Slope Fields and Euler's Method for AP Calculus

Dennis Pence
Western Michigan University
Kalamazoo, Michigan 49008
dennis.pence@wmich.edu

Introduction

Numerical methods such as Euler's method and visualizations such as slope fields are now a part of the AP Calculus syllabus. Both of these topics have been in the BC Course since 1998, and it has been announced that slope fields will be added to the AB Course in 2003 <http://www.collegeboard.org/ap/calculus/index.html>. Graphing calculators make these topics easy to implement and fun to study.

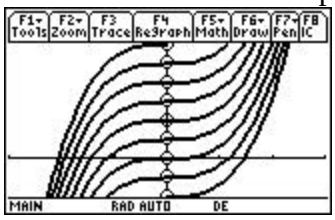
First Order Differential Equations

Soon after a calculus class learns about the derivative, the instructor can introduce the concept of a first order differential equation.

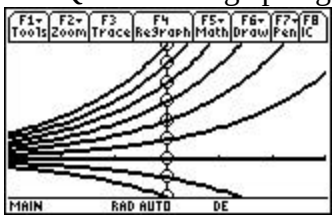
$$\frac{dy}{dt} = f(t, y)$$

Here we use the independent variable t for two reasons. First, many applications from physics and engineering have *time* as the independent variable. Second, when we move to a system of two first order equations, x and y, we will plot $(x(t), y(t))$ as parametric equations. In the TI graphing calculators with a differential equations graphing mode (now TI-85, TI-86, TI-89, TI-92 Plus), the graphing variable is also t.

The general solution for a first order differential equation (if it exists) is usually a family of functions. Here are some examples using DIFF EQUATIONS graphing on a TI-89.



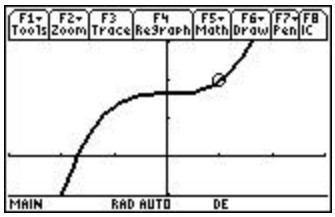
$$\frac{dy}{dt} = t^2 \quad \Rightarrow \quad y = \tfrac{1}{3}t^3 + C$$



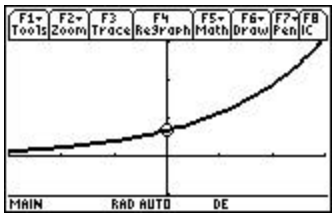
$$\frac{dy}{dt} = 0.5y \quad \Rightarrow \quad y = Ce^{0.5t}$$

Presented with a candidate for a general solution (no matter how it was obtained), we can easily check by differentiation and substitution into the equation.

A particular solution (from this family) is specified by, in addition, giving an initial condition $y(t_0) = y_0$. If we have been able to solve for the exact family as in the examples above, then we can substitute the initial condition into the formula for the family to determine the arbitrary constant in the formula.



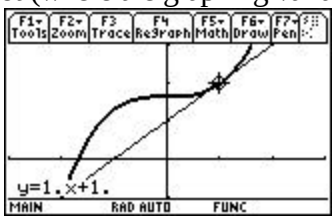
$\frac{dy}{dt} = t^2,\ y(1) = 2 \quad \Rightarrow \quad y = \frac{1}{3}t^3 + \frac{5}{3}$

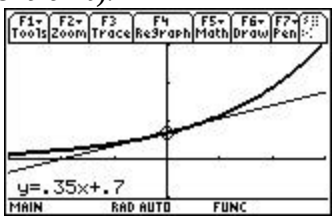


$\frac{dy}{dt} = 0.5y,\ y(0) = 0.7 \quad \Rightarrow \quad y = 0.7e^{0.5t}$

When viewing a differential equation plot on a TI-89, or TI-92 Plus, the command F8 IC allows you to move the cursor to a location in the viewing window (or to type the location) to specify another initial condition interactively. On the TI-86, the command is EXPLR with only the option to move the cursor.

Both slope fields and Euler's method are generally based upon the idea that a tangent line is a good approximation for the graph of the function *near* the point of tangency. Notice we needed to move to FUNCTION graphing to get the solution and the tangent line in the same plot (where the graphing variable is x rather than t).





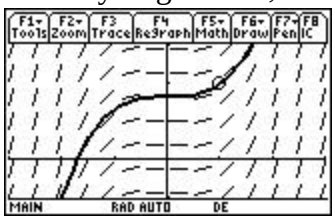
Slope Fields

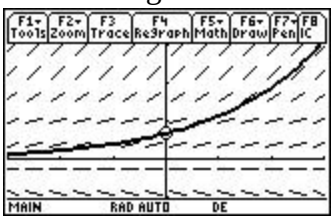
Usually the family of the general solution has a member going through every point in our viewing window (or perhaps some subset of this region). We can compute the tangent line for the particular member of the family passing through the point (a, s) without knowing the formula for the particular solution.

