'` for the non-conjugate transpose if you like; they are equivalent for real matrices.)	`t(A)`
58	Solve $A\vec{x} = \vec{b}$	`A\b` Warning: if there is no solution, MATLAB gives you a least-squares "best fit." If there are many solutions, MATLAB just gives you one of them.	`solve(A,b)` Warning: this only works with square invertible matrices.
59	Reduced echelon form of A	`rref(A)`	R does not have a function to do this
60	Compute inverse of $\mathbf{A}$	`inv(A)`	`solve(A)`
61	Compute AB^{-1}	`A/B`	`A %*% solve(B)`
62	Element-by-element division of A and B	`A ./ B`	`A / B`
63	Compute $A^{-1}B$	`A\B`	`solve(A) %*% B`
64	Square the matrix A	`A^2`	`A %*% A`
65	Raise matrix A to the k^{th} power	`A^k`	(No easy way to do this in R other than repeated multiplication `A %*% A %*% A...`)
66	Raise each element of A to the k^{th} power	`A.^k`	`A^k`
67	Set $\mathbf{w}$ to be a vector of eigenvalues of $\mathbf{A}$, and $\mathbf{V}$ a matrix containing the corresponding eigenvectors	`[V,D]=eig(A)` and then `w=diag(D)` since MATLAB returns the eigenvalues on the diagonal of $\mathbf{D}$	`tmp=eigen(A); w=tmp$values; V=tmp$vectors`
68	Compute mean of all elements in vector or matrix	`mean(v)` for vectors, `mean(A(:))` for matrices	`mean(v)` or `mean(A)`
69	Compute means of columns of a matrix	`mean(A)`	`colMeans(A)`
70	Compute means of rows of a matrix	`mean(A,2)`	`rowMeans(A)`
71	Compute standard deviation of all elements in vector or matrix	`std(v)` for vectors, `std(A(:))` for matrices. This normalizes by $n-1$. Use `std(v,1)` to normalize by n.	`sd(v)` or `sd(A)`. This normalizes by $n-1$.
72	Compute standard deviation of columns of a matrix	`std(A)`. This normalizes by $n-1$. Use `std(A,1)` to normalize by n	`apply(A,2,sd)`. This normalizes by $n-1$.
73	Compute standard deviation of rows of a matrix	`std(A,0,2)` to standardize by $n-1$, `std(A,1,2)` to standardize by n	`apply(A,1,sd)`. This normalizes by $n-1$.
74	Compute variance	`var(v)` (`var` works like `std`)	`var(v)` (`var` works like `sd`)
75	Compute sum of all elements in vector or matrix	`sum(v)` for vectors, `sum(A(:))` for matrices	`sum(v)` or `sum(A)`
76	Compute sums of columns of matrix	`sum(A)`	`colSums(A)`
77	Compute sums of rows of matrix	`sum(A,2)`	`rowSums(A)`

No.	Description	MATLAB	R
78	Compute matrix exponential $e^A = \sum_{k=0}^{\infty} A^k/k!$	`expm(A)`	`expm(Matrix(A))`, but this is part of the **Matrix** package which you'll need to install (see item 230 for how to install/load packages).
79	Compute cumulative sum of values in vector	`cumsum(v)`. Note `cumsum(A)` on array **A** will do cumulative sum of each column. Use `cumsum(A,2)` to do cumulative sum for each row, or `cumsum(A(:))` to operate over all elements in **A** (column-by-column)	`cumsum(v)`
80	Compute differences between consecutive elements of vector **v**. Result is a vector **w** 1 element shorter than **v**, where element i of **w** is element $i+1$ of **v** minus element i of **v**	`diff(v)`	`diff(v)`
81	Make a vector **y** the same size as vector **x**, which equals **4** everywhere that **x** is greater than 5, and equals 3 everywhere else (done via a vectorized computation).	`z = [3 4]; y = z((x > 5)+1)`	`y = ifelse(x > 5, 4, 3)`
82	Compute minimum of values in vector **v**	`min(v)`	`min(v)`
83	Compute minimum of all values in matrix **A**	`min(A(:))`	`min(A)`
84	Compute minimum value of each column of matrix **A**	`min(A)` (returns a row vector)	`apply(A,2,min)` (returns a vector)
85	Compute minimum value of each row of matrix **A**	`min(A, [ ], 2)` (returns a column vector)	`apply(A,1,min)` (returns a vector)
86	Given matrices **A** and **B**, compute a matrix where each element is the minimum of the corresponding elements of **A** and **B**	`min(A,B)`	`pmin(A,B)`
87	Given matrix **A** and scalar **c**, compute a matrix where each element is the minimum of **c** and the corresponding element of **A**	`min(A,c)`	`pmin(A,c)`
88	Find minimum among all values in matrices **A** and **B**	`min([A(:)  ; B(:)])`	`min(A,B)`
89	Find index of the first time `min(v)` appears in **v**, and store that index in **ind**	`[y,ind] = min(v)`	`ind = which.min(v)`

Notes:

- MATLAB and R both have a `max` function (and R has `pmax` and `which.max` as well) which behaves in the same ways as `min` but to compute maxima rather than minima.
- Functions like `exp`, `sin`, `sqrt` etc. will operate on arrays in both MATLAB and R, doing the computations for each element of the matrix.

No.	Description	MATLAB	R
90	Number of rows in A	`size(A,1)`	`nrow(A)`
91	Number of columns in A	`size(A,2)`	`ncol(A)`
92	Dimensions of A, listed in a vector	`size(A)`	`dim(A)`
93	Number of elements in vector **v**	`length(v)`	`length(v)`
94	Total number of elements in matrix A	`numel(A)`	`length(A)`
95	Max. dimension of A	`length(A)`	`max(dim(A))`
96	Sort values in vector **v**	`sort(v)`	`sort(v)`
97	Sort values in **v**, putting sorted values in **s**, and indices in **idx**, in the sense that **s**[**k**] = **x**[**idx**[**k**]]	`[s,idx]=sort(v)`	`tmp=sort(v,index.return=TRUE); s=tmp$x; idx=tmp$ix`
98	To count how many values in the vector **x** are between 4 and 7 (inclusive on the upper end)	`sum((x > 4) & (x <= 7))`	`sum((x > 4) & (x <= 7))`
99	Given vector **v**, return list of indices of elements of **v** which are greater than 5	`find(v > 5)`	`which(v > 5)`
100	Given matrix **A**, return list of indices of elements of **A** which are greater than 5, using single-indexing	`find(A > 5)`	`which(A > 5)`
101	Given matrix **A**, generate vectors **r** and **c** giving rows and columns of elements of **A** which are greater than 5	`[r,c] = find(A > 5)`	`w = which(A > 5, arr.ind=TRUE); r=w[,1]; c=w[,2]`

102	Given vector **x** (of presumably discrete values), build a vector **v** listing unique values in **x**, and corresponding vector **c** indicating how many times those values appear in **x**	`[v,c] = listvals(x)` (not standard in MATLAB ; I wrote this*). In standard MATLAB , this seems to work, although it's ugly:
		`[v,i,j]=unique(w);`
		`c = diff([0; find([diff( ...`
		`  sort(j(:))); 1])])`
			`w=table(x); c=as.numeric(w); v=as.numeric(names(w))`

103	Given vector **x** (of presumably continuous values), divide the range of values into k equally-sized bins, and build a vector **m** containing the midpoints of the bins and a corresponding vector **c** containing the counts of values in the bins	`[m,c] = bins(x,k)` (not standard in MATLAB ; I wrote this*). But see also `hist` and `histc` in standard MATLAB .	`w=hist(x,seq(min(x),max(x), length.out=k+1), plot=FALSE); m=w$mids; c=w$counts`

* The `listvals` and `bins` MATLAB functions are available on my web site at `http://www.math.umaine.edu/faculty/hiebeler/comp/matlabR.html`.

