One-parameter Families of Algebraic Curves in Matlab

Rory Mulvaney
with
Prof. Joel Roberts and Prof. Victor Reiner

University of Minnesota REU

August 3, 2000
Contents

0.1 Abstract ................................................................. 2
0.2 Package Setup .......................................................... 2
0.3 Custom Functions ....................................................... 3
   0.3.1 aPlot ................................................................. 3
   0.3.2 cone ................................................................. 4
   0.3.3 envelope ............................................................. 6
   0.3.4 fCone ................................................................. 7
   0.3.5 famEnv ................................................................. 8
   0.3.6 family ................................................................. 9
   0.3.7 makeMovie ............................................................ 10
   0.3.8 playMovie ............................................................. 11
   0.3.9 readMovie ........................................................... 12
   0.3.10 replace ............................................................... 13
   0.3.11 writeMovie .......................................................... 14
0.4 General Matlab Guide ................................................... 15
   0.4.1 Matrix and String Manipulation ................................. 15
   0.4.2 Maple and Symbolic Variables ................................. 16
   0.4.3 Example Matlab Function ....................................... 18
   0.4.4 Helpful Matlab Functions ....................................... 19
0.1 Abstract

This report documents the use of some Matlab functions developed to easily visualize and work with attributes of algebraic curves and one-parameter families of algebraic curves. Some attributes addressed include the singular points of a curve, and the envelope of a family of curves.

0.2 Package Setup

This package consists of custom Matlab functions in m-files (with a “.m” extension). In order to conveniently use these functions with Matlab, you should put the m-files in a special directory, and set an environment variable that tells Matlab where to look for the custom m-files. Follow the steps below for a standard setup (you don’t have to type after the #-signs):

Download the mfiles.tar file and place it in your home directory.

```
% cd
% mkdir MyMatlabDir # use any name you like for this directory
% cp mfiles.tar MyMatlabDir
% cd MyMatlabDir
% tar xvf mfiles.tar # this extracts the m-files into a subdirectory called mfiles
% cd
```

Next, define the environment variable that tells Matlab where to look for your m-files. If you are using a bash shell (this is probably the case, unless you’ve changed it), add the following line to your .bashrc file in your home directory (using pico, emacs, vi, or any editor).

```
export MATLABPATH="'echo $HOME'/MyMatlabDir/mfiles"
```

Note the single quotes above are backward single quotes, and remember to change MyMatlabDir to the directory you created above. Since the .bashrc script is run automatically only when a new terminal is opened, for the current terminal session the change only takes effect if you type in the line above at the command prompt. With that done, from now on, just type “matlab” to start Matlab with easy access to the mfiles in MyMatlabDir/mfiles.
0.3 Custom Functions

0.3.1 aPlot

**Description** - Plots a 2-d implicit equation.

**Syntax**

\[
[C \ h] = \text{aPlot}(f, [\text{xmin xmax ymin ymax}], \text{gridPoints})
\]

**Parameters/Return Values**

- \(f\) - is an expression in \(x\) and \(y\), and the equation \(f = 0\) is plotted
- \([\text{xmin xmax ymin ymax}]\) - indicates the range to be plotted over
- \(\text{gridPoints}\) - scalar or 1x2 vector specifying the number of grid coordinates in the \(x\) & \(y\) directions (higher for a more precise plot)
- \(\text{C}\) - contour matrix in \(C\) (see help contour) and a column vector of handles to the lines plotted
- \(\text{h}\) - column vector of handles to the lines plotted

**Example**

To plot \(y = x^2\), try

\[
[c \ handles] = \text{aPlot}('x^2 - y', [-2, 2 -.4, 5], 50)
\]

set(handles, 'color', 'r') \% changes color of lines to red

![Plot](image.png)

**Figure 1:** Plot produced by aPlot

**Algorithm**

Uses meshgrid to create an \(xGridPoints \times yGridPoints\) matrix of points, calculates the value of \(f\) at each point, and uses contour to plot only the points where \(f = 0\).

**See Also** - contour, meshgrid, ezplot
0.3.2 cone

Description - Plots the multiple tangent lines at the singularities of a two-dimensional algebraic curve.