$y(a) = s$ to go through the desired point (a, s).

$\frac{dy}{dt}(a) = f(a, y(a)) = f(a, s)$ to satisfy the differential equation.

A slope field (also called a direction field or a phase portrait) is the plot of a small line segment of many tangent lines, each centered at a point in the viewing window.





After seeing how particular solutions "flow through" the slope field a few times, students learn to sketch an approximate solution, given only the slope field.

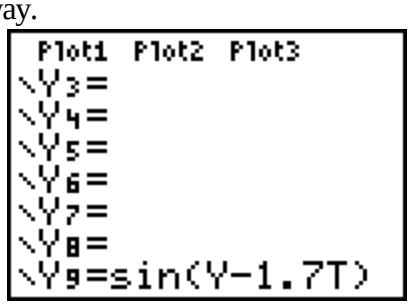
Programming Slope Fields on the TI-83

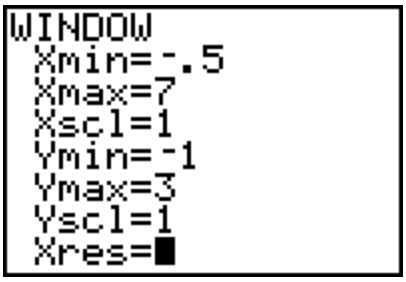
Here are the steps for getting a "DRAWN" slope field on a TI-83. Similar steps work for other TI graphing calculators with no built-in slope field formatting (including the TI-85).

The first step is to realize that you can put a formula into the Y= Editor with several variables (or parameters) other than the graphing variable X. We show this on an example with no elementary exact solution.

$$\frac{dy}{dt} = \sin\left(y - 1.7t\right) \qquad \text{Viewing Window: } -0.5 \le t \le 7,\ -1 \le y \le 3$$

I like to use the function slot $\mathbf{Y}_9$ for the purpose of storing the differential equation. Make sure to "deselect" this formula, because we do not want it to try to graph in the usual way.

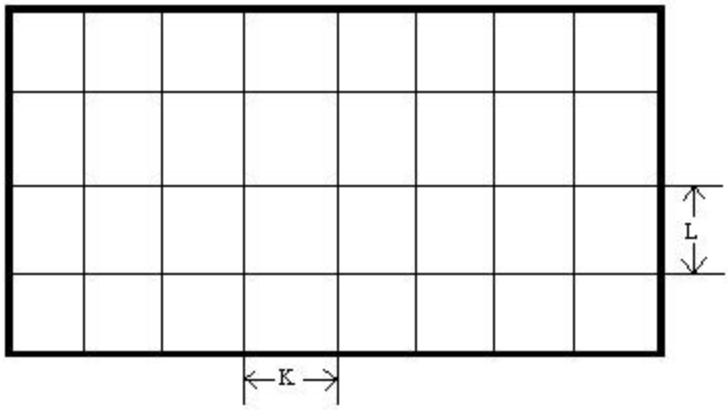




To "evaluate" this formula for certain values of Y and T, simply store the appropriate values in Y and T and type $\mathbf{Y}_9$. [Note: if you leave this slot selected and try to graph you will get an error message. The calculator treats all letters other than X as parameters. It links to the memory locations to get the needed values for these parameters. However the calculator itself uses the variable Y in plotting. When Y changes after plotting the first point, the link is broken and the error message appears.]

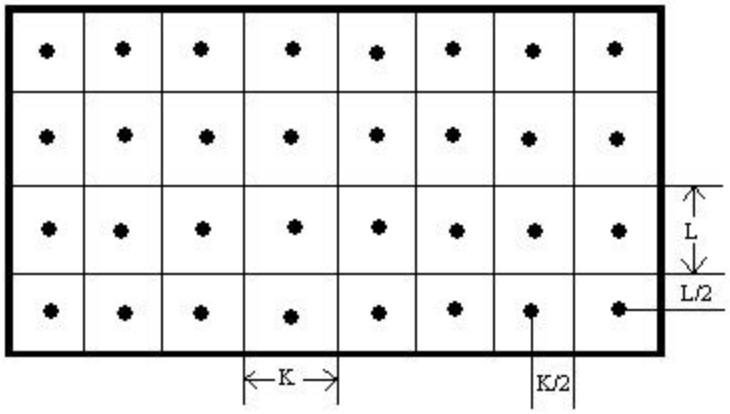
The second step is to plan the grid of points where you desire to plot slope line segments. In the figure below,