3.4 Function optimization/minimization

No.	Description	MATLAB	R
104	Find value m which minimizes a function $f(x)$ of one variable within the interval from a to b	Define function **f(x)**, then do `m = fminbnd(f, a, b)`	Define function **f(x)**, then do `m = optimize(f,c(a,b))$minimum`

105	Find value m which minimizes a function $f(x, p_1, p_2)$ with given extra parameters (but minimization is only occuring over the first argument), in the interval from a to b.	Define function **f(x,p1,p2)**, then use an "anonymous function":
		`% first define values for p1`
		`% and p2, and then do:`
		`m=fminbnd(@(x) f(x,p1,p2),a,b)`
			Define function **f(x,p1,p2)**, then:
			`# first define values for p1`
			`# and p2, and then do:`
			`m = optimize(f, c(a,b), p1=p1,`
			`p2=p2)$minimum`

106	Find values of x, y, z which minimize function $f(x, y, z)$, using a starting guess of $x = 1$, $y = 2.2$, and $z = 3.4$.	First write function **f(v)** which accepts a vector argument **v** containing values of x, y, and z, and returns the scalar value $f(x, y, z)$, then do: `fminsearch(@f,[1 2.2 3.4])`	First write function **f(v)** which accepts a vector argument **v** containing values of x, y, and z, and returns the scalar value $f(x, y, z)$, then do: `optim(c(1,2.2,3.4),f)$par`

107	Find values of x, y, z which minimize function $f(x, y, z, p_1, p_2)$, using a starting guess of $x = 1$, $y = 2.2$, and $z = 3.4$, where the function takes some extra parameters (useful e.g. for doing things like nonlinear least-squares optimization where you pass in some data vectors as extra parameters).	First write function **f(v,p1,p2)** which accepts a vector argument **v** containing values of x, y, and z, along with the extra parameters, and returns the scalar value $f(x, y, z, p_1, p_2)$, then do:
		`fminsearch(@f,[1 2.2 3.4], ...`
		`[ ], p1, p2)`
		Or use an anonymous function:
		`fminsearch(@(x) f(x,p1,p2), ...`
		`[1 2.2 3.4])`
			First write function **f(v,p1,p2)** which accepts a vector argument **v** containing values of x, y, and z, along with the extra parameters, and returns the scalar value $f(x, y, z, p_1, p_2)$, then do:
			`optim(c(1,2.2,3.4), f, p1=p1,`
			`p2=p2)$par`

3.5 Curve fitting

No.	Description	MATLAB	R
108	Fit the line $y = c_1x + c_0$ to data in vectors **x** and **y**.	`p = polyfit(x,y,1)` The return vector **p** has the coefficients in descending order, i.e. **p(1)** is c_1, and **p(2)** is c_0.	`p = coef(lm(y ~ x))` The return vector **p** has the coefficients in ascending order, i.e. **p[1]** is c_0, and **p[2]** is c_1.
109	Fit the quadratic polynomial $y = c_2x^2 + c_1x + c_0$ to data in vectors **x** and **y**.	`p = polyfit(x,y,2)` The return vector **p** has the coefficients in descending order, i.e. **p(1)** is c_2, **p(2)** is c_1, and **p(3)** is c_0.	`p = coef(lm(y ~ x + I(x^2)))` The return vector **p** has the coefficients in ascending order, i.e. **p[1]** is c_0, **p[2]** is c_1, and **p[3]** is c_2.
110	Fit n^{th} degree polynomial $y = c_nx^n + c_{n-1}x^{n-1} + \ldots + c_1x + c_0$ to data in vectors **x** and **y**.	`p = polyfit(x,y,n)` The return vector **p** has the coefficients in descending order, **p(1)** is c^n, **p(2)** is c^{n-1}, etc.	There isn't a simple function built into the standard R distribution to do this, but see the **polyreg** function in the **mda** package (see item 230 for how to install/load packages).
111	Fit the quadratic polynomial with zero intercept, $y = c_2x^2 + c_1x$ to data in vectors **x** and **y**.	(I don't know a simple way do this in MATLAB , other than to write a function which computes the sum of squared residuals and use **fminsearch** on that function. There is likely an easy way to do it in the Statistics Toolbox.)	`p=coef(lm(y ~ -1 + x + I(x^2)))` The return vector **p** has the coefficients in ascending order, i.e. **p[1]** is c_1, and **p[2]** is c_2.

4 Conditionals, control structure, loops

No.	Description	MATLAB	R

112	"for" loops over values in a vector **v** (the vector **v** is often constructed via **a:b**)	`for i=v`
		`  command1`
		`  command2`
		`end`
			If only one command inside the loop:
			`for (i in v)`
			`  command`
			or
			`for (i in v) command`
			If multiple commands inside the loop:
			`for (i in v) {`
			`  command1`
			`  command2`
			`}`

No.	Description	MATLAB	R

113	"if" statements with no else clause	`if cond`
		`  command1`
		`  command2`
		`end`
			If only one command inside the clause:
			`if (cond)`
			`  command`
			or
			`if (cond) command`
			If multiple commands:
			`if (cond) {`
			`  command1`
			`  command2`
			`}`

114	"if/else" statement	`if cond`
		`  command1`
		`  command2`
		`else`
		`  command3`
		`  command4`
		`end`
		Note: MATLAB also has an "**elseif**" statement, e.g.:
		`if cond1`
		`  command1`
		`elseif cond2`
		`  command2`
		`elseif cond3`
		`  command3`
		`else`
		`  command4`
		`end`
			If one command in clauses:
			`if (cond)`
			`  command1 else`
			`  command2`
			or
			`if (cond) cmd1 else cmd2`
			If multiple commands:
			`if (cond) {`
			`  command1`
			`  command2`
			`} else {`
			`  command3`
			`  command4`
			`}`
			Warning: the "else" must be on the same line as `command1` or the "}" (when typed interactively at the command prompt), otherwise R thinks the "if" statement was finished and gives an error.
			R does not have an "**elseif**" statement.

Logical comparisons which can be used on scalars in "`if`" statements, or which operate element-by-element on vectors/matrices:

MATLAB	R	Description
x < a	**x < a**	True if x is less than a
x > a	**x > a**	True if x is greater than a
x <= a	**x <= a**	True if x is less than or equal to a
x >= a	**x >= a**	True if x is greater than or equal to a
x == a	**x == a**	True if x is equal to a
x ~= a	**x != a**	True if x is not equal to a

Scalar logical operators:

Description	MATLAB	R
a AND b	`a && b`	`a && b`
a OR b	`a || b`	`a || b`
a XOR b	`xor(a,b)`	`xor(a,b)`
NOT a	`~a`	`!a`

The `&&` and `||` operators are short-circuiting, i.e. `&&` stops as soon as any of its terms are FALSE, and `||` stops as soon as any of its terms are TRUE.

