

MATLAB for 2214 *
Fall 1999 – Spring 2000

Department of Mathematics
Virginia Tech University

1 August 1999

Contents

1	Introduction	2
1.1	Home Use – Downloading/Locating Files	2
1.2	Math Emporium Use	2
1.3	Infant MATLAB	3
2	First Order ODEs Symbolically	5
2.1	Required Files and Descriptions	5
2.2	Using <code>slopef.m</code>	6
2.2.1	Example	7
2.3	Using <code>sol1.m</code>	8
2.3.1	Examples	8
2.4	Using <code>hw3plot.m</code>	10
2.4.1	Example	11
3	First Order ODEs Numerically	14
3.1	Required Files and Descriptions	14
3.2	Using <code>eulplot.m</code>	15
3.2.1	Example	15
3.2.2	Higher order methods	16
3.3	Using <code>rk4plot.m</code>	16
3.3.1	Example	16
3.4	Using <code>num3plot.m</code>	16
3.4.1	Example	17
3.5	Using <code>nsys.m</code>	20
4	Systems of First Order ODEs	20
4.1	Required Files and Descriptions	20
4.2	Using <code>pplane2d.m</code>	23
4.2.1	Example	23

*This material is based in large part on earlier work by Dr. Martin Day, Virginia Tech. Thanks also to John McGraw and Dr. Michael Renardy, Virginia Tech, as well as Dr. John Polking, Rice University.

4.3	Using <code>sys3plot.m</code>	23
4.3.1	Examples	24
5	One Way to Plot Solutions	27
5.1	<code>plotsln.m</code> – General Instructions	27
5.2	Using with <code>subplot</code>	27
5.3	Several plots on the same axes	28

1 Introduction

The MATLAB portion of the course is designed so you can do the work at home or in the Math Emporium, whichever you prefer. To use your home computer, you will need to have MATLAB 5.0 or higher (Student or Professional) and printer access. For Emporium use, you will need monetary credit on your Hokie Passport to print results.

1.1 Home Use – Downloading/Locating Files

There are up to 23 M-files you will need. You can download the files via the department web page directory

<http://www.math.vt.edu/cgi-bin/toolkit/fileseek.cgi?dir=/matlab/math2214>

Eventually we will use all those files in the directory with the `.m` extension. Also some files call others in the list, so you may as well get them now. To reduce confusion during this course, it is recommended that you create a directory `M2214` and subdirectory `Matlab`. Then store all MATLAB files for this course in that subdirectory, e.g., `c:\M2214\Matlab`. IMPORTANT: Each time you use MATLAB for 2214, you must be in this directory. A listing of the files organized by usage is:

General 1st Order	Numerical 1st Order	Systems	Miscellaneous
<code>bdode45.m</code>	<code>eulplot.m</code>	<code>cclin.m</code>	<code>plotsln.m</code>
<code>eqn1d.m</code>	<code>eulstdnt.m</code>	<code>ccquad.m</code>	
<code>hw3plot.m</code>	<code>initn.m</code>	<code>initsys.m</code>	
<code>hwsoll.m</code>	<code>num3plot.m</code>	<code>pend.m</code>	
<code>init.m</code>	<code>rk4.m</code>	<code>plotslnf.m</code>	
<code>slopef.m</code>	<code>rk4plot.m</code>	<code>plotxytf.m</code>	
<code>soll.m</code>	<code>initsys.m</code>	<code>pplane2d.m</code>	
	<code>nsys.m</code>	<code>pplane2f.m</code>	
		<code>sys3plot.m</code>	

1.2 Math Emporium Use

Currently the M-files necessary for MATH 2214 are only available on the MACINTOSH side of the Emporium computers. However this should pose only a minor inconvenience for PC users. There will rarely be anything that you will need to save and take with you.

1. Open MATLAB by clicking on the icon located in the **Launcher** under **Applications**.
2. When the MATLAB command window appears, use the **Open Files** icon to change to the **2214** folder, then open the file `init.m`. Do not be concerned about the warning dialog boxes. Just choose the options that will open the file.
3. Save `init.m` to the **Student's Folder** on the **Desktop** by choosing **File** from the toolbar at the top-left of the monitor screen, choosing **Save As . . .**, changing to the **Student's Folder** under **Desktop**, and choosing **Save**. This copy will become your working `init.m` file. (This step is a safety feature.

You cannot write to any files unless they are in the **Student's Folder**. Regularly saving your working file is strongly recommended.)

4. In the MATLAB command window toolbar, use the **Open Files** icon to change back to the working directory, **MATLAB 5/2214**, then choose **Cancel**. MATLAB **must** be in this directory to access the necessary files. The full path to this directory is

Desktop/Macintosh HD/Applications/MATLAB 5/2214

There is only one other difference between Emporium and home use. Because `init.m` is in a different folder than the other M-files (the **Student's Folder**), Emporium users cannot just type `init` at the command prompt (as home users can). You must copy-and-paste the full file contents into the MATLAB command window each time a change is made.

1.3 Infant MATLAB

Generally MATLAB syntax is pretty conventional (i.e., similar to current programming languages). If you are used to Mathematica, one difference that you will notice is that the arguments of functions are enclosed in parentheses (e.g., $\sin(x)$) rather than with square brackets. In MATLAB, square brackets are used to specify vectors or matrices. For instance `[1,0,0;0,1,0;0,0,1]` would specify the usual 3x3 identity matrix. Either commas or spaces can be used to separate the elements in a row; semicolons separate the rows themselves. Variables always have values: numerical (integer, real or complex), matrix, or string. String values are indicated by enclosing them in single quotes, as you will see below.

Our primary use of string variables will be to specify the right hand side (RHS) of a differential equation of the form $x' = f(t, x)$. For example if we are going to work with the differential equation $x' = 2tx$ then we would make the following assignment to a string variable named `ftx`:

```
ftx='2*t*x';
```

Note that `*` is used to indicate multiplication. Any valid MATLAB expression can make up the string, as long as the only two variables used are `t` and `x` (remember we are entering an equation of the form $x' = f(t, x)$). For instance `'exp(t^3 - 1)/sqrt(log(x^2+1))'` is also an acceptable string variable.

MATLAB commands are entered at the prompt `>>`. Virtually all commands have an associated M-file given by `command_name.m`. For example, `quit` is the command to end a MATLAB session and close the software. Typing `quit` at the prompt `>>` invokes the M-file `quit.m`. Similarly, creating a M-file with MATLAB commands and/or variable assignments also allows you to call the M-file using its name, minus the `.m` extension. The file `init.m` is such a file as is `sol1.m`. If you are in the directory/folder containing `init.m`, typing `init` will invoke the file and those commands/assignments it contains. The same holds for `sol1.m` and `sol1`. All of these files are simple ASCII text files. (Remember though that you must copy-and-paste `init.m` at the Emporium.)

Information on almost any command/file can be obtained by typing `help command` at the prompt. For example, `help quit` brings up information on the command `quit`, which is also the information on the file `quit.m`.

For more about general MATLAB use there are several books available. One of these is *Engineer's Toolkit*, required for most VT engineering majors. Another is the new tutorial compiled by the Mathematics department. MATLAB itself has a built-in tutorial (type the command `intro`).

For the beginning of this class, you will basically only need to know these simple facts. However some other information may make you more comfortable.

Some MATLAB commands

`pwd` – This stands for “print working directory”. You can use this to check that you are in `c:\M2214\Matlab\`.

`axis([a,b,c,d])` – This specifies the portion of the `tx`-plane that will be visible in the graphics window: `a<t<b`, `c<x<d`. If no graphics window is already displayed, it will create one. The `t`-axis is horizontal, the

x-axis is vertical. Generally you will not work directly with the `axis` command. Instead you will enter the desired values as the vector-valued variable `initax` in `init.m`. For example, `initax=[-2,2,-4,4]`; and `axis(initax)` sets the graphics window to $-2 < t < 2$, $-4 < x < 4$.

`plot(t,x)` – If the vectors `t` and `x` have been created and have the same size, `plot(t,x)` will plot them in 2D.

```
Example    t=0:0.1:5;
           x=t.^2;
           plot(t,x)
```

This plots the function $x = t^2$ for $0 \leq t \leq 5$.

`print` – You will need to print copies of various graphics displays. Methods may differ slightly depending on the machine/system you are using. First click on the desired graphics display window to be sure it (not the command window) is the active window. (If there is more than one graphics display window, be sure to choose the correct one.) Generally MATLAB is installed so that you only need to enter the command `print`, and your plot will be printed to your default printer. You can also “print” your graphics to a PostScript file for later printing. Use the command `print <filename.ps>` where “filename” is one of your choice.

`clear` – This clears all variables and assignments for the current session of MATLAB. Sometimes this is necessary to just start things over. Individual variables can also be cleared. Type `help clear` at the command prompt for more information.

init.m Line-by-Line `init.m` contains variable assignments and one actual command. Each of these lines could also be individually entered by hand at the command prompt. (The symbol `%` tells MATLAB to ignore everything else that follows on that particular line. It usually indicates a comment/description inserted by the author of the M-file.)

`N=20;` Creates the variable `N` and assigns it the number value 20. (The semicolon (`;`) at the end stops MATLAB from writing this information to the screen. Try entering just `N=20` to see the difference.)

`ftx='2*t*x';` Creates the variable `ftx` and assigns it the string value `2*t*x`. This is one way to create a mathematical expression for later use and evaluation. It may not be a method familiar to many MATLAB users, but has several advantages. The symbol “`*`” refers to multiplication.

