INSTRUCTIONS FOR LA 1.01

LA is a calculator-like program for manipulating matrices and vectors. The user tells the program what to do by typing commands when prompted by the computer. Once the commands have been learnt, the program is very convenient to use.

In this guide, examples appear in `typewriter text`. What you type appears underlined.

USING LA

LA is written using the Free Pascal Compiler and should run on any system where that compiler can be implemented. Currently it runs on Windows systems (32- or 64-bit) and on Apple Mac systems running OS X.

For Windows systems, install the file `la.exe` in any suitable directory (folder). Click on the file to run the program; it will open in its own window. If the installation directory is on your system path you can also start the program in a command prompt window by typing `la` in that window.

For Apple Mac systems, rename the executable file to `la` and install it in a suitable directory on your system path. You may then run the program by typing `la` in a Terminal window.

HELP

Commands in LA are divided into four main groups: (1) general commands, (2) commands for manipulating matrices, (3) commands for operating on the internal structure of a single matrix, and (4) commands for manipulating vectors. Help messages are available listing the commands in each of these four groups, by typing

```
LA >> h general
LA >> h matrix
LA >> h single
LA >> h vector
```

There is also a help message available describing the special matrices which LA makes available to the user (see below). To get this message, type

```
LA >> h special
```

MATRICES AND VECTORS

LA can store up to 30 matrices and/or vectors. Vectors are treated as just $1 \times n$ matrices - they are stored in the same place as the matrices and can be used in any command as if they were a $1 \times n$ matrix. Henceforth, the term 'matrix' will include vectors as a special case. The maximum dimensions of a matrix are 25×25, and so the maximum length of a vector is 25.

Matrix and vector names

The 32 available matrices can be identified by names of the user's choosing. Each name must begin with a letter (a to z and A to Z) but the remaining characters can be any non-space characters. The maximum length of a name is 15 characters: longer names will be truncated. Names are the ONLY place where LA differentiates between upper and lower case, so that `aa`, `AA` and `Aa` all refer to different matrices.

LA also has some built-in special matrices, such as identity matrices, zero matrices and a few others, which you can use in commands. These special matrices are given names beginning with a dollar sign. The available special matrices are:

`$e`n`,`i	$1 \times n$ row vector, the ith standard basis vector in real n-space, with a one in column i and zeroes elsewhere
`$h`n	$n \times n$ Hilbert matrix
`$i`n	$n \times n$ identity matrix
`$j`m`,`n	$m \times n$ matrix of all ones – `$j`n is short for `$j`n`,`n
`$s`n`,`i	$n \times 1$ standard basis vector, like `$e`n`,`i except it is a column vector instead of a row vector

`$z`m`,`n $m \times n$ matrix of all zeroes – `$z`n is short for `$z`n`,`n

For example, `$i5` (or `$I5`) means a 5×5 identity matrix.

The current matrix or vector

LA has the concept of a 'current matrix.' This concept is mostly used with the 'single matrix' commands, where you are working with one matrix at a time, but there is normally a current matrix no matter what command you are currently using. Usually the current matrix is the matrix which was produced as a result of the last operation you did. For example, if you typed

```
LA >> mm A $j4 C
```

to multiply matrix A by a 4×4 matrix of all ones to get matrix C, then C would become the current matrix if the multiplication was successful. You can refer to the current matrix in commands as `.` (a period), so that, for example,

```
LA >> cr . b f
```

puts the cross product of the current matrix (remember that vectors are just special matrices!) and b in f.

There are two occasions when no current matrix will be defined: when LA first starts up, and when you have killed the last current matrix using the `k` command. If there is no current matrix and you try to use it, you will receive an error message: `No current matrix` if you try to use it implicitly (in a single matrix command or in a `p` command without an argument), or `Illegal matrix/vector` if you try to use an explicit `..` The current matrix will become defined again as soon as you do something which stores a result in a matrix or vector.