Matrix logical operators (they operate element-by-element):

Description	MATLAB	R
a AND b	`a & b`	`a & b`
a OR b	`a | b`	`a | b`
a XOR b	`xor(a,b)`	`xor(a,b)`
NOT a	`~a`	`!a`

No.	Description	MATLAB	R
115	To test whether a scalar value **x** is between 4 and 7 (inclusive on the upper end)	`if ((x > 4) && (x <= 7))`	`if ((x > 4) && (x <= 7))`
116	To count how many values in the vector **x** are between 4 and 7 (inclusive on the upper end)	`sum((x > 4) & (x <= 7))`	`sum((x > 4) & (x <= 7))`
117	Test whether all values in a logical/boolean vector are TRUE	`all(v)`	`all(v)`
118	Test whether any values in a logical/boolean vector are TRUE	`any(v)`	`any(v)`

No.	Description	MATLAB	R

119	"while" statements to do iteration (useful when you don't know ahead of time how many iterations you'll need). E.g. to add uniform random numbers between 0 and 1 (and their squares) until their sum is greater than 20:	`mysum = 0;`
		`mysumsqr = 0;`
		`while (mysum < 20)`
		`  r = rand;`
		`  mysum = mysum + r;`
		`  mysumsqr = mysumsqr + r^2;`
		`end`
			`mysum = 0`
			`mysumsqr = 0`
			`while (mysum < 20) {`
			`  r = runif(1)`
			`  mysum = mysum + r`
			`  mysumsqr = mysumsqr + r^2`
			`}`
			
			(As with "if" statements and "for" loops, the curly brackets are not necessary if there's only one statement inside the "while" loop.)

No.	Description	MATLAB	R

120	"Switch" statements for integers	`switch (x)`
		`  case 10`
		`  disp('ten')`
		`  case {12,13}`
		`  disp('dozen (bakers?)')`
		`  otherwise`
		`  disp('unrecognized')`
		`end`
			R doesn't have a **switch** statement capable of doing this. It has a function which is fairly limited for integers, but can which do string matching. See `?switch` for more. But a basic example of what it can do for integers is below, showing that you can use it to return different expressions based on whether a value is $1, 2, \ldots$.
			`mystr = switch(x, 'one',`
			`  'two', 'three')`
			`print(mystr)`
			Note that **switch** returns NULL if **x** is larger than 3 in the above case. Also, continuous values of **x** will be truncated to integers.

5 Functions, ODEs

No.	Description	MATLAB	R

121	Implement a function **add(x,y)**	Put the following in **add.m**:
		`function retval=add(x,y)`
		`retval = x+y;`
		Then you can do e.g. `add(2,3)`
			Enter the following, or put it in a file and **source** that file:
			`add = function(x,y) {`
			`  return(x+y)`
			`}`
			Then you can do e.g. `add(2,3)`. Note, the curly brackets aren't needed if your function only has one line.

122	Numerically solve ODE $dx/dt = 5x$ from $t = 3$ to $t = 12$ with initial condition $x(3) = 7$	First implement function
		`function retval=f(t,x)`
		`retval = 5*x;`
		Then do `ode45(@f,[3,12],7)` to plot solution, or `[t,x]=ode45(@f,[3,12],7)` to get back vector **t** containing time values and vector **x** containing corresponding function values. If you want function values at specific times, e.g. $3, 3.1, 3.2, \ldots, 11.9, 12$, you can do `[t,x]=ode45(@f,3:0.1:12,7)`. Note: in older versions of MATLAB , use `'f'` instead of `@f`.
			First implement function
			`f = function(t,x,parms) {`
			`return(list(5*x))`
			`}`
			Then do `y=lsoda(7, seq(3,12, 0.1), f,NA)` to obtain solution values at times $3, 3.1, 3.2, \ldots, 11.9, 12$. The first column of **y**, namely **y[,1]** contains the time values; the second column **y[,2]** contains the corresponding function values. Note: **lsoda** is part of the **odesolve** package (see item 230 for how to install/load packages).

No.	Description	MATLAB	R

123	Numerically solve system of ODEs $dw/dt = 5w$, $dz/dt = 3w + 7z$ from $t = 3$ to $t = 12$ with initial conditions $w(3) = 7$, $z(3) = 8.2$	First implement function
		`function retval=myfunc(t,x)`
		`w = x(1);  z = x(2);`
		`retval = zeros(2,1);`
		`retval(1) = 5*w;`
		`retval(2) = 3*w + 7*z;`
		Then do `ode45(@myfunc,[3,12],[7; 8.2])` to plot solution, or `[t,x]=ode45(@myfunc,[3,12],[7; 8.2])` to get back vector **t** containing time values and matrix **x**, whose first column containing corresponding $w(t)$ values and second column contains $z(t)$ values. If you want function values at specific times, e.g. $3, 3.1, 3.2, \ldots, 11.9, 12$, you can do `[t,x]=ode45(@myfunc,3:0.1:12,[7; 8.2])`. Note: in older versions of MATLAB , use `'f'` instead of `@f`.
			First implement function
			`myfunc = function(t,x,parms) {`
			`w = x[1];  z = x[2];`
			`return(list(c(5*w, 3*w+7*z)))`
			`}`
			Then do `y=lsoda(c(7,8.2), seq(3,12, 0.1), myfunc,NA)` to obtain solution values at times $3, 3.1, 3.2, \ldots, 11.9, 12$. The first column of **y**, namely **y[,1]** contains the time values; the second column **y[,2]** contains the corresponding values of $w(t)$; and the third column contains $z(t)$. Note: **lsoda** is part of the **odesolve** package (see item 230 for how to install/load packages).

124	Pass parameters such as $r = 1.3$ and $K = 50$ to an ODE function from the command line, solving $dx/dt = rx(1 - x/K)$ from $t = 0$ to $t = 20$ with initial condition $x(0) = 2.5$.	First implement function
		`function retval=func2(t,x,r,K)`
		`retval = r*x*(1-x/K)`
		Then do `ode45(@func2,[0 20], 2.5, [ ], 1.3, 50)`. The empty matrix is necessary between the initial condition and the beginning of your extra parameters.
			First implement function
			`func2=function(t,x,parms) {`
			`r=parms[1];  K=parms[2]`
			`return(list(r*x*(1-x/K)))`
			`}`
			Then do
			`y=lsoda(2.5,seq(0,20,0.1) func2,c(1.3,50))`
			Note: **lsoda** is part of the **odesolve** package (see item 230 for how to install/load packages).

6 Probability and random values

No.	Description	MATLAB	R
125	Generate a continuous uniform random value between 0 and 1	`rand`	`runif(1)`
126	Generate vector of n uniform random vals between 0 and 1	`rand(n,1)` or `rand(1,n)`	`runif(n)`
127	Generate $m \times n$ matrix of uniform random values between 0 and 1	`rand(m,n)`	`matrix(runif(m*n),m,n)` or just `matrix(runif(m*n),m)`
128	Generate $m \times n$ matrix of continuous uniform random values between a and b	`a+rand(m,n)*(b-a)` or if you have the Statistics toolbox then `unifrnd(a,b,m,n)`	`matrix(runif(m*n,a,b),m)`
129	Generate a random integer between 1 and k	`floor(k*rand) + 1`	`floor(k*runif(1)) + 1` Note: `sample(k)[1]` would also work, but I believe in general will be less efficient, because that actually generates many random numbers and then just uses one of them.
130	Generate $m \times n$ matrix of discrete uniform random integers between 1 and k	`floor(k*rand(m,n))+1` or if you have the Statistics toolbox then `unidrnd(k,m,n)`	`floor(k*matrix(runif(m*n),m))+1`
131	Generate $m \times n$ matrix where each entry is 1 with probability p, otherwise is 0	`(rand(m,n)<p)*1` Note: multiplying by 1 turns the logical (true/false) result back into numeric values. You could also do `double(rand(m,n)<p)`	`(matrix(runif(m,n),m)<p)*1` (Note: multiplying by 1 turns the logical (true/false) result back into numeric values; using **as.numeric()** to do it would lose the shape of the matrix.)
132	Generate $m \times n$ matrix where each entry is a with probability p, otherwise is b	`b + (a-b)*(rand(m,n)<p)`	`b + (a-b)*(matrix( runif(m,n),m)<p)`
133	Generate a random integer between a and b inclusive	`floor((b-a+1)*rand)+a` or if you have the Statistics toolbox then `unidrnd(b-a+1)+a-1`	`floor((b-a+1)*runif(1))+a`