$$K = \frac{X\ \max - X\ \min}{8}, \qquad L = \frac{Y\ \max - Y\ \min}{4}$$

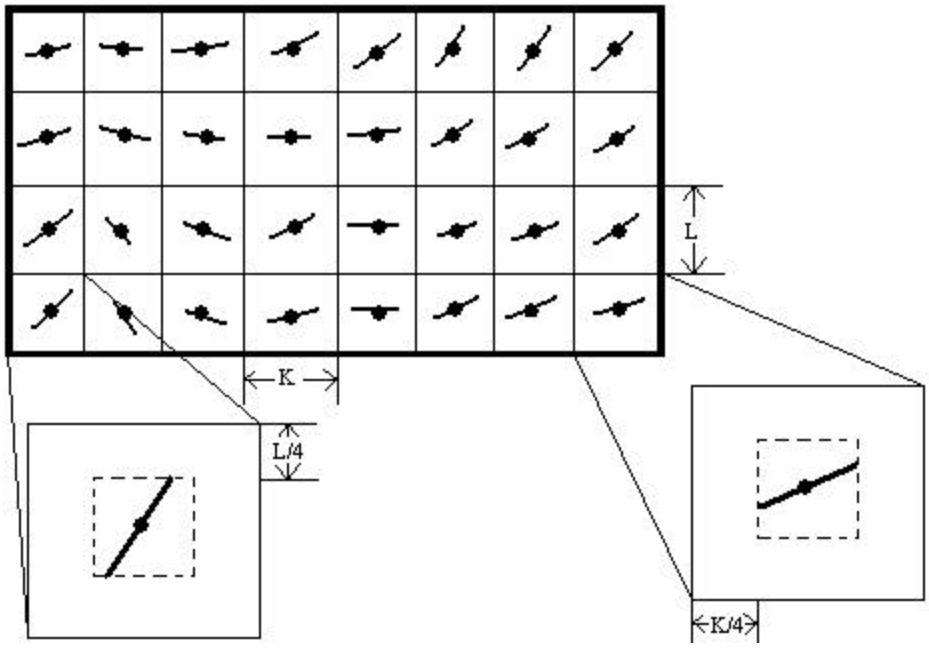


We do not want the points to be on these grid line intersections, but rather centered in these subrectangles. Thus we need to find the centers of each of these subrectangles. In our example figure, this will be the following.

$$x_i = X\ \min\ -\frac{K}{2} + iK, \quad i = 1,\cdots,8, \qquad y_j = Y\ \min\ -\frac{L}{2} + jL, \quad j = 1,\cdots,4$$



Next we work out how to draw the line segments. Suppose that we have stored the coordinates for one of the grid points in to T and Y. We evaluate $\mathbf{Y}_9$ and store this slope in M. Ideally all of the line segments would be drawn the same length. This would require either trigonometric function evaluations or complicated square roots that would just be too slow. Our compromise it to require that the line segment stay in the small dotted box in the figure below.



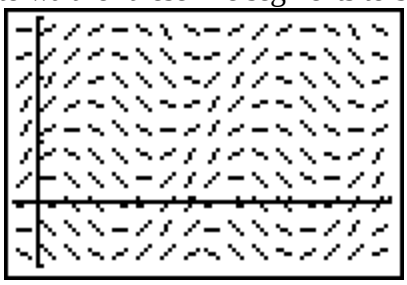
The absolute value for a slope going exactly to the corner of the dotted box will be L/K. Thus when abs(M) ≤ L/K, determine the endpoints of our line segment by the intersection of the tangent line with the vertical lines $x = T - K/4$ and $x = T + K/4$. When

abs(M) ≥ L/K, determine the endpoints of our line segment by the intersection of the tangent line with the horizontal lines $y = Y - L/4$ and $y = Y + L/4$. The TI-83 program below implements these ideas (for more grid points than in the simplified figures above). Note that the draw command Line(also changes the coordinate variable Y. Thus we store its value in Z before we issue this command, and then use Z instead.