`initax=[-2,2,-2,2];` Creates the variable `initax` and assigns it the row-vector value `[-2,2,-2,2]`. Row elements can be separated by either a comma or a space.

`ax=initax;` Creates the variable `ax` and assigns it whatever value the variable `initax` has.

`axis(ax)` `axis` is a standard MATLAB command that sets the domain and range of a plot. `axis(ax)` sets these values to those in the vector variable `ax` (hence `initax`), which is `[-2,2,-2,2]`. This command will also open a figure window if none is active.

`t=ax(1):(ax(2)-ax(1))/(N-1):ax(2);`

Creates the variable `t` and assigns it the row-vector value

$$\underbrace{[-2.0000, -1.7895, -1.5789, \dots, 1.5789, 1.7895, 2.0000]}_{20 \text{ elements}}$$

where each element is $4/19$ apart from its neighbor. The structure for this assignment is:

$$\begin{aligned} \text{initial } t\text{-value} &= \mathbf{ax}(1) &= -2 \\ \text{step size} &= \frac{\mathbf{ax}(2) - \mathbf{ax}(1)}{N-1} &= \frac{2 - (-2)}{20 - 1} = \frac{4}{19} \\ \text{final } t\text{-value} &= \mathbf{ax}(2) &= 2 \end{aligned}$$

These are the t -values MATLAB will use to plot any function using t as the independent variable. (This is one way in which MATLAB differs significantly from MATHEMATICA.) `ax(1)` and `ax(2)` refer to the first and second elements of the row-vector `ax`, respectively.

`t0=1.0932; x0=0.95208;` Creates the variables `t0`, `x0` and assigns them the number values 1.0932 and 0.95208.

`C=0.28817;` Creates the variable `C` and assigns it the number value 0.28817.

`ftxsln='C*exp(t^2)';` Creates the variable `ftxsln` and assigns it the string value `C*exp(t^2)`. `exp(t^2) = et2`. Note that the exponential function `exp` uses parentheses in its call not square brackets. This is true for all MATLAB functions.

2 First Order ODEs Symbolically

The types of ODEs that have symbolic solutions are actually in the minority. However they are used in many applications, often because we *can* solve them. Unfortunately these solutions do not always provide useful insights in their symbolic (i.e, equation) form. We can use MATLAB to plot graphical representations of first order ODE solutions. We will concentrate on plotting and comparing slopefields, symbolic solutions, and numerical solutions (via MATLAB's ODE45 solver).

2.1 Required Files and Descriptions

You will need seven of the non-standard M-files from the “M2214 Matlab Files” directory (see page 2): `init.m`, `slofef.m`, `sol1.m`, `hw3plot.m`, `bdode45.m`, `eqnid.m`, `hwsol1.m`.

init.m This is the file you will use/modify for submission of each first order ODE/IVP¹ to MATLAB. Its corresponding MATLAB command is `init` (the filename minus the `.m` extension). Each time you change the ODE `ftx`, the symbolic solution `ftxsln`, or any of the associated parameters/conditions, this file must be edited, saved, and `init` must be typed in the command window (or copy-and-paste at the Emporium). For a complete description of `init.m`, see “`init.m Line-by-Line`”, §1.3, page 4. The lines you will be most concerned with are

$$\begin{aligned} \text{ftx} &= \dots; \% \text{ Enter } \dots & (1) \\ \text{initax} &= [\dots]; \% \text{ Enter } \dots & (2) \\ \text{t0} &= \dots; \text{x0} = \dots; \% \text{ Intital conditions } \dots & (3) \\ \text{C} &= \dots; \% \text{ Initialize } \dots & (4) \\ \text{ftxsln} &= \dots; \% \text{ Enter } \dots & (5) \end{aligned}$$

(1) `ftx` is the RHS of the ODE $x' = f(t, x)$.

(2) contains the axis ranges for the graphics display. You will adjust these to find the best ranges to view

¹IVP stands for Initial Value Problem.

your graphics.

(3) are initial conditions you will enter *after* obtaining them via `slopef` and `sol1`. Hence whatever values are there at the beginning of a problem are generally immaterial. However you must have specific numerical values entered there.

(4) is obtained by hand from substituting `t0`, `x0` into your symbolic solution `ftxsln`.

(5) is the symbolic solution $x = \dots$ to the ODE $x' = f(t, x)$. You will obtain this by hand via the various techniques we will study.

Two things are important to repeat: DO NOT remove any lines from `init.m`, and be sure to use single quotes with the RHS expressions for `ftx` and `ftxsln`.

slopef.m This is a function file that plots the slopefield for the ODE entered as `ftx` in `init.m`. Its MATLAB command is `slopef(ftx,N)`. It uses a $N \times N$ grid of small slope lines to outline solutions to the ODE $x' = f(t, x)$ from data in `init.m`. The slopefield is plotted within the tx -rectangle specified by `initax`. `slopef(ftx,N)` is also automatically called by `sol1.m` and `hw3plot.m`. Note that `slopef.m` automatically erases anything currently displayed in the active graphics window.

sol1.m This file is used to generate a numerical solution to an IVP using `bdode45.m`. Its MATLAB command is `sol1`. It uses data from `init.m` and also requires direct input from the user. It first asks if a slopefield is also desired. Then next asks for a necessary initial condition (t_0, x_0) . The initial condition can be chosen by clicking directly on the graphics window with the provided cross-hairs, or by clicking outside the graphics area then entering the desired values directly in the command window. `sol1.m` works reasonably well, however in some cases the numerical routine can bog down if there are troublesome regions for the ODE. If this takes too long for your tastes, CTRL-C will make MATLAB give up its current calculation and return to the command prompt.

hw3plot.m This file creates a 3-plot page containing a slopefield, a graph of one symbolic solution, and a graph of the respective numerical solution to a first order ODE. The file also allows room for handwritten calculations on the resulting printout. Its MATLAB command is `hw3plot`. The file uses data from `init.m`, calls `slopef(ftx,N)` and `hwsol1.m`, then plots the symbolic solution given as `ftxsln`. If the user has solved the ODE correctly and entered all data into `init.m` properly, then the three plots should agree in their representations.

bdode.m, eqn1d.m, hwsol1.m These are all files called by those above and not by the user.

2.2 Using `slopef.m`

We want to plot the slopefield of the first order ODE $x' = f(t, x)$ over a user-designated window $a \leq t \leq b$, $c \leq x \leq d$.

1. Make the appropriate changes in `init.m`.

```
ftx = 'f(t,x)'; (RHS of  $x' = f(t, x)$ )
initax = [a,b,c,d]; (desired window size)
```

Save then type `init` in the command window to initialize the variables (or copy-and-paste in the Emporium).

(Unless otherwise specified, we will use $N = 20$; as originally written in `init.m`.)

2. Type `slopef(ftx,N)`
3. If the resulting plot does not show all of the general behavior for the ODE solution, you may have to choose new t and/or x -values for `initax`, then do Items 1 and 2 again.

2.2.1 Example

When you first copy `init.m`, it should be set to run this example.

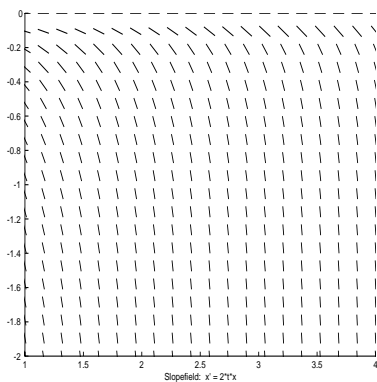
Plot the slopefield for $x' = 2tx$ over the rectangle $1 \leq t \leq 4$, $-2 \leq x \leq 0$. Investigate whether this rectangle gives a good overall picture of the ODE behavior. If not, plot over a better rectangle.

1. Change only the following in `init.m`

```
ftx = '2*t*x'; (*" indicates multiplication)
initax = [1,4,-2,0];
```

Save then type `init` in the command window to initialize the variables (or copy-and-paste in the Emporium).

2. Type `slopef(ftx,N)` to obtain



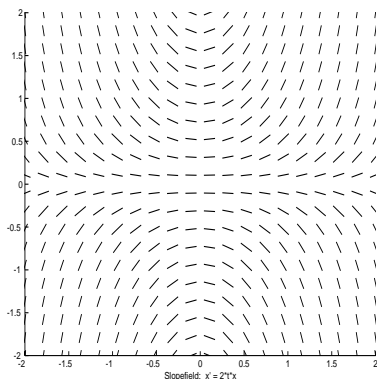
Since $x' = 2tx$ represents the slope, we should also consider when $t < 0$ and $x > 0$. Suppose we try the symmetric rectangle $-2 \leq t \leq 2$, $-2 \leq x \leq 2$.

1. This time change only

```
initax = [-2,2,-2,2];
```

Save then type `init` in the command window (or copy-and-paste in the Emporium).

2. Type `slopef(ftx,N)` to obtain



Build your intuition: Does this picture match the general slope behavior indicated by $x' = 2tx$? Look at the following cases

- (a) $t < 0$, $x > 0$, (b) $t < 0$, $x < 0$, (c) $t > 0$, $x < 0$, (d) $t > 0$, $x > 0$

2.3 Using sol1.m

We want to plot a specific numerical solution for the first order ODE $x' = f(t, x)$ over a user-designated window $a \leq t \leq b, c \leq x \leq d$. However we want freedom to enter the initial condition (t_0, x_0) either via the graphics window or command window.