Syntax

\[
\text{[singPts initialForms handles]} = \text{cone}(f, \text{[xmin xmax ymin ymax]}, \text{gridPoints})
\]

Parameters/Return Values

\( f \) - an expression in \( x \) and \( y \), the singular points are found for the equation \( f = 0 \), and the multiple tangent lines are plotted at these points

\( \text{gridPoints} \) - scalar or \( 1 \times 2 \) vector specifying the number of grid coordinates in the \( x \) & \( y \) directions (higher for a more precise plot)

\( pts \) - array of the singular points, accessed by \( pts(i).x, pts(i).y \)

\( \text{initialForms} \) - cell array of strings representing the unfactored initial form at each singularity - accessed by \( \text{char(initialForms}(i)) \)

\( \text{handles} \) - handles to the lines plotted

Example

\[
\text{syms x y}
\text{f} = (x+y+1)*(x+y-1)*(y-x-1)*(y-x+1)+(x^2-1)^4+(y^2-1)^4-1;
\text{[p initialForm h] = cone(f, [-2 2 -2 2], 200);}
\]

\% returns a char array representing the output from maple of factoring the
\% first singular point’s initial form - note it’s over the rationals, not reals
\text{char(maple(’factors’, char(initialForm(1)))})

![Plot produced by cone](image-url)

Figure 2: Plot produced by cone
Algorithm

First finds all singular points of the function $f = 0$, by solving $f = \frac{df}{dx} = \frac{df}{dy} = 0$. At each singular point, the initial form is found by translating $f$ so that the singularity is at the origin, and using Maple's mtaylor function to extract the lowest total degree (in $x$ and $y$) monomials. Next, the initial form is factored over the rationals using Maple's factors function. Each of the linear factors are then plotted as tangent lines at the singularity. Any nonlinear factors are then factored again using Maple's factors function, this time over the reals. To find factors over the reals, however, Maple requires an input in only one variable. So the algorithm simply substitutes 1 for $y$, and performs the factorization. Since the expression was homogeneous, the algorithm easily recovers the $y$ dependence by substituting $\frac{x}{y}$ for $x$, and multiplying each linear factor by $y$. All resulting linear factors are plotted as tangent lines at the singularity. Thus, all real linear factors are found and plotted.

See Also - fCone, maple, mhelp, sym
0.3.3 envelope

Description - Returns the equation of the envelope of a family of two dimensional algebraic curves.

Syntax

envelopeFunc = envelope(f, var)

Parameters/Return Values

f - should be a function of x, y and some other variable, named in var
var - the name of the parameter in the family whose envelope is to be found
envelopeFunc - the equation of the envelope, as a matlab symbolic object (looks just like the character string of the equation)

Example

env = envelope(''(x-z)^2+(y-z^2)^2 - 4', 'z')
ezplot(env);

![Plot of the envelope by ezplot](image)

Figure 3: Plot of the envelope by ezplot

Algorithm

Uses the Maple function gbasis to find a grobner basis $G$ for the variety defined by $f = \frac{df}{dt} = 0$, where $t$ is the parameter in the family defined by $f$. Returns the GCD of the equations in $G$ which don’t contain the parameter $t$.

See Also - family, famEnv, ezplot, gbasis (Maple)
0.3.4  \texttt{fCone}

\textbf{Description} - Plots a two dimensional algebraic curve, along with the multiple tangent lines at its singularities.

\textbf{Syntax}

\[ \text{[singPts iForms fHan cHan C]} = \text{fCone}(f, [\text{xMin xMax yMin yMax}], \text{gridPoints}) \]

\textbf{Parameters/Return Values}

\(f\) - expression in \(x\) and \(y\), and the equation \(f = 0\) is plotted

\textit{gridPoints} - scalar or 1x2 vector specifying the number of grid coordinates in the \(x\) & \(y\) directions (higher for a more precise plot)

\textit{singPts} - singular points of the function, as returned by cone

\textit{iForms} - initial forms, at the singular points, as returned by cone

\(f\text{Han}, \text{cHan}\) - column vector of handles to the lines of the function, and a column vector of handles to the tangent lines plotted by cone

\(C\) - contour matrix (see help contour),

\textbf{Example}

\begin{verbatim}
syms x y
f = (x+y+1)*(x+y-1)*(y-x-1)*(y-x+1)+(x^2-1)^4+(y^2-1)^4-1;
[pts forms fh ch c] = fCone(f, [-2 2 -2 2], 200);
set(ch, 'linewidth', 3); \% thickens line width of tangent lines
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fCone_plot.png}
\caption{Plot produced by fCone}
\end{figure}

\textbf{See Also} - aPlot, cone
0.3.5 famEnv

Description - Plots a one-parameter family of curves, along with the envelope of the family, in two dimensions.

