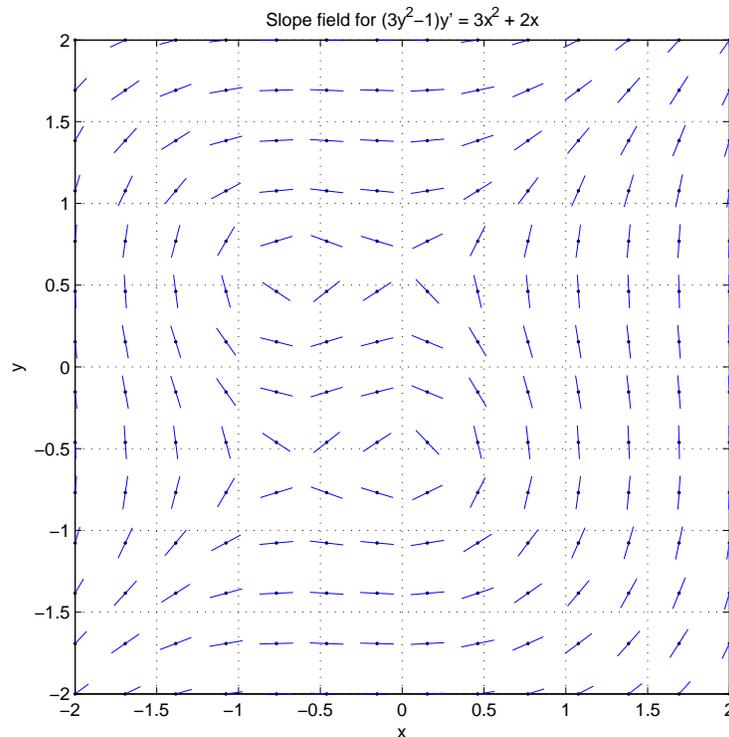


MECH 221 Computer Lab 4: A Separable ODE

OVERVIEW: Slope Fields, Separable Solutions, and Contour Plots

The sketch below shows a *slope field* for the ODE

$$\frac{dy}{dx} = \frac{3x^2 + 2x}{3y^2 - 1}. \quad (*)$$



The idea is simple: at each point (x, y) , the right side in equation $(*)$ specifies a particular value of the slope dy/dx . We can draw a short line segment having (x, y) as its midpoint and dy/dx as its slope. The figure above is produced by doing exactly this, to illustrate the slopes at a uniform grid of (x, y) points. In a geometrical interpretation, the act of “solving ODE $(*)$ ” consists of finding the curves in the (x, y) -plane whose slopes are compatible with the slope field shown above. The goal of today’s lab is to use the computer to help with this, and to finish with a counterpart of Figure 1 that shows some solution curves computed with extreme accuracy.

Notice that ODE $(*)$ is *separable*—that is, it can be rearranged into the equivalent form

$$(3y^2 - 1) dy = (3x^2 + 2x) dx$$

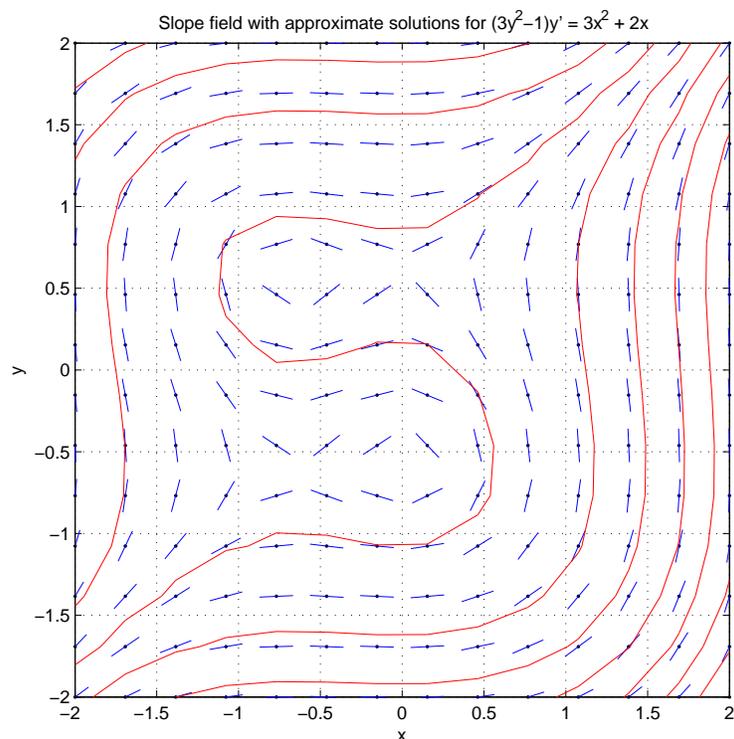
in which all the y -dependence is on one side and all the x -dependence is on the other. Integrating both sides leads to an equation of the generic form $\phi(y) = \psi(x) + K$ for single-variable functions ϕ and ψ and some constant of integration K . This can be rearranged into the general form

$$g(x, y) = C, \quad (**)$$

where C is a constant and $g(x, y)$ is defined by a suitable combination of $\phi(y)$, $\psi(x)$, and constants. Your prelab assignment involves finding an explicit formula for g in the case of ODE $(*)$. The

methods developed today will apply in any problem where an equation like (**) requires further study.