Source and target matrices and vectors

Matrices in commands can be divided in two classes: sources and targets. A source matrix is a matrix which is used as input by a command; a target matrix is used to store the result of a command. A source matrix can be a matrix name, a period (for the current matrix) or a special matrix like `$i6`. A target matrix can only be a matrix name or a period. For example, in

```
LA >> mm A $j3,2 D
```

which multiplies A by a 3×2 matrix of all ones to get D, the source matrices are A and `$j3,2`, and the target matrix is D. In the list of commands below, source matrices are indicated by s and target matrices by t.

Note on vectors

The fact that vectors are represented as $1 \times n$ matrices, or row vectors, rather than as $n \times 1$ matrices, or column vectors, needs to be remembered. Thus, if A is a 3×3 matrix and e is a 3-vector, then you cannot multiply A by e (in that order) because e is 1×3, not 3×1. To do the required multiplication you would first have to transpose e. Thus, you might issue the following commands

```
LA >> t e e
LA >> mm A e g
```

which put A times e in g. Also, all vector operations work with row vectors. Thus, if c and d are 3×1 matrices, you would have to type

```
LA >> t c c
LA >> t d d
LA >> cr c d e
```

to calculate their cross product e: c and d have to be transposed using the `t` command before e can be calculated - and e will be 1×3, not 3×1.

NUMBERS

Rationals

All matrix and vector entries in LA are rational numbers. Rational numbers are also required in certain commands – this is indicated in the list of commands below by the letter q. The following forms are legal:

{+/-}ℓ	e.g. `3`, `+72`, `-346`, `0`
{+/-}m/n	e.g. `3/4`, `+17/6`, `-12/439`
{+/-}ℓ_m/n	e.g. `13_1/4`, `+3_4/5`, `-13_3/147`

where ℓ, m and n are unsigned integers. Rational numbers cannot contain spaces.

Rational number arithmetic can quickly lead to large integers in the numerator and denominator. This may result in an overflow, because the computer can only store integers of limited size. LA tries to catch problems of this sort. If you receive a message of the form

```
Overflow:  ...  <> ...  - approx to ...
```

then you should assume that the result of your calculation is junk.

Integers

Certain commands require integers as part of their arguments - this is indicated in the list of commands below by the letter n. In all cases the integers are unsigned; in other words, `2` is legal but `+2` and `-2` will not be accepted.

ROWS AND COLUMNS

Many single matrix commands require you to specify rows or columns of the matrix you are working on. A row is specified by an expression of the form `r`n, and a column by an expression of the form `c`n - for example, `r4` means row 4 and `c5` means column 5. You cannot put a space between the `r` or `c` and the number. In the list of commands below, r means a row expression is required, c means a column expression is needed, and p means that either a row or a column expression is necessary.

OUTPUT FILE

LA (as of 0.31) now has the capability to copy its matrices and vectors into an output file. To open and close the output file, use the `fi` command, and to print matrices to the file use the `np` command. See the command list below for details.

As of 0.32, there is also a `tp` command, which prints to the output file in Plain TEX format, so that users can easily include matrices produced by LA in their TEX files.

COMMANDS

Here is a summary of the types of command argument used in the list of commands below:

n	unsigned integer
q	rational number
s	source matrix or vector
t	target matrix or vector
r	row
c	column
p	row or column

General commands

`c` s t

Copy source s to target t. For example,

```
LA >> c $i6 B
```

copies a 6×6 identity matrix into B.

`fi` {*filename*}

Opens and/or closes output file. First, the current output file (if any) is closed. Then, if a filename is specified, that file is opened as the new output file. Any previous contents in the file are destroyed. When LA starts up there is no output file. For example,

```
LA >> fi b:ques1.out
```

opens the file `ques1.out` on your B disk, and then later

```
LA >> fi
```

would close it.

`h` {*topic*}

Prints help messages on given topics. Valid topics are `general`, `matrix`, `single`, `vector` and `special`. If you give no topic, or an invalid topic, you will be given a list of topics.

`k` t_1 t_2 ...

Kill listed matrices: their contents are destroyed. For example,

```
LA >> k . D E
```

erases the current matrix and matrices D and E.

`ls`

List all matrices. Lists the names and dimensions of all matrices. The current matrix is indicated by an arrow `>` pointing to it.

`np` t_1 t_2 ...