134	Flip a coin which comes up heads with probability p, and perform some action if it does come up heads	`if (rand < p)`
		`  ...some commands...`
		`end`
			`if (runif(1) < p) {`
			`  ...some commands...`
			`}`

135	Generate a random permutation of the integers $1, 2, \ldots, n$	`randperm(n)`	`sample(n)`
136	Generate a random selection of k unique integers between 1 and n	`[s,idx]=sort(rand(n,1));` `ri=idx(1:k)` or another way is `ri=randperm(n); ri=ri(1:k)`	`ri=sample(n,k)`
137	Set the random-number generator back to a known state (useful to do at the beginning of a stochastic simulation when debugging, so you'll get the same sequence of random numbers each time)	`rand('state', 12)`	`set.seed(12)`

Note that the "*rnd," "*pdf," and "*cdf" functions described below are all part of the MATLAB Statistics Toolbox, and not part of the core MATLAB distribution.

No.	Description	MATLAB	R
138	Generate a random value from the Binomial(n, p) distribution	`binornd(n,p)`	`rbinom(1,n,p)`
139	Generate a random value from the Poisson distribution with parameter λ	`poissrnd(lambda)`	`rpois(1,lambda)`
140	Generate a random value from the Exponential distribution with mean μ	`exprnd(mu)` or `-mu*log(rand)` will work even without the Statistics Toolbox.	`rexp(1, 1/mu)`
141	Generate a random value from the discrete uniform distribution on integers $1 \ldots k$	`unidrnd(k)` or `floor(rand*k)+1` will work even without the Statistics Toolbox.	`sample(k,1)`
142	Generate n iid random values from the discrete uniform distribution on integers $1 \ldots k$	`unidrnd(k,n,1)` or `floor(rand(n,1)*k)+1` will work even without the Statistics Toolbox.	`sample(k,n,replace=TRUE)`
143	Generate a random value from the continuous uniform distribution on the interval (a, b)	`unifrnd(a,b)` or `(b-a)*rand + a` will work even without the Statistics Toolbox.	`runif(1,a,b)`
144	Generate a random value from the normal distribution with mean mu and standard deviation σ	`normrnd(mu,sigma)` or `mu + sigma*randn` will work even without the Statistics Toolbox.	`rnorm(1,mu,sigma)`

Notes:

- The MATLAB "*rnd" functions above can all take additional **r,c** arguments to build an $r \times c$ matrix of iid random values. E.g. `poissrnd(3.5,4,7)` for a 4×7 matrix of iid values from the Poisson distribution with mean $\lambda = 3.5$. The `unidrnd(n,k,1)` command above is an example of this, to generate a $k \times 1$ column vector.
- The first parameter of the R "r*" functions above specifies how many values are desired. E.g. to generate 28 iid random values from a Poisson distribution with mean 3.5, use `rpois(28,3.5)`. To get a 4×7 matrix of such values, use `matrix(rpois(28,3.5),4)`.

No.	Description	MATLAB	R
145	Compute probability that a random variable from the Binomial(n, p) distribution has value **x** (i.e. the density, or pdf).	`binopdf(x,n,p)` or `nchoosek(n,x)*p^x*(1-p)^(n-x)` will work even without the Statistics Toolbox, as long as **n** and **x** are non-negative integers and $0 \leq$ **p** ≤ 1.	`dbinom(x,n,p)`
146	Compute probability that a random variable from the Poisson(λ) distribution has value **x**.	`poisspdf(x,lambda)` or `exp(-lambda)*lambda^x / factorial(x)` will work even without the Statistics Toolbox, as long as **x** is a non-negative integer and **lambda** ≥ 0.	`dpois(x,lambda)`
147	Compute probability density function at **x** for a random variable from the exponential distribution with mean μ.	`exppdf(x,mu)` or `(x>=0)*exp(-x/mu)/mu` will work even without the Statistics Toolbox, as long as **mu** is positive.	`dexp(x,1/mu)`

No.	Description	MATLAB	R
148	Compute probability density function at $\mathbf{x}$ for a random variable from the Normal distribution with mean μ and standard deviation σ.	`normpdf(x,mu,sigma)` or `exp(-(x-mu)^2/(2*sigma^2))/(sqrt(2*pi)*sigma)` will work even without the Statistics Toolbox.	`dnorm(x,mu,sigma)`
149	Compute probability density function at $\mathbf{x}$ for a random variable from the continuous uniform distribution on interval (a, b).	`unifpdf(x,a,b)` or `((x>=a)&&(x<=b))/(b-a)` will work even without the Statistics Toolbox.	`dunif(x,a,b)`
150	Compute probability that a random variable from the discrete uniform distribution on integers $1 \ldots n$ has value $\mathbf{x}$.	`unidpdf(x,n)` or `((x==floor(x)) && (x>=1)&&(x<=n))/n` will work even without the Statistics Toolbox, as long as $\mathbf{n}$ is a positive integer.	`((x==round(x)) && (x >= 1) && (x <= n))/n`

Note: one or more of the parameters in the above "*pdf" (MATLAB) or "d*" (R) functions can be vectors, but they must be the same size. Scalars are promoted to arrays of the appropriate size.

The corresponding CDF functions are below:

No.	Description	MATLAB	R
151	Compute probability that a random variable from the Binomial(n, p) distribution is less than or equal to **x** (i.e. the cumulative distribution function, or cdf).	`binocdf(x,n,p)`. Without the Statistics Toolbox, as long as **n** is a non-negative integer, this will work: `r = 0:floor(x); sum(factorial(n)./(factorial(r).*factorial(n-r)).*p.^r.*(1-p).^(n-r))`. (Unfortunately, MATLAB 's **nchoosek** function won't take a vector argument for **k**.)	`pbinom(x,n,p)`
152	Compute probability that a random variable from the Poisson(λ) distribution is less than or equal to **x**.	`poisscdf(x,lambda)`. Without the Statistics Toolbox, as long as **lambda** $\geq$ 0, this will work: `r = 0:floor(x); sum(exp(-lambda)*lambda.^r./factorial(r))`	`ppois(x,lambda)`
153	Compute cumulative distribution function at **x** for a random variable from the exponential distribution with mean μ.	`expcdf(x,mu)` or `(x>=0)*(1-exp(-x/mu))` will work even without the Statistics Toolbox, as long as **mu** is positive.	`pexp(x,1/mu)`
154	Compute cumulative distribution function at **x** for a random variable from the Normal distribution with mean μ and standard deviation σ.	`normcdf(x,mu,sigma)` or `1/2 - erf(-(x-mu)/(sigma*sqrt(2)))/2` will work even without the Statistics Toolbox, as long as **sigma** is positive.	`pnorm(x,mu,sigma)`
155	Compute cumulative distribution function at **x** for a random variable from the continuous uniform distribution on interval (a, b).	`unifcdf(x,a,b)` or `(x>a)*(min(x,b)-a)/(b-a)` will work even without the Statistics Toolbox, as long as **b** $>$ **a**.	`punif(x,a,b)`
156	Compute probability that a random variable from the discrete uniform distribution on integers $1 \ldots n$ is less than or equal to **x**.	`unidcdf(x,n)` or `(x>=1)*min(floor(x),n)/n` will work even without the Statistics Toolbox, as long as **n** is a positive integer.	`(x>=1)*min(floor(x),n)/n`