An equation like (**) typically describes a whole family of curves in the (x, y) -plane, one for each choice of the constant C . Every such curve could be called a *contour for g* , or a *level curve for g* , or an *iso- g curve*. Matlab has a built-in command, `contour`, that draws such curves. Using it with the slope field calculated earlier and exact knowledge of the function g produces this picture:



A crude computed version of the main idea is clearly visible. Using the exact values of $g(x, y)$ at the nodes in the sketch, Matlab has used interpolation to approximate the true mathematical contours compatible with those point-values. But the coarseness of the grid does cause some problems: the basic idea is to match tangent slopes, and the piecewise-linear contours in the picture have corner points, not tangent lines, at many points! Our goal is to produce better versions of the curves shown in red here, using an approach more sophisticated (and accurate) than simply repeating the steps above with some absurdly large number of grid points.

DETAILS: *Finding Zeros, Finding Brackets, Joining Subarcs*

Root Finding. Suppose we have identified the particular value of C associated with some solution curve of interest. Let's illustrate with $C = -0.3$, a rough approximation for the value attached to the S-shaped red curve near the middle of the sketch above. (In the prelab, you should have produced a different C -value, corresponding to a nearby curve.) Then, suppose we need to know the y -value on this curve associated with the point $x = -0.5$. Finding y from equation (**) requires solving

$$g(-0.5, y) = -0.3, \quad \text{i.e.,} \quad 0 = f(y) \stackrel{\text{def}}{=} g(-0.5, y) - (-0.3). \quad (\dagger)$$

Since the function $g(x, y)$ and the number C are both known, we have a clear definition of f to use here. What we need is some automated tool that solves for y in an equation like $f(y) = 0$. What we need is what we have: our function `findzero` from Computer Lab 3 seems purpose-built for this application!

Bracketing Intervals. The S -shaped curve associated with $C = -0.3$ appears to cross the vertical line $x = -0.5$ three times. Thus we expect not one, but three different solutions for y in equation (†). To make sure we find them all, we exploit a design feature of our function `findzero`: it returns a zero of the given function inside a specified bracketing interval. To find the three y -values of interest requires three separate appeals to `findzero`, each with a different bracketing interval. A possible approach could use the commands

```
y1 = findzero(f, [0.75, 1]);    % One value seems close to 0.90
y2 = findzero(f, [0, 0.25]);   % One value seems small but positive
y3 = findzero(f, [-1.2, -0.8]); % One value seems close to -1.00
```

Here we have carefully crafted three disjoint bracketing intervals that we hope will capture the three roots of interest, using the approximate information we already have. It would be ideal to make this process more automatic, so the computer could choose bracketing intervals automatically, and adjust them for different x -values. This will be a key element of today's activities.

Solution Arcs. Look again at the S -shaped curve defined by $g(x, y) = -0.3$ in the sketch above. It is not the graph of a function, because some x -values are associated with more than one y -value. But if we select the largest y -value whenever there is a choice, it makes sense to speak of “the top branch” of the curve, and it looks like this should be smooth and well-defined at least for the region where $x \geq 1$. To plot this branch, we could use a computerized loop to visit a sequence of points x in the interval $-1 \leq x \leq 2$. For each x , we would invent a bracketing interval $[a(x), b(x)]$ of y -values that includes the y -value we seek, and then give commands something like this:

```
% At this point, x contains a number and [ax,bx] has been computed
f = @(y) g(x,y) + 0.3;      % Define the function whose root we seek
y = findzero(f, [ax,bx]);
```

In order to trace “the bottom branch” of the curve, we would focus on the interval $-2 \leq x \leq 0$, and use a different pair of formulas to define the bracketing interval $[a(x), b(x)]$ for candidate y -values. Likewise for “the middle branch”, where $-1 \leq x \leq 0$.

The top, middle, and bottom branches described above fail to capture the near-vertical parts of the S -shaped curve near the vertical lines $x = -1$ and $x = 0.5$. To do a good job on those, imagine rotating the picture one quarter turn. Pick a y -value like $y = 0.5$. The point $(x, 0.5)$ is on our curve if and only if

$$g(x, 0.5) = -0.3, \quad \text{i.e.,} \quad 0 = f(x) \stackrel{\text{def}}{=} g(x, 0.5) + 0.3.$$

Except for a new definition of f , this problem has the same form we have just been discussing. We can replace the specific $y = 0.5$ with a `for`-loop that visits a whole sequence of y -values. If we have some automatic way of producing a bracketing interval $[a(y), b(y)]$ that outlines the candidate x -values for each level, we can say `x = findzero(f, [ay, by]);` to generate an x -coordinate to complete each (x, y) -pair.

Patchwork. A general strategy is now becoming clear. Accurately tracing the S -shaped curve described above calls for computing and plotting 5 separate pieces. The top, middle, and bottom arcs are built by solving for y given x , with x in three appropriate subintervals of $-2 \leq x \leq 2$. The left and right arcs are built by solving for x given y , using y values in two well-chosen subintervals of $-2 \leq y \leq 2$. Using Matlab's command `hold on` allows any number of arcs to be added to the same set of axes, so we can make a single accurate picture from these 5 computations.