Syntax

(env hFam hEnv)=famEnv(f,[zMin, zMax], steps, [xMin, xMax, yMin, yMax], gridPoints)

Parameters/Return Values

f - always to be a string expression containing only variables x, y, & z

z - always to be the parameter that is variable in the family steps is a scalar specifying the number of frames in the movie

gridPoints - scalar or 1x2 vector specifying the number of grid coordinates in the x & y directions (higher for a more precise plot)

env - the equation of the envelope, as a matlab symbolic object (looks just like the character string of the equation)

hFam, hEnv - 2 column vectors of handles to lines in the plot, so the color of the lines can be changed

Example

[env hf he] = famEnv(’(x-z)^2+(y-z^2)^2 - 4’, [-4 4], 25, [-6 6 -2 6], 200)
set(hf, ’color’, ’r’) % changes color of family to red
set(he, ’linewidth’, 5) % changes width of envelope to 5

Figure 5: Plot produced by famEnv

See Also - family, envelope
0.3.6 family

Description - Plots a one-parameter family of algebraic curves in two dimensions.

Syntax

```
handles = family(f, [zMin, zMax], steps, [xMin, xMax, yMin, yMax], gridPoints)
```

Parameters/Return Values

- **f** - An expression in the variables $x$, $y$, and $z$, where $z$ is the parameter in the family. The equation $f = 0$ is plotted at the values of $z$ in the range.
- **steps** - is a scalar specifying the number of frames in the movie
- **gridPoints** - is a scalar or $1$x$2$ vector specifying the number of grid coordinates in the $x$ & $y$ directions (higher for a more precise plot)
- **handles** - column array of handles to lines in the plot, so the color of the lines can be changed.

Example

```
h = family(’(x-z)^2+(y-z^2)^2 - 4’, [-4 4], 25, [-6 6 -2 6], 100)
set(h, ’color’, ’r’) % changes color of lines to red
```

![Figure 6: Plot produced by family](attachment:family_plot.png)

See Also - famEnv, aPlot
0.3.7  makeMovie

Description - Creates an animation of a one-parameter family of algebraic curves in two dimensions.

Syntax

\[ M = \text{makeMovie}(f, [\text{Min}_z \text{Max}_z], \text{frames}, [\text{Min}_x \text{Max}_x \text{Min}_y \text{Max}_y], \text{gridPoints}) \]

Parameters/Return Values

\( f \) - An expression in the variables \( x, y, \) and \( z \), where \( z \) is the parameter in the family. The equation \( f = 0 \) is plotted at the values of \( z \) in the range.

\( \text{frames} \) - scalar specifying the number of frames in the movie (and the step size of \( z \))

\( \text{gridPoints} \) - scalar or 1x2 vector specifying the number of grid coordinates in the \( x \) & \( y \) directions (higher for a more precise plot)

Example

\[
\begin{align*}
s & \text{ys} \ x \ y \ z \\
f & = y^2 - x^3 + 3*x + z; \\
m & = \text{makeMovie}(f, [2 -2], 10, [-3 3 -3 3], 100); \\
\text{movie}(m); & \% \text{plays the movie at full speed}
\end{align*}
\]

![Figure 7: Last frame of movie](image)

See Also - writeMovie, readMovie, playMovie, family, movie
0.3.8 playMovie

**Description** - Plays the movie that was saved to disk using writeMovie, and also restores the title and axis ranges, which aren’t stored in a movie matrix.

**Syntax**

\[ M = \text{playMovie}(\text{name}) \]

**Parameters/Return Values**

*name* - character string of the name the movie was saved as

\[ M \] - the movie matrix that can be played using movie()

**Example**

```matlab
syms x y z
f = y^2 - x^3 + 3*x + z;
m = makeMovie(f, [2 -2], 10, [-3 3 -3 3], 100);
writeMovie(m, 'myMovie');
mov = playMovie('myMovie');
```

![Figure 8: Last frame of movie](image)

**See Also** - writeMovie, movie, makeMovie, readMovie
0.3.9  readMovie

Description - Reads a movie that was saved by writeMovie into a movie matrix. The axis and title information is read into a string that may be eval’d to restore those attributes.