7 Graphics

7.1 Various types of plotting

No.	Description	MATLAB	R
157	Create a new figure window	`figure`	`windows()` (when running R in Windows), `quartz()` (in Mac OS-X), or `x11()` (in Linux)
158	Select figure number n	`figure(n)` (will create the figure if it doesn't exist)	`dev.set(n)` (returns the actual device selected; will be different from n if there is no figure device with number n)
159	List open figure windows	I don't know of a way to do this in MATLAB	`dev.list()`
160	Close figure window(s)	`close` to close the current figure window, `close(n)` to close a specified figure, and `close all` to close all figures	`dev.off()` to close the currently active figure device, `dev.off(n)` to close a specified one, and `graphics.off()` to close all figure devices.
161	Plot points using open circles	`plot(x,y,'o')`	`plot(x,y)`
162	Plot points using solid lines	`plot(x,y)`	`plot(x,y,type='l')` (Note: that's a lower-case 'L', not the number 1)
163	Plotting: color, point markers, linestyle	`plot(x,y,str)` where **str** is a string specifying color, point marker, and/or linestyle (see table below) (e.g. `'gs--'` for green squares with dashed line)	`plot(x,y,type=str1, pch=arg2,col=str3, lty=arg4)` See tables below for possible values of the 4 parameters
164	Plotting with logarithmic axes	`semilogx`, `semilogy`, and `loglog` functions take arguments like **plot**, and plot with logarithmic scales for x, y, and both axes, respectively	`plot(..., log='x')`, `plot(..., log='y')`, and `plot(..., log='xy')` plot with logarithmic scales for x, y, and both axes, respectively
165	Make bar graph where the x coordinates of the bars are in **x**, and their heights are in **y**	`bar(x,y)` Or just `bar(y)` if you only want to specify heights. Note: if A is a matrix, `bar(A)` interprets each column as a separate set of observations, and each row as a different observation within a set. So a 20×2 matrix is plotted as 2 sets of 20 observations, while a 2×20 matrix is plotted as 20 sets of 2 observations.	Can't do this in R; but `barplot(y)` makes a bar graph where you specify the heights, `barplot(y,w)` also specifies the widths of the bars, and `hist` can make plots like this too.
166	Make histogram of values in **x**	`hist(x)`	`hist(x)`
167	Given vector **x** containing integer values, make a bar graph where the x coordinates of bars are the values, and heights are the counts of how many times the values appear in **x**	`[v,c]=listvals(x); bar(v,c)` where **listvals** is a MATLAB function I wrote (available on my web page)	`hist(x,(min(x)-.5):(max(x)+.5))`

No.	Description	MATLAB	R
168	Given vector **x** containing continuous values, lump the data into k bins and make a histogram / bar graph of the binned data	`[m,c] = bins(x,k); bar(m,c)` where **bins** is a MATLAB function I wrote (available on my web page)	`hist(x,seq(min(x), max(x), length.out=k+1))`
169	Make a plot containing errorbars of height **s** above and below (x, y) points	`errorbar(x,y,s)`	`errbar(x,y,y+s,y-s)` Note: **errbar** is part of the **Hmisc** package (see item 230 for how to install/load packages).
170	Make a plot containing errorbars of height **a** above and **b** below (x, y) points	`errorbar(x,y,b,a)`	`errbar(x,y,y+a,y-b)` Note: **errbar** is part of the **Hmisc** package (see item 230 for how to install/load packages).
171	Other types of 2-D plots	`stem(x,y)` and `stairs(x,y)` for other types of 2-D plots. `polar(theta,r)` to use polar coordinates for plotting.	`pie(v)`
172	Make a 3-D plot of some data points with given x, y, z coordinates in the vectors **x**, **y**, and **z**.	`plot3(x,y,z)` This works much like **plot**, as far as plotting symbols, linetypes, and colors.	`cloud(z~x*y)` You can also use arguments **pch** and **col** as with `plot`. To make a 3-D plot with lines, do `cloud(z~x*y,type='l', panel.cloud=panel.3dwire)`
173	Surface plot of data in matrix **A**	`surf(A)` You can then click on the small curved arrow in the figure window (or choose "Rotate 3D" from the "Tools" menu), and then click and drag the mouse in the figure to rotate it in three dimensions.	`persp(A)` You can include shading in the image via e.g. `persp(A,shade=0.5)`. There are two viewing angles you can also specify, among other parameters, e.g. `persp(A, shade=0.5, theta=50, phi=35)`.

174	Surface plot of $f(x, y) = sin(x + y)\sqrt{y}$ for 100 values of x between 0 and 10, and 90 values of y between 2 and 8	`x = linspace(0,10,100);`
		`y = linspace(2,8,90);`
		`[X,Y] = meshgrid(x,y);`
		`Z = sin(X+Y).*sqrt(Y);`
		`surf(X,Y,Z)`
		`shading flat`
			`x = seq(0,10,100)`
			`y = seq(2,8,90)`
			`f = function(x,y)`
			`  return(sin(x+y)*sqrt(y))`
			`z = outer(x,y,f)`
			`persp(x,y,z)`

175	Other ways of plotting the data from the previous command	`mesh(X,Y,Z)`, `surfc(X,Y,Z)`, `surfl(X,Y,Z)`, `contour(X,Y,Z)`, `pcolor(X,Y,Z)`, `waterfall(X,Y,Z)`. Also see the `slice` command.	`contour(x,y,z)` Or do `s=expand.grid(x=x,y=y)`, and then `wireframe(z~x*y,s)` or `wireframe(z~x*y,s,shade=TRUE)` (Note: **wireframe** is part of the **lattice** package; see item 230 for how to load packages). If you have vectors **x**, **y**, and **z** all the same length, you can also do `symbols(x,y,z)`.

Adding various labels or making adjustments to plots

No.	Description	MATLAB	R
176	Set axis ranges in a figure window	`axis([x1 x2 y1 y2])`	You have to do this when you make the plot, e.g. `plot(x,y,xlim=c(x1,x2), ylim=c(y1,y2))`
177	Add title to plot	`title('somestring')`	`title(main='somestring')` adds a main title, `title(sub='somestring')` adds a subtitle. You can also include **main=** and **sub=** arguments in a **plot** command.
178	Add axis labels to plot	`xlabel('somestring')` and `ylabel('somestring')`	`title(xlab='somestring', ylab='anotherstr')`. You can also include **xlab=** and **ylab=** arguments in a **plot** command.
179	Add grid lines to plot	`grid on` (and `grid off` to turn off)	`grid()` Note that if you'll be printing the plot, the default style for grid-lines is to use gray dotted lines, which are almost invisible on some printers. You may want to do e.g. `grid(lty='dashed', col='black')` to use black dashed lines which are easier to see.
180	Add figure legend to top-left corner of plot	`legend('first', 'second', 'Location', 'NorthWest')`	`legend('topleft', legend=c('first', 'second'), col=c('red', 'blue'), pch=c('*','o'))`

MATLAB note: sometimes you build a graph piece-by-piece, and then want to manually add a legend which doesn't correspond with the order you put things in the plot. You can manually construct a legend by plotting "invisible" things, then building the legend using them. E.g. to make a legend with black stars and solid lines, and red circles and dashed lines: `h1=plot(0,0,'k*-'); set(h1,'Visible', 'off'); h2=plot(0,0,'k*-'); set(h2,'Visible', 'off'); legend([h1 h2], 'blah, 'whoa')`. Just be sure to choose coordinates for your "invisible" points within the current figure's axis ranges.