Print listed matrices to output file in normal format. Operation is the same as the `p` command below, except that the matrices are printed in the current output file instead of on the screen.

`p` t_1 t_2 ...

Print listed matrices. Prints the given matrices. If there is no current matrix, then the last matrix printed becomes the current matrix. If no matrices are given, just prints the current matrix. Thus, if there is no current matrix, then

```
LA >> p A B C
```

prints A, then B, then C, and C becomes the current matrix. If the current matrix is F, then

```
LA >> p
```

just prints F and

```
LA >> p A . B
```

prints A, then F, then B (and the current matrix is still F).

`q`

Quit. Exits from LA. The program will also stop if the user types Control-Z (the end of file character).

`r` n t

Read a row vector. Reads a row vector of length n into target t. The user will be prompted for the elements of the vector, which should be rational numbers in one of the forms mentioned above. The entries should be separated by spaces and/or tabs, and need not all appear on the same line. For example,

```
LA >> r 8 A
Enter vector:  1_1/2 -3/2   5/3 0
7 -6 -5_1/2 18
```

makes A the 1×8 vector $(3/2, -3/2, 5/3, 0, 7, -6, -11/2, 18)$. If you type an illegal character (not part of a number) when typing your vector entries, LA will ignore anything that follows the illegal character, and fill out the rest of the vector with zeroes. Thus,

```
LA >> r 7 A
Enter vector:  1 2 # 4 5 6 7
```

will result in A being the vector (1, 2, 0, 0, 0, 0, 0), and you will receive an `End of line ignored` message. (This can actually be a useful feature if you have a vector with a lot of zeroes at the end.)

`r` n_1 n_2 t

Read a matrix. Reads an $n_1 \times n_2$ matrix into target t. The user will be prompted for the elements of the matrix, which should be entered row by row. The elements should be legal rational numbers, separated by spaces and/or tabs, on one or more lines. It is NOT necessary to enter the numbers so that each row is on a separate line. For example,

```
LA >> r 3 4 B
Enter matrix:
1 2 -3/2
0 4 6 -9/7 5 0 8 8_1/3 9
```

makes B the matrix

```
B:
      1    2    -3/2    0
      4    6    -9/7    5
      0    8    25/3    9
```

As with reading a vector, typing an illegal character will cause LA to ignore all later entries, and fill the remainder of the matrix with zeroes.

`rc` n t

Read a column vector. Reads a column vector of length n into target t. The user will be prompted for the elements of the vector, which should be rational numbers in one of the forms mentioned above. The entries should be separated by spaces and/or tabs, and need not all appear on the same line. For example,

```
LA >> rc 3 c
Enter vector:  1_1/2 -3/2 5/3
```

makes c the 3×1 vector

```
c:
        3/2
       -3/2
        5/3
```

Behaviour when illegal characters are encountered is the same as for the `r` command above: the remainder of the vector is filled with zeroes.

`rn` t_1 t_2

Rename the matrix (or vector) t_1 to t_2. There must currently be no other matrix named t_2. For example,

```
LA >> rn A A1
```

renames the matrix A to A1.

`se` t

Select t as the current matrix. For example,

```
LA >> se G
LA >> p
```

would make G the current matrix, so that the `p` command would print G.

`tp` t_1 t_2 ...

Print listed matrices to output file in Plain TEX format, for inclusion in a TEX file. Apart from the format used, operation is the same as the `p` and `np` commands.

`vf` {+/-}

Turn verification on (`+`) or off (`-`). Verification is a feature which prints the current matrix after most commands, to show you the result of your command. For example, if verification is on, and you type

```
LA >> c $e2,1 A
```

then the program would print

```
A:
                1      0
```

to show you the result. Typing `vf +` turns this on, `vf -` turns it off, and just typing `vf` changes it to the reverse of what it currently is. Notice that there must be a space between the `vf` and the `+` or `-`. When LA starts up, verification is on.

