MATH V23 LECTURE NOTES (Bowen)
Section 2.1
Solution Curves Without a Solution

Unless otherwise stated, it will be assumed in this section that *x* is the independent variable, and *y* is the dependent variable, for each differential equation discussed.

Differential equations without analytic solutions. Earlier in the course, we briefly discussed the possibility that a differential equation might have no analytic or closed-form solution (or an analytic solution that is too difficult to find). However, it is often sufficient to characterize the solutions(s) by creating an approximate graph of the solution (or family of solutions), as the pictorial representation provided by the graph can help us determine the behavior of the solution(s), both locally and generally, even if we do not have equations for them.

Direction fields as a tool to plot approximate solutions to an ODE. This is particularly straightforward for first-order ODEs. A graph of the solution(s) can answer questions such as

* What is the behavior of a solution near a given point?
* How does the solution behave as the independent variable approaches ?

You will get much better results using “real” preprinted graph paper, rather than the “homemade” kind where you draw some or all the grid lines yourself. You may download and print rectangular graph paper PDFs from the course web page before starting the homework assignment for this section, or download your favorite graph paper from another instructor on the Internet. (Or do it the old-fashioned way: buy it at the bookstore. Unused pages carefully removed from the end of an old chemistry lab notebook may also work for you.)

To create the graph, we take advantage of an essential principle of differential calculus; namely, that the derivative of a function, evaluated at a given point, provides the slope of the tangent line to its graph at that point. Given a first-order ODE of the form



(recall from section 1.1 that this is the “normal form of a first-order ODE), we may immediately conclude (given that the existence of the derivative is guaranteed by the above equation) that any solution *y* is *continuous* on its interval of validity *I*. From the precalculus definition of continuity (*i.e.*, that the graph may be drawn without lifting pencil from paper), we determine that the corresponding solution curve has no breaks, meaning no discontinuities, such as vertical asymptotes, jumps, or removable discontinuities (“holes”). From the implied existence of the derivative, we also determine that there is a unique tangent line at each point, at least within *I*. The function  defined above is called the ***slope function*** or the ***rate function***, and it assigns to the point  the slope of the tangent line to a solution function *y*.

We represent this slope graphically at each selected point  by a short line segment called a ***lineal element***. An example of a single lineal element is shown in the text [Zill] in Figure 2.1.1(b). If we systematically plot a large number (dozens or even hundreds) of lineal elements for a collection of  points on a rectangular region of the *xy*-plane, we obtain what is called a ***slope field*** or (more commonly) a ***direction field***. It is customary (although not required) to add rightward-pointing arrowheads to each lineal element, as in Figure 2.1.3(a) in the text.

IMPORTANT: Do *not* take the derivative of  before computing the slopes of the lineal elements! You should be plugging *x* and *y* values from each ordered pair *directly into*  as given. If you examine the first-order ODE stated earlier, you can see that  is *already* equal to the derivative  of the solution(s) *y*. Stated differently, in this context, if *y* is the solution to the equation, then (unlike the usual case in algebra) ; rather, . Perhaps this would have been clearer in the textbook if the author had written the ODE as



as we may not have the same strong mental association between *y* and  as we do between *y* and *f*.

(Aside: If you have taken Physics V05 and plotted electric field lines, this process may feel similar; this is not a coincidence. However, it’s a little easier in the context of DEs, because in physics, if you have multiple source charges, such as a dipole, you must combine two or more electric field arrows as a vector sum to get the “final” arrow at each field point. In this course, you just need to compute a numerical derivative for each field point by plugging its *x* and *y* coordinates into , and “guesstimate” the corresponding tilt of the tangent line, to obtain the arrow for that point.)

If you are detail-oriented, there is an interesting way to use a protractor to draw the tilt angle of each lineal element very accurately. Place your calculator into degrees (not radians) mode, and for each slope *m* that you calculate, use it to compute . The resulting number is the angle (relative to horizontal, stated in degrees) of tilt of the lineal element. Positive angles tilt upward and rightward (increasing function *y*), and negative angles tilt downward and rightward (decreasing function *y*). Using a protractor in this manner is not an expectation of the course (“guesstimating” is fine, as plotting direction fields is a time-consuming process), but it is a way to be precise when needed. To test this, try the slope ; you should get an angle of . If you get 0.785398… or , your calculator is still in radians mode; change it to degrees. Note, however, that for graphs involving trigonometric functions (such as homework problem #4 in the textbook), the calculations are generally performed using radians. In real life, these plots are almost universally generated by computers, not humans.