1. Make the appropriate changes in `init.m`.

```
ftx = 'f(t,x)'; (RHS of  $x' = f(t, x)$ )
initax = [a,b,c,d]; (desired window size)
```

Save then type `init` in the command window to initialize the variables (or copy-and-paste in the Emporium).

2. If necessary, use `slopef.m` to determine suitable entries for `initax` (i.e., the plot rectangle).
3. Type `sol1`. You will be asked if you want a new slopefield. Answering "y" for "yes" will erase any existing slopefield and plot a new one.
4. At the command prompt you will be instructed to

`Click on initial point; outside the graphics frame for manual entry`

Use the crosshairs to click on the desired initial condition (t_0, x_0) or click outside the plot, but inside the graphics window, for manual entry of the t_0, x_0 -values in the command window.

2.3.1 Examples

Example 1 When you first copy `init.m`, it should be set to run this example.

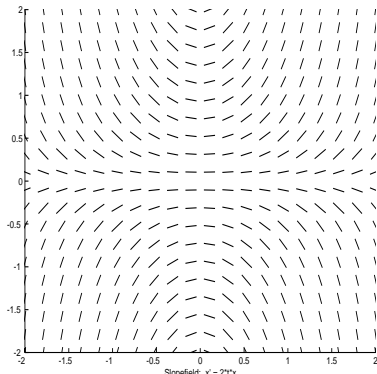
Plot a slopefield for $x' = 2tx$ over a rectangle illustrating the general characteristics of the ODE. Then plot one solution to the ODE within this rectangle.

1. Change only the following in `init.m`

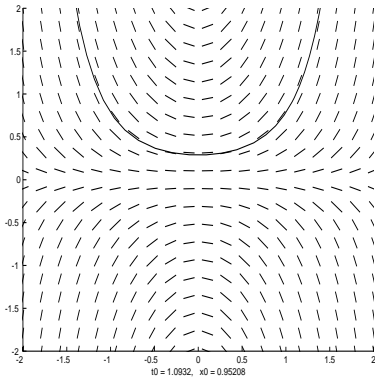
```
ftx = '2*t*x'; ("*" indicates multiplication)
initax = [1,4,-2,0];
```

Save then type `init` in the command window to initialize the variables (or copy-and-paste in the Emporium).

2. See the example for `slopef`, page 8, to investigate the window size `initax = [1,4,-2,0]`; then make the change to `initax = [-2,2,-2,2]`; in `init.m`. Save then type `init` in the command window to initialize the variables (or copy-and-paste in the Emporium).
3. Type `sol1`. Answer "y" to draw new slopefield



- Have MATLAB draw a solution close to $(t_0, x_0) = (1, 1)$ by clicking near that initial condition on the slopefield.



From the graphics display we observe the actual (t_0, x_0) -value. (In this case $(1.0932, 0.95208)$.)

Example 2

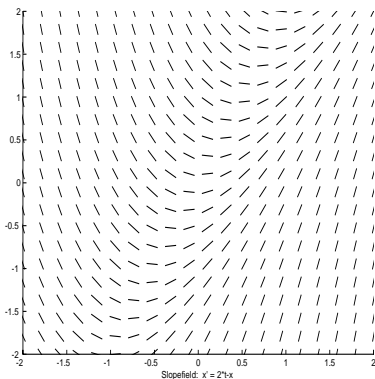
Investigate the behavior of the ODE $x' = 2t - x$. Plot one solution to the ODE in an appropriate rectangle that illustrates the general behavior of the ODE.

- Change only the following in `init.m`

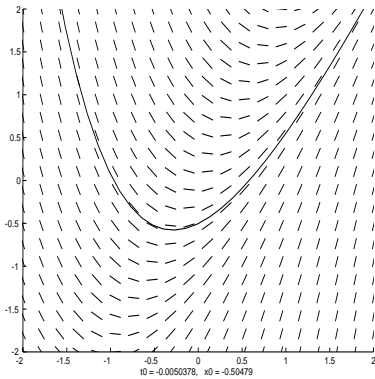
```
ftx = '2*t-x';
initax = [-2,2,-2,2];
```

Save then type `init` in the command window to initialize the variables (or copy-and-paste in the Emporium).

- Use `slopef.m` to determine if the entries for `initax` are suitable.



- Type `sol1`. Answer “n” to retain the current slopefield.
- Choose manual entry by clicking with the crosshairs inside the graphics window but outside of the actual plot. At each respective prompt enter $t_0 = -0.00504$ and $x_0 = -0.50479$.



2.4 Using `hw3plot.m`

Tying graphical, symbolic, and numerical representations together is an important goal of this course. `hw3plot.m` is designed to bring these together on one page.

1. Solve the indicated ODE by hand using whichever method is appropriate to the textbook section containing the problem. This is to be done on scratch paper.
2. Make the appropriate entries in `init.m` for `ftx` and `ftxsln`. Make an educated guess for `initax`-values. Be sure some value is entered for each of `t0`, `x0`, `C`. Save then type `init` in the command window to initialize the variables (or copy-and-paste in the Emporium).
3. Type `slopef(ftx,N)` to check that the plot window size is appropriate. Adjust `initax` in `init.m` if necessary. Save then type `init` in the command window (or copy-and-paste in the Emporium).
4. Type `sol1` and choose initial conditions (i.e., (t_0, x_0)) that produce a solution illustrating the “interesting” behaviour of the ODE. You may need to experiment here.
5. Note the `t0` and `x0`-values returned by `sol1` at the bottom of the plot and enter these values in `init.m`.
6. Solve for `C` by substituting `t0` and `x0` into the general symbolic ODE solution you obtained by hand, then enter the result into `init.m`.
7. Save `init.m` then type `init` in the command window to initialize the new variables (or copy-and-paste in the Emporium).
8. Type `hw3plot`. A “y” answer to “Do you already have a separate figure window for `hw3plot`? (y/n)” creates a new MATLAB Figure window specifically for `hw3plot`. If you have already used `hw3plot` in this session, an appropriate window should already exist, and you can answer “n “. In your Figure window, you should have three plots.
 - (a) Upper right contains Figure 1, the slopefield for `ftx`.
 - (b) Lower left contains Figure 2, your symbolic solution for the `C` obtained with the initial conditions (t_0, x_0) . This is plotted over the slope field.
 - (c) Lower right contains Figure 3, the numerical solution of $x' = f(t, x)$ determined via `sol1` plotted over the slope field.
9. If the solutions to Figures 2 and 3 do not match, there are generally two possibilities: (a) the existence and uniqueness theorems governing ODEs have been violated; or (b) you did not solve the ODE properly or made an error solving for `C`. In both cases, the numerical solution from MATLAB is usually correct. In case (a), your symbolic plot should look like the numerical plot except that yours will contain more of the graph. Your graph has “jumped” over a discontinuity. There is nothing to change at this time, but you should include an explanation to this effect. If your graph looks different in other ways, you are probably in case (b). Check your work and any of Items 1-8 that are appropriate for errors. If the solutions to Figures 2 and 3 are the same, print a copy of the display.

10. In the upper left of your printout, neatly and orderly rewrite the work you did to solve the ODE in Item 1 and to find C .

2.4.1 Example

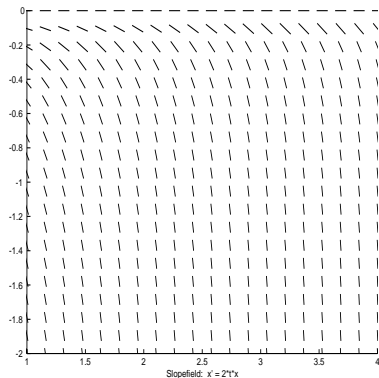
Use `hw3plot.m` to create a slopefield, numerical plot, and plot of the symbolic solution to the ODE $x' = 2tx$.

1. Separable equation

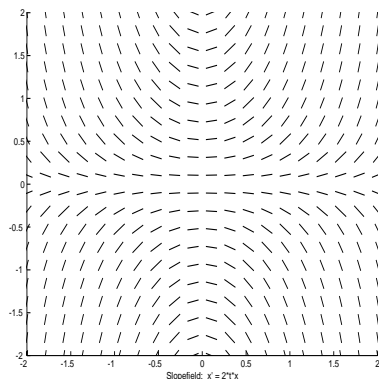
$$\begin{aligned} \frac{1}{x} dx &= 2t dx \implies \int \frac{1}{x} dx = \int 2t dt \\ \implies \ln|x| &= t^2 + C_1 \implies e^{\ln|x|} = |x| = e^{t^2+C_1} = C_2 e^{t^2} \quad (\text{for } C_2 = e^{C_1}) \\ \implies x &= C e^{t^2} \quad (\text{for } C = \pm C_2 = \pm e^{C_1}) \end{aligned}$$

2. `ftx = '2*t*x';`
`initax = [1,4,-2,0];`
`t0 = 0; x0 = 0; C = 1;`
`ftxsln = 'C*exp(t^2)';`

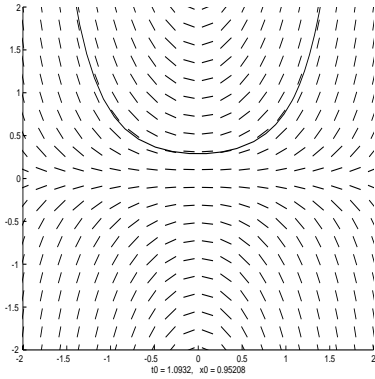
3. `slopef(ftx,N)` results in



Change `initax = [-2,2,-2,2]`; then `slopef(ftx,N)` results in



4. Running `sol1` and placing pointer at about $(1,1)$ results in



5. From the display we have $t_0 = 1.0932$ and $x_0 = 0.95208$ which we enter into `init.m`.

6. Using $x = Ce^{t^2}$, we solve $x_0 = Ce^{t_0^2}$; i.e.,

$$t_0 = 1.0932, x_0 = 0.95208 \implies 0.95208 = Ce^{1.0932^2} \implies$$

$$C = 0.95208 e^{-1.0932^2} = 0.28817$$

which we enter into `init.m`.