`w` n

Change column width to n. When LA prints matrices or vectors, it tries to print them so that all columns are the same width, but this may not be possible if the numbers involved are too large. To get around this problem, the width of the columns can be controlled using this command. For example,

```
LA >> w 15
```

changes the width of printed columns to 15. Just typing `w` returns the width to the value it has when LA starts up, namely 7. The number n must be between 1 and 40 inclusive.

`$` *system-command*

A dollar sign followed by a space indicates that the remainder of the line is to be treated as a command to your operating system. This allows the user to issue operating system commands, such as `dir`, from inside LA. For example,

```
LA >> $ dir
```

`$$` *command-interpreter-path*

The double dollar command tells LA where your command interpreter lives. LA needs this information in order to implement the `$` command, above. The default setting is `cmd.exe` (without an explicit path), which should work on most Windows systems. However, you may have to tell LA where your command interpreter lives; for example,

```
LA >> $$ c:\windows\system32\cmd.exe
Command interpreter is c:\windows\system32\cmd.exe
```

The `$$` command with no argument will tell you where LA is currently trying to find your command interpreter.

Single matrix commands

All of the commands in this section operate on the current matrix.

`a` q p_1 p_2

Add q times p_1 to p_2. If q is omitted it is assumed to be 1, and if q is given as just `+` or `-` it is assumed to be 1 or -1 respectively. Here p_1 and p_2 must be of the same type: both rows, or both columns. For example,

```
LA >> a 3/2 r1 r2
```

adds 3/2 times row 1 to row 2 of the current matrix, and

```
LA >> a - c1 c4
```

adds -1 times column 1 to column 4, while

```
LA >> a 2 r1 c2
Cannot mix rows and columns
```

produces an error message.

`ar` p_1 p_2 `...`

Create a new matrix by arranging specified rows and columns of the current matrix. This command is very versatile: for example, it lets you form a submatrix of a matrix, permute the rows or columns of a matrix, or duplicate some of the rows or columns of a matrix. If rows r_1, r_2, r_3, ..., r_k and columns c_1, c_2, c_3, ..., c_ℓ are specified (in that order) then the new matrix formed has in its i, j entry the entry which was in row r_i and column c_j of the original matrix. If no rows are given, then the original rows are used in their original order, and similarly for the columns. Suppose for example that the current matrix is a 4×5 matrix. Then the command

```
LA >> ar r1 r2 c2 c4 c5
```

would form the submatrix consisting of rows 1 and 2 and columns 2, 4 and 5 of the original matrix. The command

```
LA >> ar r3 r4 r2 r1
```

would rearrange the rows of the matrix, making row 3 the new first row, row 4 the new second row, row 2 the new third row, and row 1 the new fourth row. The command

```
LA >> ar c1 c1 c1 c2 c3 c4 c5
```

would add two extra copies of column 1 to the beginning of the matrix. If an illegal row or column is specified, the whole command will have no effect.

`d` p q

Divide p by q. For example,

```
LA >> d r4 -5/2
```

divides row 4 by $-5/2$, and

```
LA >> d c5 17
```

divides column 5 by 17.

`de` p_1 p_2 `...`

Delete specified rows and columns from current matrix. For example,

```
LA >> de r1 c3 r5 c2 c5
```

deletes rows 1 and 5 and columns 2, 3 and 5. If an illegal row or column is given, the whole command will have no effect.

`e` q r c

Enter the value q in position r, c. This command is often used to edit entries of a matrix. For example, if you mistype the entry in row 2, column 3 of matrix A as 0 instead of 10, you could fix it as follows:

```
LA >> r 5 4 A
Enter matrix:
...
LA >> e 10 r2 c3
```

The column specification can come before the row specification, and if there is only one row or column, the row or column (respectively) need not be specified.

`gp` r c

Pivot on entry at position r, c, using a 'lower pivot' as employed by Gaussian elimination. Tells program to use elementary row operations to make entry at r, c equal to 1, and then make all entries in column c below the pivot position equal to 0. Rows above the pivot row remain unchanged. The entry at position r, c must be nonzero. Example:

```
LA >> gp c3 r4
```

As shown in this example, the row need not come before the column. Compare with `pi` command, below.

`m` p q

Multiply p by q. Similar to `d` command above. Note that q is not allowed to be zero, because then this would not be an elementary row or column operation. Use the `ze` command below to make a row or column all zero.