Figure 2.1.2 of the textbook shows how, once the direction field is obtained, a sample of representative solution graphs may be obtained by drawing curves in such a way that their local tilt at each point matches the direction suggested by the nearest lineal element. The process may naturally involve a bit of guesstimating, especially if there is rapid variation in the directions of the lineal elements within a small region of the graph. The three curves depicted correspond to different initial conditions, and are members of a solution family.

Most of the homework problems ask you to use computer software to create direction fields, and then to plot curves for a selection of initial conditions. Any modern computer algebra system (CAS), such as Maple or Mathematica, should be able to produce direction fields and then plot the curve passing through a given point. However, not everyone has access to these programs (they are not free). At one time, Maple was installed at the BEACH computer lab in the LRC building, but it was an expensive annual license, and our unnamed cost-cutter-in-chief canceled the subscription a year or two ago. If you have a functioning copy of Maple (perhaps from purchasing the textbook for an earlier course), or Mathematica, you should be able figure out how to create these plots by going online and searching for tutorials. Fortunately, if you don’t have a copy of one of these programs, there are free alternatives available.

My recently-discovered favorite alternative is wxMaxima, which is free, open-source (GNU GPLv2), and reasonably well documented. It is a GUI for Maxima, a software based on MACSYMA-DOE that was originally developed for use on Unix/Linux-based systems via command line in a Terminal session. Fortunately, it’s now possible to download binaries that function reasonably well (they are slightly clunky) on Windows boxes; get your copy (if you’d like) at <https://sourceforge.net/projects/maxima/files/Maxima-Windows/> by clicking the green “Download Latest Version” button. (See the screenshot below where it says CLICK HERE.) Note that this is about a 100-megabyte download; you might (or might not) get it quicker if you use a campus Internet connection. Find the file in your Downloads folder, and double-click to install.



Let’s try using wxMaxima to create a direction field plot. Consider the equation



(note that the derivative is defined implicitly, as it contains both *x* and *y*). The wxMaxima function that creates direction fields is called “plotdf.” In its simplest form, “plotdf” assumes it is being given an equation of the form , so all we need to do is tell it what  is. After we open an instance of wxMaxima, we start by typing the command

load("plotdf");

including the semicolon at the end of the line, because the “plotdf” package is not loaded into memory by default. To send each command to the software to be processed, press the keyboard combination SHIFT+ENTER (not just the ENTER button). The system response should look something like

(%i1) load("plotdf");

(%o1) "C:\maxima-5.38.1\share\maxima\5.38.1\_5\_gdf93b7b\_dirty\share\dynamics\plotdf.lisp"

(%i2)

The blank line starting with “(%i2)” is where wxMaxima is expecting us to type our next command. In wxMaxima syntax, functions such as sine and cosine require parentheses around the arguments, and multiplications must be explicitly typed using the asterisk symbol. To obtain a direction field, we define the given equation to wxMaxima as

plotdf(sin(x)\*cos(y));

then press SHIFT+ENTER again to see the results, which show in an interactive pop-up window. If we simply click on the plot, we get a superimposed plot of the specific solution passing through the point on which we clicked. So, for example, if we click on  (or as close to it as we can get), we find the specific solution having initial conditions . Clicking on other points of the graph allows us to simultaneously display many members of the family of solutions. Clicking on the “replot” icon  re-creates the direction field with our most recently selected specific solution. Clicking on the “config” icon  brings us to a menu where we may resize the axes, pick initial conditions for a specific solution (for , the syntax is “0 0”), select colors of the lineal elements and the graph of the specific solutions, specify whether we want the specific solution graphed over its entire interval of validity, or only to the right of the initial condition, and so on. By adding complexity to the original command, we may also cause these settings to appear on the graph when it is first drawn. Here are some examples:

* Create the direction field with a specific solution (“trajectory”) already included (in this case, the one that passes through ):

plotdf(sin(x)\*cos(y),[trajectory\_at,3,4]);

* Show the same as above, but only the portion of the trajectory to the right of  (“backward” may be used instead of “forward” to show the trajectory only to the left of):

plotdf(sin(x)\*cos(y),[trajectory\_at,3,4], [direction,forward]);

* Show the same as above, but only the first quadrant, with the upper right corner of the graph at :

plotdf(sin(x)\*cos(y),[trajectory\_at,3,4], [direction,forward],
[x,0,10], [y,0,12]);

* Show the same as above, with a graph of  also superimposed:

plotdf(sin(x)\*cos(y),[trajectory\_at,3,4], [direction,forward],
[x,0,10], [y,0,12], [xfun,"1+cos(x)"]);

* Show the same as above, with coefficients (parameters) and sliders (to adjust the values of the parameters in real time to “play” with the graph) added (on my system, moving the sliders temporarily removes the “xfun” curve until the next time the “replot” icon is clicked; this may be either a bug or a feature of the software):

plotdf(sin(m\*x)\*cos(n\*y),[trajectory\_at,3,4], [direction,forward],
[x,0,10], [y,0,12], [xfun,"1+cos(x)"], [parameters,"m=1,n=1"],
[sliders,"m=0.1:5,n=0.1:5"]);

The above additional arguments to “plotdf” may be used in any combination, but you must be careful about brackets, commas, and quotes to avoid triggering syntax errors. Additional arguments are documented in the “plotdf” reference, available at <http://maxima.sourceforge.net/docs/manual/de/maxima_66.html>. A tutorial with different examples and information is also available at <http://www.walkingrandomly.com/?p=2079>, but it uses a more complicated parametric syntax to specify .