Completing Items 7-10, we obtain the finished product illustrated on the next page. (Note: your work in the upper left will be neatly hand written, not typed.)

95, §2.3, pp 38

$$x' = 2tx$$

Separable equation

$$\frac{dx}{dt} = 2tx \implies \frac{1}{x} dx = 2t dt \implies$$

$$\int \frac{1}{x} dx = \int 2t dt \ln|x| = t^2 + C_1$$

$$e^{\ln|x|} = |x| = e^{t^2+C_1} = e^{t^2} e^{C_1}$$

$$\implies |x| = C_2 e^{t^2} \quad (\text{for } C_2 = e^{C_1})$$

$$\implies x = C e^{t^2} \quad (\text{for } C = \pm C_2 = \pm e^{C_1})$$

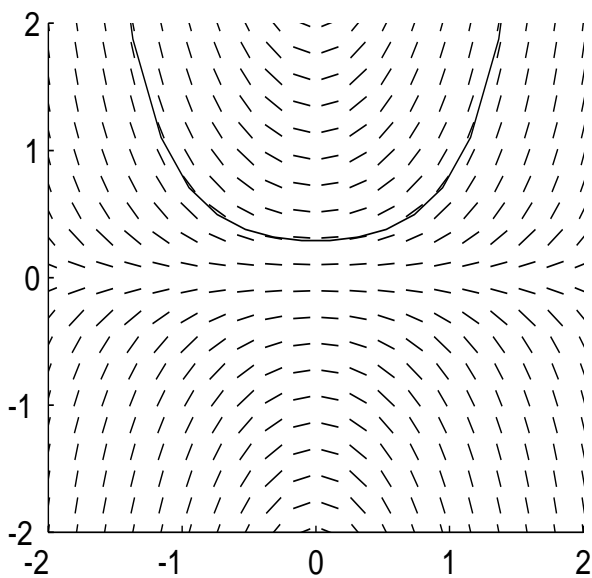
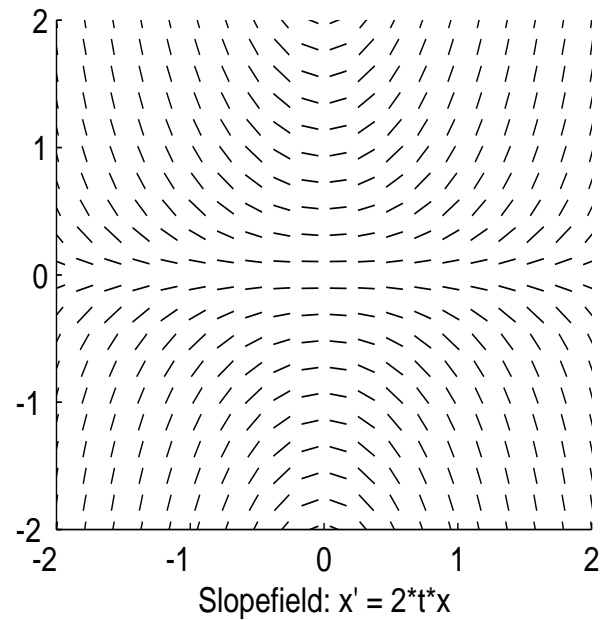
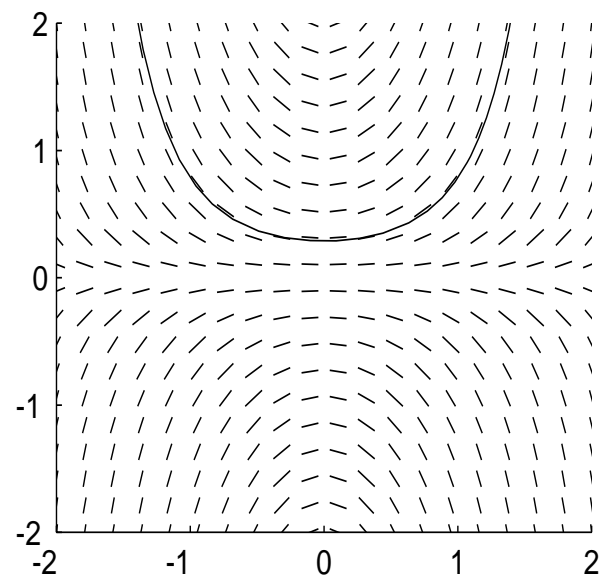
Choosing initial conditions

$$t_0 = 1.0932, x_0 = 0.95208 \implies$$

$$0.95208 = C e^{1.0932^2} \implies$$

$$C = 0.95208 e^{-1.0932^2} = 0.28817$$

$$\implies x \approx 0.28817 e^{t^2}$$

Symbolic soln: $x = C \cdot \exp(t.^2)$, $C = 0.28817$ Numerical soln: $t_0 = 1.0932$, $x_0 = 0.95208$

3 First Order ODEs Numerically

Numerical methods are arguably the most important methods for dealing with differential equations today. Even equations that seem simple at first may not have symbolic solutions. We will look at two numerical methods: Euler's method and the 4th order Runge-Kutta method.

3.1 Required Files and Descriptions

Eight non-standard M-files will be needed: `initn.m`, `eulplot.m`, `rk4plot.m`, `num3plot.m`, `eulstdnt.m`, `initnsys.m`, `nsys.m`, and `rk4.m`. You will need to download them.

initn.m This is similar to `init.m`. Each time you change the ODE `ftx` or one of the associated parameters/conditions, this file must be edited, saved, and `initn` must be typed in the command window (or copy-and-paste at the Emporium).

`ftx` is the RHS of the ODE $x' = f(t, x)$.

`eulerfcn` is the RHS of the iterating function for Euler's method.

`hvec` is a vector of step sizes that Euler's method will use. If only one step size is needed, a number may be used instead of a vector.

`t0`, `tf`, `x0` are the starting time, ending time, and starting place, respectively. They will generally be given with the problem.

Observe that the axis for the Figure window (`initax`) is now set from the values `t0`, `tf` for `t` as well as `x0` and an estimate for `x(tf)`.

eulplot.m Once the data from `initn.m` has been entered and initialized, typing `eulplot` will create a graph of each Euler approximation associated with a step size from `hvec`. All are plotted on the same axis on top of the slope field. The Matlab command window will show the computed numbers.

rk4plot.m Once the data from `initn.m` has been entered and initialized, typing `rk4plot` will create a graph of each 4th order Runge-Kutta approximation associated with a step size from `hvec`. All are plotted on the same axis on top of the slope field. (Note: We will not derive the iteration scheme for Runge-Kutta methods, so there is no analog to `eulerfcn` in `initn.m`. However you must have some string entered for `eulerfcn` so the variable is defined.) As for `eulplot.m`, the command window shows the computed numbers.

num3plot.m Typing `num3plot` will create a three-plot window like that for `hw3plot` – one using `eulplot.m`, one using `rk4plot.m`, and one comparing the “error” between the two.

eulstdnt.m, rk4.m These are numerical routines called by `eulplot`, `num3plot`, and `rk4plot` to create numerical solutions to the ODE. `eulstdnt.m` uses Euler's method. `rk4.m` uses a 4th order Runge-Kutta method. `eulstdnt.m` requires the variable `eulerfcn`, which you will enter into `initn.m`.

initnsys.m This is the analog of `initn.m` for a 2 by 2 system. You enter two functions `eulerfcn1` and `eulerfcn2` for the right hand sides of the Euler iteration for your two equations. `h` is the step size, `t0`, `tf`, `x0`, and `y0` are the starting time, final time and initial conditions.

nsys.m This is the numerical routine used to solve a 2 by 2 system.

3.2 Using `eulplot.m`

3.2.1 Example

When you first copy the above files, they should be set to run the following example.

Use Euler's method to approximate a solution to the ODE $x' = 2tx$. By hand, set up the iterating equation and solve for three iterations using the initial condition $x(-2) = 1$ and step size $h = 0.1$. Next investigate graphically using `eulplot` with the four step sizes $h = 0.5, 0.25, 0.1, , 0.01$. Identify each plot with its respective step size. Explain the rather odd behavior for $t > 0$ with $h = 0.5, 0.25$ when all plots are compared. You may need to calculate a few Euler iterations by hand for these step sizes. (Does this suggest more than one way that step size can affect results?)

1. Iterating equation $x_{n+1} = x_n + h(2t_n x_n)$ with $h = 0.1, t_0 = -2, x_0 = 1 \implies$

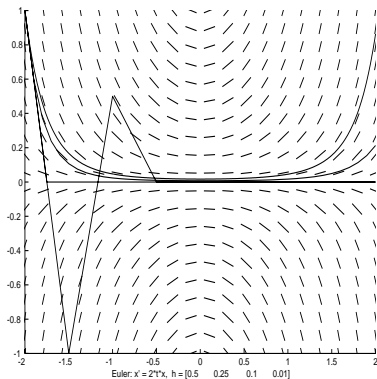
$$\begin{aligned} x_1 &= 1 + 0.1(2(-2)1) = 0.6 \\ x_2 &= 0.6 + 0.1(2(-1.9)0.6) = 0.372 \\ x_3 &= 0.372 + 0.1(2(-1.8)0.372) = 0.23808 \end{aligned}$$

2. Change only the following in `initn.m`

```
ftx='2*t*x';
eulerfcn='x+h*(2*t*x)';
hvec=[.5,.25,.1,.01];
t0=-2; tf=2; x0=1;
```

Save, then type `initn` to initialize the variables (or copy-and-paste at the Emporium).