Syntax

\[ M, str] = \text{readMovie}(\text{movieName}) \]

Parameters/Return Values

\( M \) - the movie matrix

\( str \) - character string that may be eval’d to restore title and axis information

Example

\[
\text{syms} \ x \ y \ z \\
f = y^2 - x^3 + 3*x + z;
\]
\[
m = \text{makeMovie}(f, [2 -2], 10, [-3 3 -3 3], 100);
\]
\[
\text{writeMovie}(m, 'myMovie');
\]
\[
[\text{mov} \ str] = \text{readMovie}('myMovie');
\]
\[
\text{eval}(str); \quad \% \text{sets the title and x and y range of the movie}
\]
\[
\text{movie(mov);} \quad \% \text{plays the movie that was read from files}
\]

![Graph](image.png)

Figure 9: Last frame of movie

See Also - playMovie, makeMovie, eval
0.3.10 replace

Description - Substitutes several variables at once. Syntax

\[ \text{func} = \text{replace}([\text{in1 in2 in3...}], [\text{out1 out2 out3...}]) \]

Parameters/Return Values - in’s and out’s must be symbolic objects (created with “syms”) - they’re the variables to be substituted. The output variable func is the resulting expression.

Semantics - The variable \text{out1} is substituted for \text{in1}, \text{out2} for \text{in2}...

Example

```plaintext
syms x y z
f = (x-z)^2+(y-z^2)^2 - 4
g = \text{replace}(f, [y z], [z y]); \% in g, y & z are swapped
```

Algorithm - First replaces all “in” variables \text{inx} with \text{inxqqTempqq}. Then replaces all temporary variables \text{inxqqTempqq} with \text{outx}.

See Also - syms
0.3.11 writeMovie

Description - Writes a series of .tiff files to a subdirectory “movies/name”, where name is an argument supplied. Also writes an additional file containing extra title and axis information.

Syntax

writeMovie(movieMatrix, name)

Parameters/Return Values

movieMatrix - a movie matrix, as created by makeMovie, or with getframe
name - character string designating the name the movie is saved to disk as (in a directory “movies/name”)

Example

syms x y z
f = y^2 - x^3 + 3*x + z;
m = makeMovie(f, [2 -2], 10, [-3 3 -3 3], 100); writeMovie(m, 'movieName');

Figure 10: Last frame of movie

See Also - readMovie, makeMovie, playMovie, getframe, movie
0.4 General Matlab Guide

0.4.1 Matrix and String Manipulation

Matrix Manipulation

Matrices are very important in Matlab, as Matlab’s name suggests, so it is very important to have a grasp of matrix manipulation in Matlab. In Matlab, one can select whole or partial rows and columns out of a matrix or even select an ordered set of arbitrary elements.

For a very good introduction to this, the reader is referred to a few specific pages in Matlab’s Html documentation. To find it, at the Matlab prompt, type “helpdesk”. This should invoke a Netscape window at the main (local) Matlab help page. On the left, under Matlab topics, click on “Getting Started”. In the left frame of the page that follows, clicking on “Matrices and Magic Squares” and “Working with Matrices” will provide an excellent introduction, including an explanation of the colon (:) operator.

One important matrix manipulation is concatenation of arrays, which also turns out to be useful in string manipulation in Matlab. Given row vectors $a$ and $b$, to concatenate $b$ onto the end of $a$, simply type:

$$\text{>> } a = [a \ b]$$

String Manipulation

Especially when dealing with symbolic expressions and the Maple kernel from Matlab, some knowledge of dealing with character strings in Matlab is helpful.

Essentially, a character string is just a row vector of characters, so one can build a string using the usual matrix concatenation techniques. Of course, it’s still displayed as a string, not as a row vector. To enter a string, just enclose a series of characters inside single quotes. If the string is actually to contain a single quote, the single quote should be typed twice; for example:

$$\text{>> } 'I’m a string!’$$

ans =

I’m a string!

$$\text{>> } \text{str} = [\text{ans } \ ‘ \text{Now I’m longer!’}]$$

str =

I’m a string! Now I’m longer!

One needs to be careful about data type conversion when forming a string with symbols that
don’t represent strings. If you don’t specify a conversion, char() is applied to other types. For example, ['Give me some ' 36] returns 'Give me some $', since char(36) returns '$'. Some useful conversions are num2str(), str2num(), sym() (converts to a symbolic object), and of course char(). So, if you really didn’t want '$', you should type ['Give me some ' num2str(36)].