`pi` r c

Pivot on entry at position r, c, using a 'full pivot' as employed by Gauss-Jordan elimination. Tells program to use elementary row operations to make entry at r, c equal to 1, and then make all other entries in column c equal to 0. The entry at position r, c must be nonzero. Useful for simplex algorithm. Example:

```
LA >> pi r3 c2
```

The row need not come before the column. Compare with `gp` command, above.

`rp` p_1 p_2 ...

Replace specified rows and columns, reading new values from the keyboard. The user will first be prompted for the rows specified, and then for the columns specified. Note that if both rows and columns are specified, then the value at the intersection of one of the specified rows and one of the specified columns will be taken from the specification for the column, overriding the value specified for the row if it is different. For example, if we have a 3×4 matrix:

```
LA >> rp c4 r1 c2
Enter new row 1:  1 2 5 -3
Enter new column 2:  2 4 6
Enter new column 4:  7 5 1
```

The value in row 1, column 4 ends up as 7, not as -3. As with the `r` and `rc` commands, extra numbers at the end of a row or column are ignored, and illegal characters result in the rest of the row or column being filled with zeroes.

`s` q p_1 p_2

Subtract q times p_1 from p_2. Same comments apply as for `a` command above.

`sa` t

Save a copy of the current matrix in t. This is NOT the same as '`c . t`' because '`c . t`' changes the current matrix to t, while '`sa t`' leaves the current matrix unchanged. For example, if the current matrix is E then

```
LA >> sa B
```

copies E to B and leaves E as the current matrix.

`un`

Undo the last single matrix command. Will work provided there has been no intervening command which could affect either the identity or contents of the current matrix (for example, a command which stores results in a target matrix). Example:

```
LA >> p
a:
      1    2    3
LA >> de c2
a:
      1    3
LA >> un
a:
      1    2    3
```

`x` p_1 p_2

Exchange p_1 and p_2. Here p_1 and p_2 must be the same type: both rows or both columns. For example,

```
LA >> x r3 r2
```

exchanges rows 2 and 3 of the current matrix.

`ze` p_1 p_2 ...

Zero out specified rows and columns. Overwrites the given rows and columns with zeroes. For example,

```
LA >> ze c3 r4 r6 c1
```

makes everything in rows 4 and 6 and columns 1 and 3 equal to zero. If an illegal row or column is specified, the whole command will have no effect.

Matrix commands

`am` q_1 s_1 q_2 s_2 t

Add q_1 times s_1 to q_2 times s_2 and put the result in t. If one of q_1 or q_2 is left out then it is assumed to be 1. If one of q_1 or q_2 is given as just a sign, `+` or `-`, then it is assumed to be 1 or -1 respectively. For example,

```
LA >> am 3 A - B C
```

puts C equal to 3A−B.

`cp` s

Find the characteristic polynomial of s. The matrix s must be square. Example:

```
LA >> cp $i3
Characteristic polynomial is x^3 - 3 x^2 + 3 x - 1
```

This command is currently implemented using the fact that the coefficient of x^i in the characteristic polynomial of an $n \times n$ matrix is $(-1)^{n-i}$ times the sum of the determinants of all the principal submatrices of side $n - i$. Thus, it requires $2^n - 1$ determinant evaluations. THIS COMMAND MAKE TAKE A LONG TIME FOR LARGE MATRICES.

`dt` s

Find the determinant of s. The matrix s must be square. Example:

```
LA >> dt $i4
Determinant is 1
```

`ga` s t

Put row echelon form of s in t, i.e. apply Gaussian elimination to s and store the result in t. For example,

```
LA >> ga A AR
```

reduces the matrix A to row echelon form, using Gaussian elimination, and stores the result in the matrix AR. Note that the row echelon form of a matrix is not in general unique. Compare to `gj` command, below.

`gi` s t

Find a general inverse of s and put in t. A general inverse of a matrix A is a matrix G so that GA = R, where R is the reduced row echelon form of A. It is useful for solving equations of the form Ax = b: these can be reduced to the form Rx = Gb which is easy to solve. If A is square and invertible then G is just the ordinary inverse of A. Example:

```
LA >> gi . B
```

A matrix may have more than one general inverse.