Attention. The numerical values outlined above, like $C = -0.3$ for the curve and $x = \pm 1$ for the transition points between sub-arcs, were provided only to focus the discussion. Your work in the lab will typically involve different numbers from the ones shown here.

LAB ACTIVITIES: *Producing an Accurate Graphic*

1. Get `slopefield.m` for Connect and run it. You should get the basic slope field associated with ODE (*), as shown in this writeup and in the prelab, with a red contour curve that is clearly wrong. That's because the function $g(x, y)$ and constant C described in line (**) above have been entered incorrectly. You found the right values in the prelab. Put those in and run the script again. The result should be roughly compatible with the plots shown above, but obviously not accurate enough for any serious purpose.

Grant yourself a few minutes to play around. Try changing the number of nodes used by `slopefield` to improve the picture. Or, adjust the details in the call to the `contour` function so it draws a few more curves. (Say `help contour` to learn about your options.)

2. Edit `slopefield.m` so that the script draws not only the slope field, but also the decorations you drew on paper in the prelab. In detail, the picture should now have one red contour *and*
 - The lines along which $\frac{dy}{dx}$ is undefined.
 - The lines along which $\frac{dy}{dx} = 0$.
 - The conic section associated with $\frac{dy}{dx} = 1$.
 - The closed loop associated with $\frac{dy}{dx} = -1$.
3. Add commands to the script to make it calculate the three values of c for which $(-0.8, c)$ lies on our contour of interest. Call these numbers c_1, c_2, c_3 , and label them from smallest to largest, so $c_1 < c_2 < c_3$. Arrange for two of these points, $(-0.8, c_2)$ and $(-0.8, c_3)$, to appear on the plot. A command like the following should achieve this:

```
plot(-0.8,c2,'ro','MarkerSize',12);
```

4. Add commands to the script to make it apply `findzero` to calculate the three values of d for which $(0.5, d)$ lies on our contour of interest. Call these numbers d_1, d_2, d_3 , and label them from smallest to largest, so $d_1 < d_2 < d_3$. Arrange for two of these points, $(0.5, d_1)$ and $(0.5, d_2)$, to appear on the plot.

Notes: (i) When we built function `findzero`, we filled it with printing commands to produce informative output. Now that we know it works, and we need to use it more than 500 times in every application of our script, all this printing is just noise. Consider editing `findzero` to remove the printing. (If you do, save a copy of your beautiful original first.)

- (ii) If you never got `findzero` to work, just use Matlab's built-in function `fzero` instead. This substitution will cost you some of the joy of self-sufficiency, but the penalty in marks

will be minor. The design specs for `findzero` in Computer Lab 3 were carefully arranged so that `fzero` would be very nearly a drop-in replacement at this point. (`fzero` has more options, but if you call it like `findzero`, then it will work in almost the same way. The only difference is that our `findzero` returns a short interval containing the desired root in the form of a 2-element vector, whereas Matlab's `fzero` returns a single scalar approximation to the root. It's easy to adapt to this small difference.)

5. The four dots plotted in items 3–4 will provide the transition points between five sub-arcs on our contour of interest. Working from bottom to top, let's give these the names bottom, right, middle, left, and top. Focus on the bottom arc, with associated interval $-2 \leq x \leq 0.5$. Invent functions $a(x)$ and $b(x)$ so that for each x in this region, the interval $[a(x), b(x)]$ brackets the y -value on the bottom arc. Then make a vector of 101 equally-spaced x -values from -2 to 0.5 , and use `findzero` to populate a vector with the 101 corresponding y -values on the bottom arc. Plot the points you produce on the slope-field sketch.

Hint: Careful study of the extra guidelines added to the picture in Step 2 might inspire suitable *constant functions* for use as $a(x)$ and $b(x)$.

6. Repeat the work in item 5 for the four remaining arcs. Use the interval $-0.8 \leq x \leq 0.5$ for the middle arc and $-0.8 \leq x \leq 2.0$ for the top, but switch orientations and use $d_1 \leq y \leq d_2$ for the right and $c_2 \leq y \leq c_3$ for the left. Each sub-arc should have 101 points.

Hand-in Checklist.

- A single plot, with the student name in its computer-printed title, showing
 - the slope field as originally given,
 - four lines and two conics adding further info about the slope field,
 - a rather dreadful approximation to the desired curve generated by the `contour` function,
 - a highly accurate trace of the desired curve generated by piecing together five sub-arcs,
 - four prominent markers showing the nodes where the sub-arcs are connected end-to-end.
- The six values c_1, c_2, c_3 and d_1, d_2, d_3 described in items 3–4, each accurate to 6 or more significant digits.
- A printed copy of the ultimate script `slopefield.m`, in which
 - the student's name, lab group, and UBC ID appear in the electronic file and therefore on the printout, and
 - the student's choice of bracketing intervals for each sub-arc computation is highlighted in some way, so the marker can easily locate and check it.