(More Advanced) It is also possible to build an array of strings (actually a 2-d matrix of characters - it’s even possible to build such a matrix of arbitrary dimension) using strvcat or str2mat. (The difference between these is that strvcat ignores empty strings.) An unfortunate side effect of building such a matrix is that shorter strings in the array have spaces appended at the end to make them as long as the longest string in the array (so they fill out the matrix). This whitespace can be chopped by converting the string array to a cell array using cellstr(). Elements of the cell array may then be converted back into (chopped) character strings. For example:

>> array = [];
>> array = strvcat(array, 'I');
>> array = strvcat(array, 'am');
>> array = strvcat(array, 'an array of strings.')
returns: array = I
           am
           an array of strings
>> [array(1, : ) , ' am a string!']
returns ans = I am a string!
>> cellArray = cellstr(array)
>> [char(cellArray{1}) , ' am a string!']
returns ans = I am a string!

0.4.2 Maple and Symbolic Variables in Matlab

Matlab has essentially all the functionality of Maple, minus Maple’s displaying and plotting capabilities. Statements may be passed directly to the Maple kernel by use of the maple() function, though there are some Matlab functions (part of Matlab’s symbolic math toolbox) that process symbolic variables and call maple() for you, such as diff(), int(), expand(), and subs(). In order to use these functions you must first declare the symbolic variables to matlab, using the syms command:

>> syms x
   OR
>> syms x y m n variable1 var2

There are two basic syntaxes for the maple() function. The first is as follows:

Syntax

[result status] = maple(statement)

Parameters/Return Values
*statement* - character string that's passed to the maple kernel

*result* - either a character string or symbolic object (which can be converted to a string via `char(result)`)

*status* - integer error code (0 for no errors)

**Use**

Just pass statements to the Maple kernel as though you were using maple (except don’t include the usual ending semicolon). Definitions are remembered as long as Matlab is running.

**Examples**

```matlab
>> maple('a:=5')
>> charStr = maple('a')  % returns a character string containing the number 5
>> aNumber = str2num(charStr)  % returns the number 5

>> maple('int(x^2, x)')  % returns 1/3*x^3 as char array

>> syms x
>> int(x^2, x)  % returns 1/3*x^3 as sym object (matlab function that calls maple)
```

The second form of syntax for the `maple()` function is as follows:

**Syntax**

```
[result status] = maple('function', ARG1, ARG2, ...)
```

**Parameters/Return Values**

*function* - character string of the name of the Maple function; can also be of the form ’var:=function’

*ARG1* - can be symbolic objects, character strings, or numbers

**Use**

The Maple statement, “*function(ARG1, ARG2, ...)*” is executed by the Maple kernel. Arguments that are character strings are interpreted in Maple’s variable space. Always bear in mind that the result is returned as a character string or symbolic object, but not a number, so one should apply any necessary conversions.

**Examples** - “op” is a Maple function used to select elements from ordered lists and other structures.

```matlab
>> maple('a:=[11, 12, 13, 14, 15]')
>> Index = 1;
>> maple('Index := 5')  % Index has a different value in Maple’s space
```
>> str2num(maple('op', 'Index', 'a')) % returns 11 (used Matlab's Index)
>> str2num(maple('op', 'Index', 'a')) % returns 15

>> syms x y
>> f = x^2 - y^2 % f is automatically made into a sym object
>> result = maple('factors', f)
returns: result = [1, [[x-y, 1], [x+y, 1]]] as a sym object
>> fact = maple('op', '[2, 1, 1]', result)
returns: fact = x-y as a symbolic object

See Also - op (Maple function - use Matlab function mhelp to look up), syms, str2num, char, sym (converts char string to sym object, if it contains already declared sym variables)

0.4.3 Writing a Function (By Example)

function[out1, out2] = myFunc(x, y, z)
%[out1 out2] = myFunc(x, y, z)
%everything after a % is considered a comment
%when "help functionName" is typed, all comment lines
%immediately following the function declaration are displayed
%
%To work, this must be stored in a file called myFunc.m
%
%z - is a scalar, x and y are row vectors
%
%if z is greater than or equal to zero, out1 contains the dot
% product x*y, and out2 contains the difference y-x
%if z is less than zero, out1 contains the mean of the numbers
% in x, and out2 contains their total

if (z < 0)
    lengthOfX = length(x);
    out2 = 0;
    for i = 1 : lengthOfX
        out2 = out2 + x(i);
    end
    out1 = out2/lengthOfX
else
    out1 = x * y;
    out2 = y - x;
end
disp('Done');

% myFunc returns values curently stored in out1 & out2
% the function may be called like: "[a b] = myFunc(c, d, e)"
% however, it may also be called as "a = myFunc(c, d, e)"
% or simply "myFunc(c, d, e)" - the first output variable will
% be stored in the special Matlab variable "ans".