`gj` s t

Put reduced row echelon form of s in t, i.e. apply Gauss-Jordan elimination to s and store the result in t. For example,

```
LA >> gj . .
```

reduces the current matrix to reduced row echelon form, using Gauss-Jordan elimination. The reduced row echelon form of a matrix is unique. Compare to `ga` command, above.

`i` s t

Find inverse of s and put in t. The matrix s must be square. LA will tell you if s is not invertible; otherwise its inverse will be stored in t. Example:

```
LA >> i A B
Not invertible
```

`jh` s_1 s_2 t

This joins s_1 and s_2 horizontally to form a new matrix, which is stored in t. Obviously, s_1 and s_2 must have the same number of rows. For example, if A is

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

then the command

```
LA >> jh A $i2 B
```

results in B being the matrix

```
B:
      1   2   3   1   0
      4   5   6   0   1
```

`jv` s_1 s_2 t

This joins s_1 and s_2 vertically to form a new matrix, which is stored in t. The matrices s_1 and s_2 must have the same number of columns. For example,

```
LA >> jv $j1,3 $z2,3 X
```

results in X being the matrix

```
X:
      1   1   1
      0   0   0
      0   0   0
```

`mm` s q t

Multiply s by q and put result in t. For example,

```
LA >> mm A 2/3 B
```

makes B = (2/3)A.

`mm` s_1 s_2 t

Multiply s_1 by s_2 and put result in t. For example,

```
LA >> mm A G D
```

makes D = AG.

`nu` s t

Find a basis for the nullspace of s, i.e. a basis for the set of vectors $\{\mathrm{x} : s\mathrm{x} = 0\}$, and put this basis into the matrix t as its columns. The elements of the nullspace are then exactly the vectors of the form tx, where x is a vector with the same number of rows as t has columns. Example:

```
LA >> nu A K
```

`rk` s

Find rank of s. Example:

```
LA >> rk A
Rank is 3
```

`so` t s_1 s_2 t
`so` s_1 t s_2 t

Solve matrix equations of the form $t\ s_1 = s_2$ or $s_1\ t = s_2$ for t. For example,

```
LA >> so A x b x
```

means to solve Ax=b for x, and

```
LA >> so Y C D Y
```

means to solve YC=D for Y. You will be told whether there are no, one, or many solutions. In the case where there are many solutions only one will be given, but you can use the nullspace command `nu` above to work out the complete solution space.

`t` s t

Put the transpose of s in t. For example

```
LA >> t $e3,3 A
```

makes A a 3×1 column vector corresponding to the unit vector k (`$e3,3`).

Vector commands

`av` q_1 s_1 q_2 s_2 t

Adds q_1 times s_1 to q_2 times s_2 to get t. Same as the matrix command `am`: see above.

`cr` s_1 s_2 t

Put the cross product of s_1 and s_2 in t. Both s_1 and s_2 must be 3-vectors, i.e 1×3 matrices. For example,

```
LA >> cr a g h
```

makes h = a × g.

`do` s_1 s_2

Calculate the dot product of s_1 and s_2. Example:

```
LA >> do a $e4,3
Dot product is -7/8
```

`mv` s q t

Multiply s by q to get t. Same as the matrix command `mm`: see above.

`oc` s_1 s_2 t

Put part of s_1 orthogonal to s_2 in t. The vector t is also known as the 'vector component of s_1 orthogonal to s_2.' Note that s_2 cannot be the zero vector. Example:

```
LA >> oc $j1,3 a h
```

`pc` s_1 s_2 t

Put part of s_1 parallel to s_2 into t. t is also known as the 'vector component of s_1 parallel to s_2' or as the 'orthogonal projection of s_1 onto s_2.' As in the `oc` command, s_2 cannot be the zero vector. For example,

```
LA >> pc . $e3,2 .
```

replaces the current vector by its projection onto the unit vector `j` (`$e3,2`).

COPYRIGHT NOTICE