No.	Description	MATLAB	R
181	Adding more things to a figure	`hold on` means everything plotted from now on in that figure window is added to what's already there. `hold off` turns it off. `clf` clears the figure and turns off hold.	`points(...)` and `lines(...)` work like **plot**, but add to what's already in the figure rather than clearing the figure first. **points** and **lines** are basically identical, just with different default plotting styles. Note: axes are not recalculated/redrawn when adding more things to a figure.
182	Plot multiple data sets at once	`plot(x,y)` where **x** and **y** are 2-D matrices. Each column of **x** is plotted against the corresponding column of **y**. If **x** has only one column, it will be re-used.	`matplot(x,y)` where **x** and **y** are 2-D matrices. Each column of **x** is plotted against the corresponding column of **y**. If **x** has only one column, it will be re-used.
183	Plot $\sin(2x)$ for x between 7 and 18	`fplot('sin(2*x)', [7 18])`	`curve(sin(2*x), 7, 18, 200)` makes the plot, by sampling the value of the function at 200 values between 7 and 18 (if you don't specify the number of points, 101 is the default). You could do this manually yourself via commands like `tmpx=seq(7,18,200); plot(tmpx, sin(2*tmpx))`.
184	Plot color image of integer values in matrix **A**	`image(A)` to use array values as raw indices into colormap, or `imagesc(A)` to automatically scale values first (these both draw row 1 of the matrix at the top of the image); or `pcolor(A)` (draws row 1 of the matrix at the bottom of the image). After using `pcolor`, try the commands `shading flat` or `shading interp`.	`image(A)` (it rotates the matrix 90 degrees counterclockwise: it draws row 1 of A as the left column of the image, and column 1 of A as the bottom row of the image, so the row number is the x coord and column number is the y coord). It also rescales colors. If you are using a colormap with k entries, but the value k does not appear in A, use `image(A,zlim=c(1,k))` to avoid rescaling of colors. Or e.g. `image(A,zlim=c(0,k-1))` if you want values 0 through $k-1$ to be plotted using the k colors.
185	Add colorbar legend to image plot	`colorbar`, after using `image` or `pcolor`.	Use `filled.contour(A)` rather than `image(A)`, although it "blurs" the data via interpolation, or use `levelplot(A)` from the **lattice** package (see item 230 for how to load packages). To use a colormap with the latter, do e.g. `levelplot(A,col.regions= terrain.colors(100))`.
186	Set colormap in image	`colormap(hot)`. Instead of `hot`, you can also use `gray`, `flag`, `jet` (the default), `cool`, `bone`, `copper`, `pink`, `hsv`, `prism`. By default, the length of the new colormap is the same as the currently-installed one; use e.g. `colormap(hot(256))` to specify the number of entries.	`image(A,col=terrain.colors(100))`. The parameter 100 specifies the length of the colormap. Other colormaps are `heat.colors()`, `topo.colors()`, and `cm.colors()`.

No.	Description	MATLAB	R
187	Build your own colormap using Red/Green/Blue triplets	Use an $n \times 3$ matrix; each row gives R,G,B intensities between 0 and 1. Can use as argument with **colormap**. E.g. for 2 colors: `mycmap = [0.5 0.8 0.2 ; 0.2 0.2 0.7]`	Use a vector of hexadecimal strings, each beginning with '`#`' and giving R,G,B intensities between 00 and FF. E.g. `c('#80CC33','#3333B3')`; can use as argument to **col=** parameter to `image`. You can build such a vector of strings from vectors of Red, Green, and Blue intensities (each between 0 and 1) as follows (for a 2-color example): `r=c(0.5,0.2); g=c(0.8,0.2); b=c(0.2,0.7); mycolors=rgb(r,g,b)`.

MATLAB plotting specifications, for use with `plot`, `fplot`, `semilogx`, `semilogy`, `loglog`, etc:

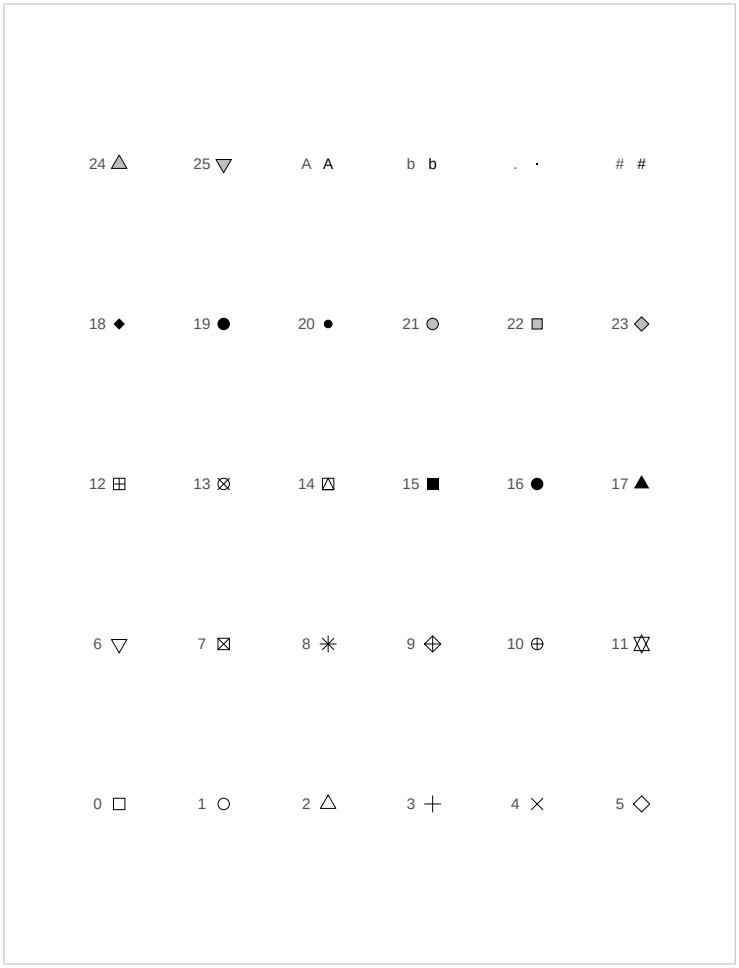
Symbol	Color	Symbol	Marker	Symbol	Linestyle
`b`	blue	`.`	point (.)	`-`	solid line
`g`	green	`o`	circle ($\circ$)	`:`	dotted line
`r`	red	`x`	cross ($\times$)	`-.`	dash-dot line
`c`	cyan	`+`	plus sign ($+$)	`--`	dashed line
`m`	magenta	`*`	asterisk ($*$)		
`y`	yellow	`s`	square ($\square$)		
`k`	black	`d`	diamond ($\Diamond$)		
`w`	white	`v`	triangle (down) ($\triangledown$)		
		`^`	triangle (up) ($\triangle$)		
		`<`	triangle (left) ($\triangleleft$)		
		`>`	triangle (right) ($\triangleright$)		
		`p`	pentragram star		
		`h`	hexagram star		