0.4.4 Helpful Matlab Functions

The intent of this section is to make the reader aware of the existence of useful Matlab functions. To fill in the details, just type "help functionName".

Most Helpful

```text
colon(:) - try 7:10 and 7:2:10 - a colon by itself as a matrix index selects all indices in that dimension - A(:,2) selects the entire 2nd column
diff - differentiation
eval, feval - evaluates a string as a series of matlab commands, feval can be used to call an arbitrary function
ezplot - can do 2-d implicit plots
factors (maple function) - factors a univariate polynomial over any field, or a multivariate poly over the rationals
for - a counting loop
help - help on any matlab function
helpdesk - invokes Netscape for the most comprehensive help and documentation
if/else - control statement (self explanatory)
int - integration
maple - access to the Maple kernel and pretty much any Maple function
mhelp - help on Maple commands
mtaylor (Maple function) - returns a multivariate taylor series (needs readlib(mtaylor))
np (Maple function) - selects items out of an ordered list, or other Maple structures
percent(%) - comment character - everything after % on a line is ignored
period(.) - when it precedes a matrix operator (such as * or ^), it indicates an element-wise operation (just like + always is)
semicolon(;) - when included at end of a statement, it suppresses output; also used to separate elements in column concatenation
subs - substitution for a symbolic variable in an expression
syms - used to declare symbolic variables, as syms x y z
while - loop control statement - loops while a condition is met
who/whos - gives information on what variables are declared, and specific info about each variable
```
Very Helpful

char - converts to a character string

clear - clears some or all variables

comma(,) - can be used to separate elements in row concatenation (a space may also be used); also separates function args and matrix indices

diary - logs matlab input/output to a text file

exclamation(!) - placed at the beginning of a line, it sends the contents of the line to the command shell; some important commands don’t need a !, such as ls, cd, cp, mv, etc.

find - very powerful - returns a set of indices whose elements in a matrix meet a condition

gbasis (Maple function) - finds the reduced grobner basis for an ordering - needs with(grobner)

getframe - used to populate a movie matrix, and play a movie

hold - controls whether to clear the current plot before performing another plot operation

isempty - determines whether an existing matrix = [] (has 0 elements)

length - returns the largest dimension of a matrix

num2str - changes a number type to character string type

plot - plots functions, taking vectors as input (not symbolic expressions)

set - used to change attributes such as color and linewidth in a plot

size - returns an n-dimensional array of lengths, given an n-dimensional matrix

str2num - converts a character string to a number type

symvar - finds what symbolic variables are present in a string

vectorize - given a string expression, it converts it to a strictly element-wise operation, by replacing all “*” with “.*”, etc., so that a calculation can be performed on many input combinations in one statement (using matrices); used by aPlot

vpa - (stands for variable precision arithmetic) - used to convert a symbolic expression containing square-roots of constants to a symbolic expression with numerical approximations of the roots

Helpful

cellstr - converts a character array to a cell array

contour - produces a contour plot (used by aPlot)

disp/display - displays text to the terminal

error - used to send an error (in functions)
getenv - returns the value of an environment variable

ishold - determines whether “hold” is on (see hold)

linspace - returns a vector of \( n \) linearly spaced points between scalars \( a \) and \( b \)

load - can load variables and what-not from a “.mat” file

meshgrid - (used by aPlot) - takes two row vectors representing the valid \( x \) and \( y \) coordinates, and returns two matrices representing the \( x \) and \( y \) values at various points on a grid (used with vectorize)

save - can save variables and what-not to a “.mat” file

strcmp - compares two strings

strrep - replaces occurrences of a substring \( s \) within a string \( str \) with another string \( t \)

strtok - used to extract substrings between delimiters such as spaces or double quotes

strvcat - (similar to str2mat) - forms an array of character strings (actually a matrix of characters)

sym - (different from sym) converts to a symbolic object

try/catch - used to detect errors and recover (usually for programs)

warning - can specify whether warnings are displayed; can also send a warning and specify what message is displayed

which - tells which file implements a function, so you can read the file and see exactly how it is implemented