3. Type `eulplot` to get



4. For $h = 0.5$, we have

$$\begin{aligned} x_1 &= 1 + 0.5(2(-2)1) = -1 \\ x_2 &= -1 + 0.5(2(-1.5)(-1)) = 0.5 \\ x_3 &= 0.5 + 0.5(2(-1)0.5) = 0 \\ x_4 &= 0 + 0.5(2(-0.5)0) = 0 \end{aligned}$$

Each subsequent iteration will also be zero.

For $h = 0.25$, we have

$$\begin{aligned} x_1 &= 1 + 0.25(2(-2)1) = 0 \\ x_2 &= 0 + 0.25(2(-1.75)0) = 0 \end{aligned}$$

Each subsequent iteration will also be zero.

3.2.2 Higher order methods

You can use `eulplot.m` for methods other than the Euler method if you change `eulerfcn` accordingly. For instance, if you use the improved Euler method for the equation $x' = t^2 + x^2$, then the iteration equation is

$$x_{n+1} = x_n + \frac{h}{2}(t_n^2 + x_n^2) + \frac{h}{2}((t_n + h)^2 + (x_n + h(t_n^2 + x_n^2))^2).$$

Consequently, you want to have the following line in `initn.m`:

```
eulerfcn='x+h/2*(t^2+x^2)+h/2*((t+h)^2+(x+h*(t^2+x^2))^2);'
```

3.3 Using `rk4plot.m`

3.3.1 Example

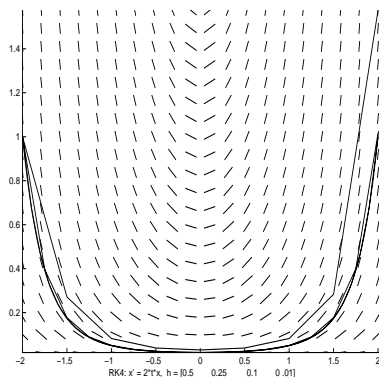
Use `rk4plot` to graphically investigate numerical solutions to the ODE $x' = 2tx$. Use the four step sizes $h = 0.5, 0.25, 0.1, 0.01$. This M-file uses a 4th order Runge-Kutta scheme. Identify each plot with its respective step size.

1. Change only the following in `initn.m`

```
ftx='2*t*x';
hvec=[.5,.25,.1,.01];
t0=-2; tf=2; x0=1;
```

Save, then type `initn` to initialize the variables (or copy-and-paste at the Emporium).

2. Type `rk4plot`



(How does the plot for `eulplot` compare with that for `rk4plot` when $h = 0.1$? Which is more accurate for the same step size?)

3.4 Using `num3plot.m`

1. Set up the appropriate iterating equation for Euler's method that pertains to the ODE being considered. If instructed, solve a few iterations of the method by hand.
2. Make the necessary entries in `initn.m`.

```

ftx = 'f(t,x)'; (RHS of  $x' = f(t,x)$ )
eulerfcn = 'x_n + h f(t,x)'; (Iterating equation for Euler's method)
hvec = [h_1, h_2, ..., h_n]; (Desired step sizes)
t0 = t_0; tf = t_f; x0 = x_0; (initial t-value, final t-value, initial x-value, respectively)

```

Save and type `initn` (or copy-and-paste at the Emporium).

3. Type `num3plot`. This may take up to a few minutes depending on your machine, the choice of step size(s), and/or the difficulty of the ODE. In your Figure window, you should have three plots.
 - (a) Upper right contains Figure 1, the Euler approximations for each of the step sizes in `hvec`, plotted on top of the slope field.
 - (b) Lower right contains Figure 3, the RK4 approximations for each of the step sizes in `hvec`, plotted on top of the slope field.
 - (c) Lower left contains Figure 2, a graphical representation of errors between the Euler solutions and Runge-Kutta solutions for each step size in `hvec`. `t` is the independent variable, and error is plotted as the dependent variable.
4. Make a visual check that your hand computations match the appropriate plot(s) in Figure 1. Check that the solutions for each method tend to follow the slope field as the step size gets smaller. If all appears correct, print a copy of the display.
5. In the upper left of your printout, neatly and orderly rewrite the work you did for Euler's method in Item 1.

3.4.1 Example

Use Euler's method to approximate a solution to the ODE $x' = 2tx$. By hand, set up the iterating equation and solve for three iterations using the initial condition $x(-2) = 1$ and step size $h = 0.1$. Next investigate both Euler's method and the 4th order Runge-Kutta method graphically using `num3plot` with the four step sizes $h = 0.5, 0.25, 0.1, 0.01$. Identify each plot with its respective step size. Explain the rather odd behavior for $t > 0$ with $h = 0.5, 0.25$ when all plots are compared. You may need to calculate a few Euler iterations by hand for these step sizes. (Does this suggest more than one way that step size can affect results?)

1. Iterating equation $x_{n+1} = x_n + h(2t_n x_n)$ with $h = 0.1, t_0 = -2, x_0 = 1 \implies$

$$\begin{aligned}
 x_1 &= 1 + 0.1(2(-2)1) = 0.6 \\
 x_2 &= 0.6 + 0.1(2(-1.9)0.6) = 0.372 \\
 x_3 &= 0.372 + 0.1(2(-1.8)0.372) = 0.23808
 \end{aligned}$$

2. Change only the following in `initn.m`

```

ftx='2*t*x';
eulerfcn='x+h*(2*t*x)';
hvec=[.5,.25,.1,.01];
t0=-2; tf=2; x0=1;

```

Save, then type `initn` to initialize the variables (or copy-and-paste at the Emporium).

3. Type `num3plot`
4. Here one should investigate and show results such as in Item 4, pp 15, for `eulplot`.

For $h = 0.5$, we have

$$x_1 = 1 + 0.5 (2(-2)1) = -1$$

$$x_2 = -1 + 0.5 (2(-1.5)(-1)) = 0.5$$

$$x_3 = 0.5 + 0.5 (2(-1)0.5) = 0$$

$$x_4 = 0 + 0.5 (2(-0.5)0) = 0$$

Each subsequent iteration will also be zero.

For $h = 0.25$, we have

$$x_1 = 1 + 0.25 (2(-2)1) = 0$$

$$x_2 = 0 + 0.25 (2(-1.75)0) = 0$$

Each subsequent iteration will also be zero.

5. The finished product is on the next page. (Note: your work in the upper left will be neatly hand written, not typed.) Also be sure to identify which approximation goes with which stepsize from `hvec` for all three graphs.

95, §2.3, pp 38

$$x' = 2tx, \quad x(-2) = 1, \quad h = 0.5, 0.25, 0.1, 0.01$$

Euler's Method:

$$x_{n+1} = x_n + h(2t_n x_n)$$

For $h = 0.1$, we have

$$x_1 = 1 + 0.1(2(-2)1) = 0.6$$

$$x_2 = 0.6 + 0.1(2(-1.9)0.6) = 0.372$$

$$x_3 = 0.372 + 0.1(2(-1.8)0.372) = 0.23808$$

For $h = 0.5$, we have

$$x_1 = 1 + 0.5(2(-2)1) = -1$$

$$x_2 = -1 + 0.5(2(-1.5)(-1)) = 0.5$$

$$x_3 = 0.5 + 0.5(2(-1)0.5) = 0$$

$$x_4 = 0 + 0.5(2(-0.5)0) = 0$$

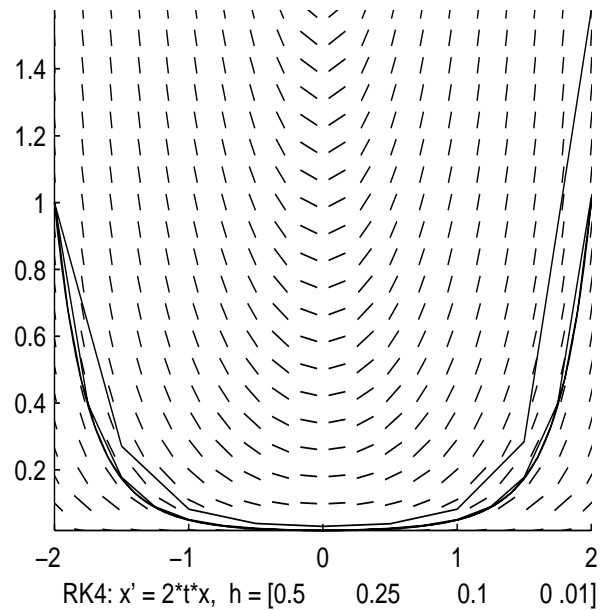
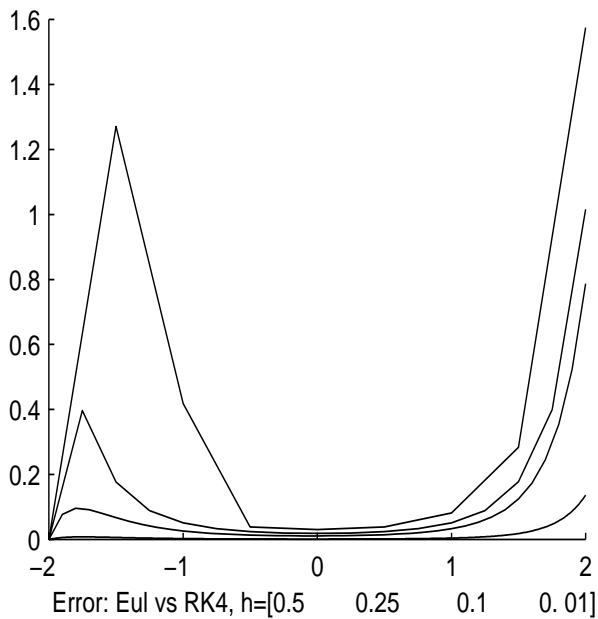
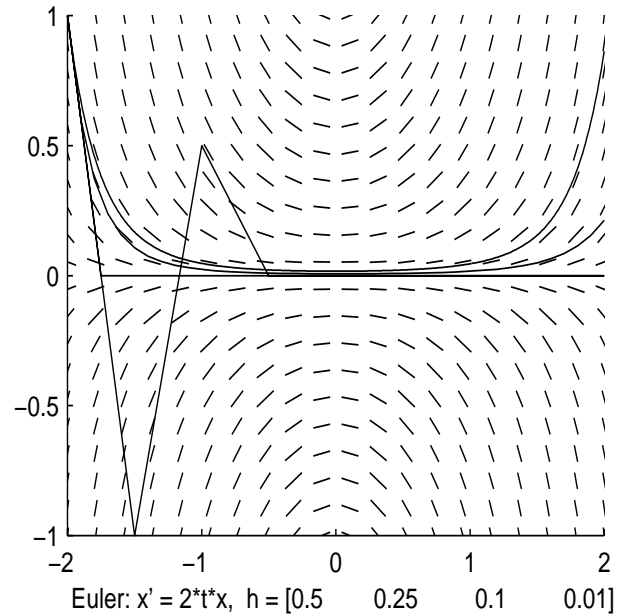
Each subsequent iteration will also be zero.

For $h = 0.25$, we have

$$x_1 = 1 + 0.25(2(-2)1) = 0$$

$$x_2 = 0 + 0.25(2(-1.75)0) = 0$$

Each subsequent iteration will also be zero.



3.5 Using `nsys.m`

The routine `nsys.m` will use the Euler method to solve a 2 by 2 system. To set up the calculation, you need to edit `initsys.m`. For instance, if your system of equations is

$$\begin{aligned}x' &= t + x + y, \\y' &= x^2,\end{aligned}$$

your initial conditions are $x(0) = 2$, $y(0) = 3$, and your desired step size is $h = 0.1$, and you want to compute up to the final time $tf = 0.5$, then `initsys.m` should read as follows:

```
eulerfcn1='x+h*(t+x+y)';
eulerfcn2='y+h*x^2';
h=0.1;
t0=0; tf=0.5; x0=2; y0=3;
```

After you edit `initsys.m`, save it, then run `initsys`, followed by `nsys`. The graphics window will show a plot of x and y versus t , while the command window shows the computed numerical values.

4 Systems of First Order ODEs

Since systems of differential equations are more complicated than single equations, handling them in MATLAB is a bit different. One must submit the RHS of the whole system to one of MATLAB's solvers rather than the RHS of just a single equation. So we can no longer just enter `ftx=...` directly into an `init.m` file as before. For systems, MATLAB requires that the RHS of the system be submitted as a matrix function that is contained in its own M-file. However for many systems, we can create generic function M-files, then supply the parameters specific to the situation in some kind of initialization file. (In our case, `initsys.m`.) To simplify matters, the generic M-files for functions as well as for solving and plotting will be made available to you. For those of you working at home, you can download them. At the Emporium, the files will be loaded onto the computers there. One further simplification will be to generally work in \mathbb{R}^2 so we can obtain relatively simple graphs.

4.1 Required Files and Descriptions

The nine new M-files you will need are: `initsys.m`, `pplane2d.m`, `sys3plot.m`, `cclin.m`, `ccquad.m`, `pend.m`, `plotslnf.m`, `plotxytf.m`, `pplane2f.m`.

initsys.m This is similar to `init.m`. Each time you change the type of system or any of the associated parameters/conditions in the system, this file must be edited, saved, and `initsys` must be typed in the command window (or copy-and-paste at the Emporium).

`t0`, `tf` are the starting and ending times. They will generally be given with the problem.

`x0vec` is the vector containing the initial x -values for each equation in the system. They will generally be given with the problem.

`initax` is the plot window sizing vector **only** for use with `pplane2d.m`. The first two entries correspond to the desired x_1 -range (horizontal). The second two entries correspond to the desired x_2 -range (vertical).

`A` is the matrix of parameters associated with the linear part of the system.

`B` is the matrix of parameters associated with the nonlinear part of the system (if any).

`x1syms` is the symbolic solution $x_1(t)$ to the first equation in the system; e.g., if a system for a 2nd order ODE, `x1syms` is the symbolic solution to the ODE. It must be entered in single quotes.

`x2syms` is the symbolic solution $x_2(t)$ to the second equation in the system; e.g., if a system for a 2nd order ODE, `x2syms` is the symbolic solution to the first derivative $x'(t) = x_2(t)$. It must be entered in single quotes.

`x1syms` and `x2syms` are used for graphing, mainly with 2×2 systems.

`eqn_name` is the name of the MATLAB function M-file that describes the RHS of the system. It must be entered in single quotes AND without the `.m` extension.

`symbolicplot` is the type of plot desired in the lower-left of `sys3plot`. Choosing 'solutions' will plot the symbolic solutions `x1syms` and `x2syms` wrt the time interval `t0` to `tf`. Choosing 'phase' will plot the two symbolic solutions `x1syms` and `x2syms` against each other in a phase plane plot. Choosing 'eigenvals' will plot the eigenvalues of the matrix `A`. Choosing '' will result in no plot in the lower-left. A choice is made by commenting out the others with a `%` at the beginning of each line containing the undesired options.

The rest of the entries in `initsys.m` remain the same for all systems and cases. They are not changed by the user.

pplane2d.m Typing `pplane2d` will create a numerically generated phase plane plot for the system. The ranges for the plot window are taken from `initax` in `initsys.m`.

sys3plot.m Typing `sys3plot` will create a three-plot window like that for `hw3plot` and `num3plot` for comparing MATLAB generated plots and user generated symbolic plots. The types of plots depend on the choice for `symbolicplot` in `initsys.m`.

plotslnf.m, plotxytf.m, pplane2d.m These are plotting routines called by `sys3plot.m`.

The other M-files are function M-files representing systems drawn from modeling physical phenomena.

cclin.m This represents the $n \times n$ linear, homogeneous, constant coefficient system $\mathbf{x}' = \mathbf{A} \mathbf{x}$. Such systems are often derived from n th order, linear, homogeneous ODEs with constant coefficients.

$$\begin{aligned} x'_1 &= a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \\ x'_2 &= a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \\ &\vdots \\ x'_n &= a_{n1} x_1 + a_{n2} x_2 + \cdots + a_{nn} x_n \end{aligned}$$

$$\Leftrightarrow \underbrace{\begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{bmatrix}}_{\mathbf{x}'} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{\mathbf{x}}$$

$$\begin{aligned} \Rightarrow \mathbf{A} &= [a_{11}, a_{12}, \cdots, a_{1n}; a_{21}, a_{22}, \cdots, a_{2n}; \cdots; a_{n1}, a_{n2}, \cdots, a_{nn}]; \\ \mathbf{B} &= []; \end{aligned}$$

ccquad.m This represents the 2×2 autonomous system associated with two first order, quadratic ODEs with constant coefficients. **Competing species** and **predator-prey** models are forms of this system.

$$\begin{aligned}x' &= a_1 x + b_1 y + d_1 xy + \beta_1 x^2 + \gamma_1 y^2 = F(x, y) \\y' &= a_2 x + b_2 y + d_2 xy + \beta_2 x^2 + \gamma_2 y^2 = G(x, y) \\ \iff \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_1 xy + \beta_1 x^2 + \gamma_1 y^2 \\ d_2 xy + \beta_2 x^2 + \gamma_2 y^2 \end{bmatrix} = \begin{bmatrix} F(x, y) \\ G(x, y) \end{bmatrix} \\ \implies \mathbf{A} &= [a_1, b_1; a_2, b_2]; \quad \mathbf{B} = [d_1, \beta_1, \gamma_1; d_2, \beta_2, \gamma_2];\end{aligned}$$

The **linearization** of this system is accomplished via the four partial derivatives

$$\begin{aligned}F_x(x, y) &= a_1 + d_1 y + 2\beta_1 x, & F_y(x, y) &= b_1 + d_1 x + 2\gamma_1 y \\G_x(x, y) &= a_2 + d_2 y + 2\beta_2 x, & G_y(x, y) &= b_2 + d_2 x + 2\gamma_2 y\end{aligned}$$

evaluated at any critical point $(x_0, y_0) \implies$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} \approx \begin{bmatrix} F_x(x_0, y_0) & F_y(x_0, y_0) \\ G_x(x_0, y_0) & G_y(x_0, y_0) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{near } (x_0, y_0)$$

Once the linearization has been formed, we use `cclin.m` with

$$\mathbf{A} = [F_x(x_0, y_0), F_y(x_0, y_0); G_x(x_0, y_0), G_y(x_0, y_0)]; \quad \mathbf{B} = [];$$

to investigate **stability** about each (x_0, y_0) .