R plotting specifications for **col** (color), **pch** (plotting character), and **type** arguments, for use with `plot`, `matplot`, `points`, and `lines`:

`col`	Description	`pch`	Description	`type`	Description
`'blue'`	Blue	`'a'`	a (similarly for other characters, but see **'.'** below for an exception	`p`	points
`'green'`	Green	`19`	solid circle	`l`	lines
`'red'`	Red	`20`	bullet (smaller circle)	`b`	both
`'cyan'`	Cyan	`21`	open circle	`c`	lines part only of "`b`"
`'magenta'`	Magenta	`22`	square	`o`	lines, points overplotted
`'yellow'`	Yellow	`23`	diamond	`h`	histogram-like lines
`'black'`	Black	`24`	triangle point-up	`s`	steps
`'#RRGGBB'`	hexadecimal specification of Red, Green, Blue	`25`	triangle point-down	`S`	another kind of steps
(Other names)	See `colors()` for list of available color names.	`'.'`	rectangle of size 0.01 inch, 1 pixel, or 1 point (1/72 inch) depending on device	`n`	no plotting
			(See table on next page for more)		

R plotting specifications for **lty** (line-type) argument, for use with `plot`, `matplot`, `points`, and `lines`:

`lty`	Description
0	blank
1	solid
2	dashed
3	dotted
4	dotdash
5	longdash
6	twodash



R plotting characters, i.e. values for **pch** argument (from the book *R Graphics*, by Paul Murrell, Chapman & Hall / CRC, 2006)

No.	Description	MATLAB	R
188	Divide up a figure window into smaller sub-figures	`subplot(m,n,k)` divides the current figure window into an $m \times n$ array of subplots, and draws in subplot number k as numbered in "reading order," i.e. left-to-right, top-to-bottom. E.g. `subplot(2,3,4)` selects the first sub-figure in the second row of a 2×3 array of sub-figures. You can do more complex things, e.g. `subplot(5,5,[1 2 6 7])` selects the first two subplots in the first row, and first two subplots in the second row, i.e. gives you a bigger subplot within a 5×5 array of subplots. (If you that command followed by e.g. `subplot(5,5,3)` you'll see what's meant by that.)	There are several ways to do this, e.g. using `layout` or `split.screen`, although they aren't quite as friendly as MATLAB 's. E.g. if you let $A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, then `layout(A)` will divide the figure into 6 sub-figures: you can imagine the figure divide into a 3×3 matrix of smaller blocks; sub-figure 1 will take up the upper-left 2×2 portion, and sub-figures 2–6 will take up smaller portions, according to the positions of those numbers in the matrix A. Consecutive plotting commands will draw into successive sub-figures; there doesn't seem to be a way to explicitly specify which sub-figure to draw into next. To use `split.screen`, you can do e.g. `split.screen(c(2,1))` to split into a 2×1 matrix of sub-figures (numbered 1 and 2). Then `split.screen(c(1,3),2)` splits sub-figure 2 into a 1×3 matrix of smaller sub-figures (numbered 3, 4, and 5). `screen(4)` will then select sub-figure number 4, and subsequent plotting commands will draw into it. A third way to accomplish this is via the commands `par(mfrow=)` or `par(mfcol=)` to split the figure window, and `par(mfg=)` to select which sub-figure to draw into. Note that the above methods are all incompatible with each other.
189	Force graphics windows to update	`drawnow` (MATLAB normally only updates figure windows when a script/function finishes and returns control to the MATLAB prompt, or under a couple of other circumstances. This forces it to update figure windows to reflect any recent plotting commands.)	R automatically updates graphics windows even before functions/scripts finish executing, so it's not necessary to explictly request it.

7.2 Printing/saving graphics

No.	Description	MATLAB	R
190	To print/save to a PDF file named **fname.pdf**	`print -dpdf fname` saves the contents of currently active figure window	First do `pdf('fname.pdf')`. Then, do various plotting commands to make your image, as if you were plotting in a window. Finally, do `dev.off()` to close/save the PDF file. To print the contents of the active figure window, do `dev.copy(device=pdf, file='fname.pdf'); dev.off()`. (But this will not work if you've turned off the display list via `dev.control(displaylist='inhibit')`.)
191	To print/save to a PostScript file **fname.ps** or **fname.eps**	`print -dps fname` for black & white PostScript; `print -dpsc fname` for color PostScript; `print -deps fname` for black & white Encapsulated PostScript; `print -depsc fname` for color Encapsulated PostScript. The first two save to **fname.ps**, while the latter two save to **fname.eps**.	`postscript('fname.eps')`, followed by your plotting commands, followed by `dev.off()` to close/save the file. Note: you may want to use `postscript('fname.eps', horizontal=FALSE)` to save your figure in portrait mode rather than the default landscape mode. To print the contents of the active figure window, do `dev.copy(device=postscript, file='fname.eps'); dev.off()`. (But this will not work if you've turned off the display list via `dev.control(displaylist='inhibit')`.) You can also include the `horizontal=FALSE` argument with `dev.copy()`.
192	To print/save to a JPEG file **fname.jpg** with jpeg quality = 90 (higher quality looks better but makes the file larger)	`print -djpeg90 fname`	`jpeg('fname.jpg',quality=90)`, followed by your plotting commands, followed by `dev.off()` to close/save the file.

7.3 Animating cellular automata / lattice simulations

No.	Description	MATLAB	R
193	To display images of cellular automata or other lattice simulations while running in real time	Repeatedly use either `pcolor` or `image` to display the data. Don't forget to call `drawnow` as well, otherwise the figure window will not be updated with each image.	If you simply call `image` repeatedly, there is a great deal of flickering/flashing. To avoid this, after drawing the image for the first time using e.g. `image(A)`, from then on only use `image(A,add=TRUE)`, which avoids redrawing the entire image (and the associated flicker). However, this will soon consume a great deal of memory, as all drawn images are saved in the image buffer. There are two solutions to that problem: (1) every k time steps, leave off the "`add=TRUE`" argument to flush the image buffer (and get occasional flickering), where you choose k to balance the flickering vs. memory-usage tradeoff; or (2) after drawing the first image, do `dev.control(displaylist='inhibit')` to prohibit retaining the data. However, the latter solution means that after the simulation is done, the figure window will not be redrawn if it is resized, or temporarily obscured by another window. (A call to `dev.control(displaylist='enable')` and then one final `image(A)` at the end of the simulation will re-enable re-drawing after resizing or obscuring, without consuming extra memory.)