Isoclines. For a first-order ODE of the form , an ***isocline*** is a curve on the *xy*-plane for which the value of  is constant; for example,  or . In general, isoclines do not coincide with the curves representing members of the family of solutions. However, when preparing a direction field plot by hand, starting by drawing one or more isoclines may help obtain the direction field faster. This is because every point on an isocline corresponds to a lineal element having the same slope (same tilt) as every other lineal element on the isocline. If one is using the protractor method to draw the direction field, it means that every point on an isocline has a lineal element that is tilted at the same angle, so that angle may be used for several lineal elements instead of just one, reducing the number of angle calculations required.

Autonomous first-order ODEs. For a first-order ODE of the form , if the function on the right side of the equation is written using *y* only (that is, *x* is not visible, or ), then the equation is said to be ***autonomous***. If the differential equation models a physical process (such as the examples discussed in section 1.3 of the textbook [Zill]) for which the independent variable is *t* (time), then *t* does not explicitly appear in the differential equation, and so the process is called ***time-independent***. Examples of time-independent processes include the population dynamics, radioactive decay, Newton’s law of cooling, and LRC circuit illustrations in section 1.3.

Critical points of an autonomous ODE. For an autonomous ODE , the zeros of  are called ***critical points***, ***equilibrium points***, or ***stationary points***. In this context, the critical points are where , not (as in algebra) where . As in calculus, the terminology refers to points on the solution curve at which the value of the derivative  is equal to zero. (Some calculus texts also include, as critical points, those places where the derivative is undefined, such as cusps, discontinuities, and vertical tangents. We are specifically excluding those types of “critical points” in this discussion.)

If  (where *c* is a constant) is a critical point of an autonomous ODE, then  is a solution of the ODE. Such constant solutions are called ***equilibrium solutions***, and are the only constant solutions of an autonomous ODE. In the Newton’s law of cooling example from section 1.3,  is the equilibrium solution; if the “hot” object’s initial temperature is equal to that of the surroundings, then its temperature will remain constant at that value for as long as the temperature of the surroundings does not change.

Non-equilibrium solutions of an autonomous ODE will be either increasing or decreasing functions, depending on whether the sign of  is positive or negative, respectively. These may be represented using a ***phase portrait***. Example 3 in the textbook illustrates an example of a phase portrait. For the Newton’s law of cooling example, the phase portrait would look like the diagram at right.

The arrowheads indicate that if the initial temperature of the “hot” object was greater than that of the surroundings (higher than  on the axis), the value of the solution  would decrease (downward-pointing arrowhead) as time increased (the object would cool down). If the object started with a temperature below that of the surroundings (lower than  on the axis), then  would increase (upward-pointing arrowhead; the object would warm up).

Attractors and repellers. In the Newton’s law of cooling example, the solution curves move toward the equilibrium solution  as the independent variable approaches positive infinity (), regardless of whether the object’s initial temperature was above or below . For this reason,  is called an ***attractor***, or a ***stable solution***. If the solution curves move away from an equilibrium solution as , then that solution is called a ***repeller***, or an ***unstable solution***. It is also possible to have an equilibrium solution for which solution curves may move both toward (*e.g.*, if the initial condition is a point below the equilibrium solution) and away from (*e.g.*, if the initial condition is a point above the equilibrium solution) the equilibrium solution; such solutions are called ***semi-stable***. Phase portraits for semi-stable solutions are shown in Figures 2.1.9(c) and 2.1.9(d) in the textbook.

Translation property of autonomous ODEs. Because autonomous ODEs do not explicitly include *x*, the derivative at any point only depends on the y-coordinate of that point. In a direction-field plot, this means that the slope of each lineal element on the plot depends only on how far the corresponding point is located above or below the *x*-axis. From this, it immediately follows that a horizontal translation (shift)  of any solution  of an autonomous ODE is also a solution of the autonomous ODE, where *h* is a constant indicating the magnitude (that is, the size, in *x*-axis units) of the shift. (The textbook uses *k* for the constant, but since most textbooks use *k* to represent a vertical shift, I will use *h* in lecture.)