The competing species and predator-prey forms are

Competing species

$$\begin{aligned}x' &= \epsilon_1 x - \alpha_1 xy - \sigma_1 x^2 \\y' &= \epsilon_2 y - \alpha_2 xy - \sigma_2 y^2\end{aligned} \implies \begin{aligned}a_1 &= \epsilon_1, & d_1 &= -\alpha_1, & \beta_1 &= -\sigma_1 \\a_2 &= \epsilon_2, & d_2 &= -\alpha_2, & \beta_2 &= -\sigma_2\end{aligned}$$

$$\implies \mathbf{A} = \begin{bmatrix} \epsilon_1 & 0 \\ 0 & \epsilon_2 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} -\alpha_1 & -\sigma_1 & 0 \\ -\alpha_2 & 0 & -\sigma_2 \end{bmatrix}$$

$$\implies \mathbf{A} = [\epsilon_1, 0; 0, \epsilon_2]; \quad \mathbf{B} = [-\alpha_1, -\sigma_1, 0; -\alpha_2, 0, -\sigma_2];$$

Predator-prey

$$\begin{aligned}x' &= ax - \alpha xy \\y' &= -cy + \gamma yx\end{aligned} \implies \begin{aligned}a_1 &= a, & d_1 &= -\alpha \\b_2 &= -c, & d_2 &= \gamma\end{aligned}$$

$$\implies \mathbf{A} = \begin{bmatrix} a & 0 \\ 0 & -c \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} -\alpha & 0 & 0 \\ \gamma & 0 & 0 \end{bmatrix}$$

$$\implies \mathbf{A} = [a, 0; 0, -c]; \quad \mathbf{B} = [-\alpha, 0, 0; \gamma, 0, 0];$$

The `symbolicplot` option '`eigenvals`' is useful only for the linearization.

pend.m This represents the system associated with the 2nd order, nonlinear, homogeneous ODE with constant coefficients that describes the motion of a damped pendulum.

$$\begin{aligned} x' &= & y &= F(x, y) \\ y' &= -\omega^2 \sin x - \gamma y & &= G(x, y) \end{aligned} \iff \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & -\gamma \end{bmatrix} \begin{bmatrix} \sin x \\ y \end{bmatrix}$$

$$\implies \mathbf{A} = []; \quad \mathbf{B} = [0, 1; -\omega^2, -\gamma];$$

The **linearization** of this system is accomplished via the four partial derivatives

$$\begin{aligned} F_x(x, y) &= 0, & F_y(x, y) &= 1 \\ G_x(x, y) &= -\omega^2 \cos x, & G_y(x, y) &= -\gamma \end{aligned}$$

evaluated at any critical point $(x_0, y_0) \implies$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} \approx \begin{bmatrix} 0 & 1 \\ -\omega^2 \cos x_0 & -\gamma \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{near } (x_0, y_0)$$

Once the linearization has been formed, we use `cclin.m` with

$$\mathbf{A} = [0, 1; -\omega^2 \cos x_0, -\gamma]; \quad \mathbf{B} = [];$$

to investigate **stability** about each (x_0, y_0) .

The `symbolicplot` option '`eigenvals`' is useful only for the linearization.

4.2 Using `pplane2d.m`

4.2.1 Example

Create a phase plane plot for the system

$$\mathbf{x}' = \begin{bmatrix} 1 & 1 \\ 4 & 1 \end{bmatrix} \mathbf{x}, \quad \begin{aligned} -2.5 &\leq x_1 \leq 2.5, \\ -2.5 &\leq x_2 \leq 2.5 \end{aligned}$$

(Note that this is a linear, homogeneous system with constant coefficients.)

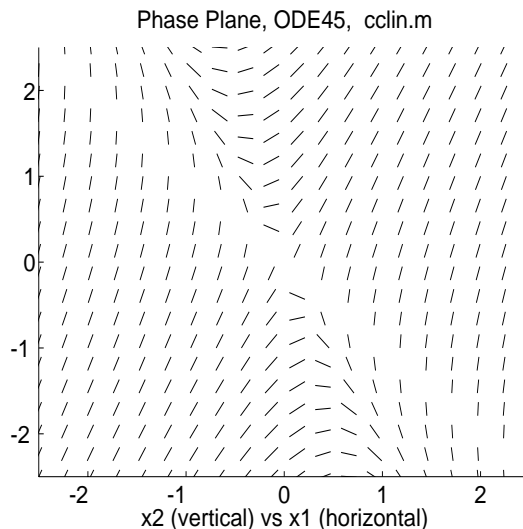
1. Change only the following in `initsys.m`

```
initax = [-2.5, 2.5, -2.5, 2.5];
A = [1, 1; 4, 1]
eqn_name = 'cclin';
```

However make sure there are entries for each of the other variables. (e.g., `B = []`; is okay.) Save `initsys.m`, then type `initsys` (or copy-and-paste at the Emporium).

2. Type `pplane2d`

The plot can be reduced and placed in one of the four corners of the screen using the `subplot` command. Type `help subplot` at the MATLAB command prompt for more information.



4.3 Using `sys3plot.m`

1. Solve the system symbolically (if appropriate).

2. Make the necessary changes in `initsys.m`. Note that the `B` matrix must be given some value, even if it is not used. In such cases, the empty matrix is best; i.e., `B = []`. Choosing the desired `symbolicplot` option is done by commenting the others out; i.e., putting a `%` at the beginning of each line containing the undesired options. Save `initsys.m` then type `initsys` (or copy-and-paste at the Emporium).
3. Type `sys3plot`. This may take up to a few minutes depending on your machine and the difficulty of the ODE system. In your Figure window, you should have three plots.
 - (a) Upper right contains Figure 1, the solution plots $x_1(t)$, $x_2(t)$ generated by MATLAB's ODE45 solver. $x_1(t)$ is the solid line, $x_2(t)$ is the dashed line. Time t is the independent variable.
 - (b) Lower right contains Figure 3, the phase plane portrait of $x_2(t)$ vs $x_1(t)$ as generated by MATLAB's ODE 45 solver.
 - (c) Lower left contains Figure 2: (i) The solution plots $x_1(t)$, $x_2(t)$ generated from your symbolic solution entered as `x1syb` and `x2syb` in `initsys.m`; (ii) the phase plane portrait generated from your symbolic solution entered as `x1syb` and `x2syb` in `initsys.m`; (iii) a plot of the eigenvalues of matrix `A`; or (iv) nothing. This depends on the selected option for `symbolicplot`.
4. Make a visual check that the information in all figures and your "hand work" match/agree. If all appears correct, print a copy of the display.
5. In the upper left of your printout, neatly and orderly rewrite the work you did for Item 1 (if applicable). If you were not directed to solve the system symbolically, use this area to answer any other questions.

4.3.1 Examples

• `cclin.m`

$$\begin{aligned} x_1' &= x_1 + x_2, & x_1(0) &= 3, & x_2(0) &= 2, & t_0 &= 0, & t_f &= 2 \\ x_2' &= 4x_2 + x_2 \end{aligned}$$

$$1. \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 \\ 4 & 1 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \implies \mathbf{x} = \boldsymbol{\xi} e^{rt}$$

$$0 = \begin{vmatrix} 1-r & 1 \\ 4 & 1-r \end{vmatrix} = r^2 - 2r - 3 = (r-3)(r+1) \implies r_1 = 3, \quad r_2 = -1$$

$$\circ r_1 = 3 \implies (\mathbf{A} - 3\mathbf{I}) \boldsymbol{\xi}^{(1)} e^{rt} = \mathbf{0} = (\mathbf{A} - 3\mathbf{I}) \boldsymbol{\xi}^{(1)}$$

$$\implies \text{solve } \left[\begin{array}{cc|c} -2 & 1 & 0 \\ 4 & -2 & 0 \end{array} \right] \implies \dots \implies -2\xi_1 + \xi_2 = 0$$

$$\implies \boldsymbol{\xi}^{(1)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \implies \mathbf{x}^{(1)} = \boldsymbol{\xi}^{(1)} e^{3t} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} e^{3t}$$

$$\circ r_2 = -1 \implies (\mathbf{A} + \mathbf{I}) \boldsymbol{\xi}^{(2)} e^{rt} = \mathbf{0} = (\mathbf{A} + \mathbf{I}) \boldsymbol{\xi}^{(2)}$$

$$\implies \text{solve } \left[\begin{array}{cc|c} 2 & 1 & 0 \\ 4 & 2 & 0 \end{array} \right] \implies \dots \implies 2\xi_1 + \xi_2 = 0$$

$$\implies \boldsymbol{\xi}^{(2)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \implies \mathbf{x}^{(2)} = \boldsymbol{\xi}^{(2)} e^{-t} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} e^{-t}$$

After establishing linear independence, we have the general solution

$$\mathbf{x} = c_1 \mathbf{x}^{(1)} + c_2 \mathbf{x}^{(2)} \iff \mathbf{x} = c_1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} e^{3t} + c_2 \begin{bmatrix} 1 \\ -2 \end{bmatrix} e^{-t}$$

$$\iff \begin{aligned} x_1(t) &= c_1 e^{3t} + c_2 e^{-t} \\ x_2(t) &= c_1 2 e^{3t} - c_2 2 e^{-t} \end{aligned}$$

$$\mathbf{x}(0) = \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} c_1 + c_2 \\ 2c_1 - 2c_2 \end{bmatrix} \implies \begin{aligned} c_1 &= 2 \\ c_2 &= 1 \end{aligned}$$

$$\implies \begin{aligned} x_1(t) &= 2e^{3t} + e^{-t} \\ x_2(t) &= 4e^{3t} - 2e^{-t} \end{aligned}$$

2. Change only the following in `initsys.m`

```
t0 = 0; tf = 2;
x0vec = [3,2];
A = [1, 1; 4, 1];
B = [];
x1symb = '2*exp(3*t)+exp(-t)';
x2symb = '4*exp(3*t)-2*exp(-t)';
eqn_name = 'cclin';
symbolicplot = 'solutions';
% symbolicplot = 'phase';
% symbolicplot = 'eigenvals';
% symbolicplot = '';
```

Save, then type `initsys` to initialize (or copy-and-paste at the Emporium).