8 Working with files

No.	Description	MATLAB	R
194	Create a folder (also known as a "directory")	`mkdir dirname`	`dir.create('dirname')`
195	Set/change working directory	`cd dirname`	`setwd('dirname')`
196	See list of files in current working directory	`dir`	`dir()`
197	Run commands in file 'foo.m' or 'foo.R' respectively	`foo`	`source('foo.R')`
198	Read data from text file "data.txt" into matrix A	`A=load('data.txt')`	`A=as.matrix(read.table('data.txt'))`
199	Write data from matrix A into text file "data.txt"	`save data.txt A -ascii`	`write(A, file='data.txt', ncolumn=dim(A)[2])`

9 Misc

9.1 Variables

No.	Description	MATLAB	R
200	Assigning to variables	`x = 5`	`x <- 5` or `x = 5`
201	Short list of defined variables	`who`	`ls()`
202	Long list of defined variables	`whos`	`ls.str()`
203	See detailed info about the variable **ab**	`whos ab`	`str(ab)`
204	See detailed info about all variables with "ab" in their name	`whos *ab*`	`ls.str(pattern='ab')`
205	Clear one variable	`clear x`	`rm(x)`
206	Clear two variables	`clear x y`	`rm(x,y)`
207	Clear all variables	`clear all`	`rm(list=ls())`
208	See what type of object **x** is	`class(x)`	`class(x)`
209	(Variable names)	Variable names must begin with a letter, but after that they may contain any combination of letters, digits, and the underscore character. Names are case-sensitive.	Variable names may contain letters, digits, the period, and the underscore character. They cannot begin with a digit or underscore, or with a period followed by a digit. Names are case-sensitive.

9.2 Misc

No.	Description	MATLAB	R

210	Line continuation	If you want to break up a MATLAB command over more than one line, end all but the last line with three periods: "...". E.g.:
		`x = 3 + ...`
		`  4`
			In R, you can spread commands out over multiple lines, and nothing extra is necessary. R will continue reading input until the command is complete. E.g.:
			`x = 3 +`
			`  4`

211	Controlling formatting of output	`format short g` and `format long g` are handy; see `help format`	`options(digits=6)` tells R you'd like to use 6 digits of precision in values it displays (it is only a suggestion, not strictly followed)
212	Exit the program	`quit` or `exit`	`q()`
213	Comments	`% this is a comment`	`# this is a comment`
214	Print a string	`disp('hi there')`	`print('hi there')`
215	Print a string containing single quotes	`disp('It''s nice')`	`print('It\'s nice')` or `print("It's nice")`
216	Give prompt and read input from user	`x = input('Enter data:')`	`print('Enter data:')` `x = scan()`
217	Concatenate strings	`['two hal' 'ves']`	`paste('two hal', 'ves', sep='')` Note: this will not combine strings which are in different elements within a vector. E.g. `v=c('two hal', 'ves')` followed by `paste(v, sep='')` will not combine the strings into a single string. For that, do `paste(v, collapse='')`.
218	Convert number to string	`num2str(x)`	`as.character(x)`
219	Pause for x seconds	`pause(x)`	`Sys.sleep(x)`

220	Use **sprintf** to create a formatted string. Use **%d** for integers ("d" stands for "decimal", i.e. base 10), **%f** for floating-point numbers, **%e** for scientific-notation floating point, **%g** to automatically choose **%e** or **%f** based on the value. You can specify field-widths/precisions, e.g. **%5d** for integers with padding to 5 spaces, or **%.7f** for floating-point with 7 digits of precision. There are many other options too; see the docs.	`x=2; y=3.5;`
		`s=sprintf('x is %d, y=%g', ...`
		`  x, y)`
			`x=2; y=3.5`
			`s=sprintf('x is %d, y is %g',`
			`  x, y)`

221	Wait for user to press any key	`pause`	Don't know of a way to do this in R, but `scan(quiet=TRUE)` will wait until the user presses the Enter key

No.	Description	MATLAB	R
222	Measure CPU time used to do some commands	`t1=cputime; ...commands... ; cputime-t1`	`t1=proc.time(); ...commands... ; (proc.time()-t1)[1]`
223	Measure elapsed ("wall-clock") time used to do some commands	`tic; ...commands... ; toc` or `t1=clock; ...commands... ; etime(clock,t1)`	`t1=proc.time(); ...commands... ; (proc.time()-t1)[3]`
224	Print an error message an interrupt execution	`error('Problem!')`	`stop('Problem!')`
225	Print a warning message	`warning('Smaller problem!')`	`warning('Smaller problem!')`
226	Putting multiple statements on one line	Separate statements by commas or semicolons. A semicolon at the end of a statement suppresses display of the results (also useful even with just a single statement on a line), while a comma does not.	Separate statements by semicolons.
227	Evaluate contents of a string `s` as command(s).	`eval(s)`	`eval(parse(text=s))`
228	Show where a command is	`which sqrt` shows you where the file defining the **sqrt** function is (but note that many basic functions are "built in," so the MATLAB function file is really just a stub containing documentation). This is useful if a command is doing something strange, e.g. **sqrt** isn't working. If you've accidentally defined a *variable* called **sqrt**, then `which sqrt` will tell you, so you can `clear sqrt` to erase it so that you can go back to using the *function* **sqrt**.	R does not execute commands directly from files, so there is no equivalent command.
229	Query/set the search path.	`path` displays the current search path (the list of places MATLAB searches for commands you enter). To add a directory `~/foo` to the beginning of the search path, do `addpath ~/foo -begin` or to add it to the end of the path, do `addpath ~/foo -end` (Note: you should generally add the full path of a directory, i.e. in Linux or Mac OS-X something like `~/foo` as above or of the form **/usr/local/lib/foo**, while under Windows it would be something like **C:/foo**)	R does not use a search path to look for files.

No.	Description	MATLAB	R
230	Install and load a package.	MATLAB does not have packages. It has toolboxes, which you can purchase and install. "Contributed" code (written by end users) can simply be downloaded and put in a directory which you then add to MATLAB's path (see item 229 for how to add things to MATLAB's path).	To install e.g. the **odesolve** package, you can use the command `install.packages('odesolve')`. You then need to load the package in order to use it, via the command `library('odesolve')`. When running R again later you'll need to load the package again to use it, but you should not need to re-install it. Note that the **lattice** package is typically included with binary distributions of R, so it only needs to be loaded, not installed.

10 Spatial Modeling

No.	Description	MATLAB	R
231	Take an L×L matrix **A** of 0s and 1s, and "seed" fraction p of the 0s (turn them into 1s), not changing entries which are already 1.	`A = (A | (rand(L) < p))*1;`	`A = (A | (matrix(runif(L^2),L) < p))*1`
232	Take an $L \times L$ matrix **A** of 0s and 1s, and "kill" fraction p of the 1s (turn them into 0s), not changing the rest of the entries	`A = (A & (rand(L) < 1-p))*1;`	`A = (A & (matrix(runif(L^2),L) < 1-p))*1`
233	Do "wraparound" on a coordinate **newx** that you've already calculated. You can replace **newx** with **x+dx** if you want to do wraparound on an offset x coordinate.	`mod(newx-1,L)+1` Note: for portability with other languages such as C which handle MOD of negative values differently, you may want to get in the habit of instead doing `mod(newx-1+L,L)+1`	`((newx-1) %% L) + 1` Note: for portability with other languages such as C which handle MOD of negative values differently, you may want to get in the habit of instead doing `((newx-1+L)%%L) + 1`
234	Randomly initialize a portion of an array: set fraction p of sites in rows **iy1** through **iy2** and columns **ix1** through **ix2** equal to 1 (and set the rest of the sites in that block equal to zero). Note: this assume **iy1 < iy2** and **ix1 < ix2**.	`dx=ix2-ix1+1; dy=iy2-iy1+1; A(iy1:iy2,ix1:ix2) = ... (rand(dy,dx) < p0)*1;`	`dx=ix2-ix1+1; dy=iy2-iy1+1; A[iy1:iy2,ix1:ix2] = (matrix(runif(dy*dx),dy) < p0)*1`

Index of MATLAB commands and concepts

Index of R commands and concepts