3. Type `sys3plot`
4. Make a visual check that your symbolic work matches the appropriate plot in Figure 1. If all appears correct, print a copy of the display.
5. The finished product is on the next page. (Note: Your work in the upper left will be neatly hand written, not typed.) Also note that the MATLAB plots are only as good as the information you supplied for `t0`, `tf`, `x0vec`, and `A`. Remember GIGO (Garbage In, Garbage Out).

Example 1, pp 371: $x_1' = x_1 + x_2, \quad x_1(0) = 3, \quad x_2(0) = 2,$
 $x_2' = 4x_2 + x_2, \quad t_0 = 0, \quad t_f = 2$

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 \\ 4 & 1 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \implies 0 = \begin{vmatrix} 1-r & 1 \\ 4 & 1-r \end{vmatrix} = r^2 - 2r - 3$$

$$\implies r_1 = 3, \quad r_2 = -1$$

$$r_1 = 3 \implies \begin{bmatrix} -2 & 1 & 0 \\ 4 & -2 & 0 \end{bmatrix} \implies -2\xi_1 + \xi_2 = 0 \implies \boldsymbol{\xi}^{(1)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$r_2 = -1 \implies \begin{bmatrix} 2 & 1 & 0 \\ 4 & 2 & 0 \end{bmatrix} \implies$$

$$2\xi_1 + \xi_2 = 0 \implies \boldsymbol{\xi}^{(2)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

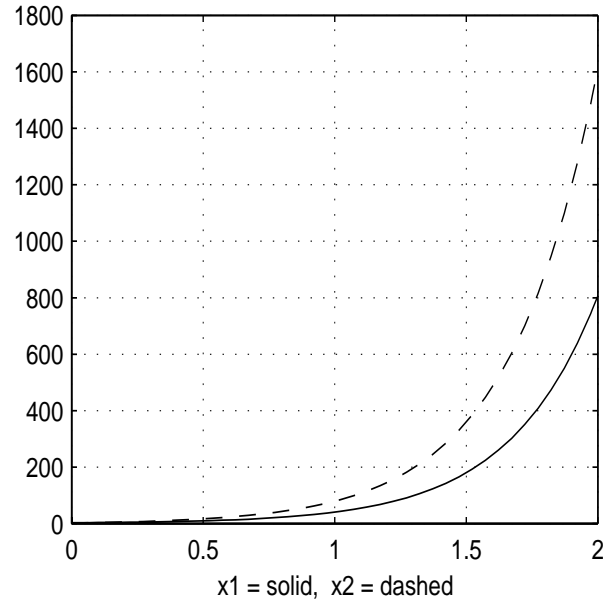
$$\implies \mathbf{x} = c_1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} e^{3t} + c_2 \begin{bmatrix} 1 \\ -2 \end{bmatrix} e^{-t}$$

$$\begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} c_1 + c_2 \\ 2c_1 - 2c_2 \end{bmatrix}$$

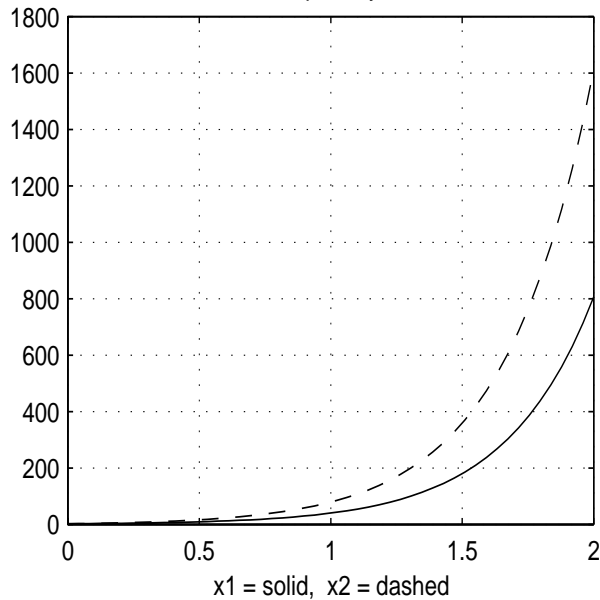
$$\implies c_1 = 2, \quad c_2 = 1$$

$$\implies \mathbf{x} = \begin{bmatrix} 2 \\ 4 \end{bmatrix} e^{3t} + \begin{bmatrix} 1 \\ -2 \end{bmatrix} e^{-t}$$

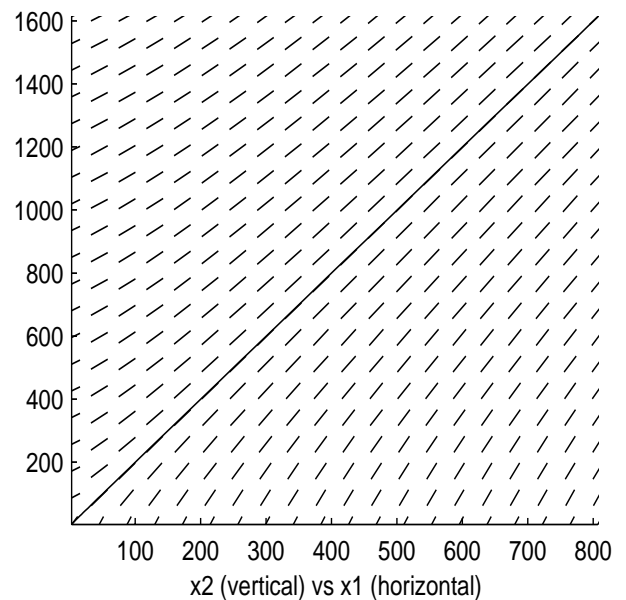
Solution plot, ODE45



Solution plot, Symbolic



Phase Plane, ODE45



5 One Way to Plot Solutions

If one is comfortable with MATLAB, there are easier ways to plot the solution to an ODE than will be given here. However the following uses what is previously contained in this documentation and requires little knowledge of how to program with MATLAB.

5.1 `plotsln.m` – General Instructions

`plotsln.m` is designed to create a MATLAB plot of an explicit solution to an ODE. The plot is generated using information from `init.m`. (Actually `plotsln.m` is general enough to plot any function defined in terms of the independent variable t .)

1. If at home, download `plotsln.m` from the MATLAB files directory via the web page containing the other M-files and save in the appropriate directory. If at the Emporium, `plotsln.m` is already installed.
2. Start MATLAB. At home change to the appropriate directory. At the Emporium copy `init.m` to the Student's Folder.
3. Change only the following variables in `init.m`

`initax=` Enter endpoints that will give some appropriate figure window. You may have to experiment with this, but often the ODE and initial conditions will give you a good indication where to start.

`t0=` This is the given initial t -value.

`x0=` This is the given initial x -value.

`ftxsln=` RHS of your solution equation $x = \phi(t)$.

4. Save `init.m` then type `init` at the command prompt (or copy-and-paste).
5. Type `plotsln`
6. If the plot looks correct, print it.

5.2 Using `subplot`

If you wish to have a smaller plot that is placed toward one of the four corners of the screen/page, you can use the `subplot` command BEFORE using `plotsln`. Here is the full call to place the plot in each of the four corners:

- `subplot(2,2,1)` uses the upper left.
- `subplot(2,2,2)` uses the upper right.
- `subplot(2,2,3)` uses the lower left.
- `subplot(2,2,4)` uses the lower right.

As you might guess, you could plot as many as four plots to a screen/page using `init`, `subplot` and `plotsln` combinations. The following will plot the solutions to two different problems on the same screen/page in the upper right, then lower right positions. (Note: at the Emporium, copy-and-paste `init.m` instead.)

- Save relevant info for Problem 1 in `init.m`.

```
>> init
>> subplot(2,2,2)
>> plotsln
```

- Save relevant info for Problem 2 in `init.m`.

```
>> init
>> subplot(2,2,4)
>> plotsln
```

To return the Figure window to its standard full-picture mode, type `clf` at the command prompt.

5.3 Several plots on the same axes

You can use `plotsln` and the `hold on` command to plot more than one curve on the same set of axes – even though it requires a bit of double-duty. Here is one way to do it for two ODE solution curves. More can be added by extending the process.

1. Do Items 3-5, above, for your first solution. When you have an acceptable window, write down your values from `initax`.
2. Do Items 3-5 for your second solution. Write down your new values for `initax`.
3. Choose the more favorable `initax`-values for both solutions and enter in `init.m`.
4. Do Items 3-5 again for your first solution *except* keep `initax` fixed.
5. Type `hold on` at the command prompt `>>`
6. Do Items 3-5 again for your second solution *except* keep `initax` fixed.

When you are done, type `hold off` to end the `hold` feature.