TI-83 prgmSLOPE

`FnOff`	`B+I*K®T:D+J*L®Y`
`Xmin®A`	`Y®Z`
`(Xmax-A)/16®K`	`Y₉®M`
`Ymin®C`	`If abs(M)£S:Then`
`(Ymax-C)/10®L`	`Line(T-P,Z-MP,T+P,Z+MP)`
`A-K/2®B:C-L/2®D`	`Else`
`L/K®S`	`Line(T-Q/M,Z-Q,T+Q/M,Z+Q)`
`K/4®P:L/4®Q`	`End`
`For(I,1,16,1)`	`End`
`For(J,1,10,1)`	`End`

Running this program gives the following slope field for the differential equation we have stored $\mathbf{Y}_9$ = `sin(Y-1.7T)` in the viewing window set. You can experiment with the numbers 16 and 10 (each appears twice in the program) balancing how long you are willing to wait for these line segments to be draw with resolution.



Euler's Method for Numerical Solution

Many differential equations do not have an exact symbolic solution. Even some, that do have an exact solution, have such a complicated formula that a numerical solution might be more useful. We seek to approximate a solution $y(t)$, $t_0 \le t \le t_f$ to a first order differential equation $dy/dt = f(t, y)$ with initial conditions $y(t_0) = y_0$. We decide to accept N values for approximations to y evaluated at equally spaced t-values. Let

$$h = \frac{t_f - t_0}{N}; \qquad t_i = t_0 + ih, i = 1, \cdots, N \ .$$

We know $y(t_0)$ by the given initial condition. We also know the slope of the line tangent to y at the point (t_0, y_0) to be $m_0 = f(t_0, y_0)$ from the differential equation. This "first" tangent line has equation

$$y - y_0 = m_0 \left(t - t_0\right).$$

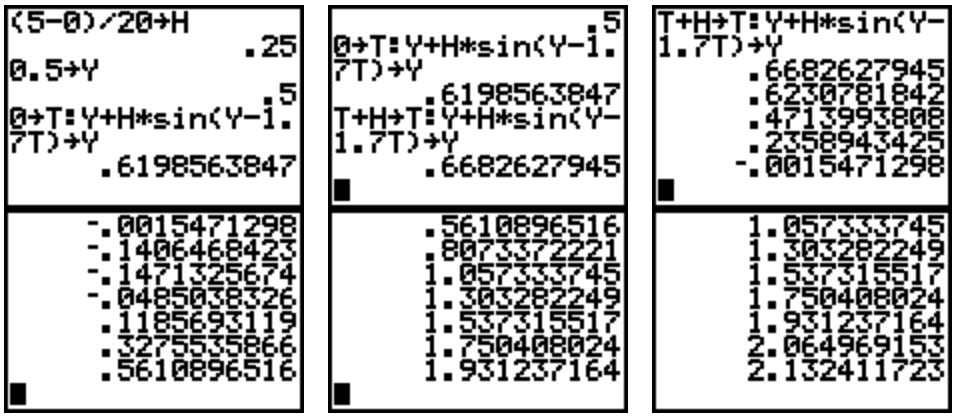
We simply approximate $y(t_1)$ by evaluating the tangent line equation at $t = t_1$. This gives

$$y(t_1) \approx y_1 = y_0 + m_0\,(t_1 - t_0) = y_0 + h\,f(t_0, y_0)\,.$$

To get the next approximate value, $y(t_2) \approx y_2$, we simply use a "second" tangent line originating at the point (t_1, y_1) in the slope field. Continuing to take repeated steps by h on the t-interval, we get the recursive formula

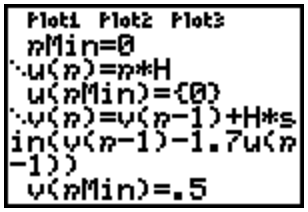
$$y(t_n) \approx y_n = y_{n-1} + h\,f(t_{n-1}, y_{n-1}), \quad n = 1, \cdots, N\,.$$

It is now common to study similar recursively defined sequences in many high school texts. In the Core Plus Mathematics Project (from Western Michigan University), this is called a "now-next" process. Such sequences can be evaluated in the home screen of a TI graphing calculator. Consider $dy/dt = \sin(y - 1.7t)$, $y(0) = 0.5$, $t_f = 5$, $N = 20$.

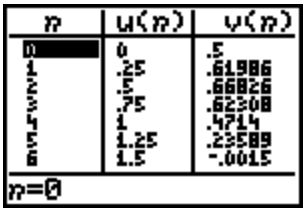


Once you get started (with the computation that gives y_2 on the second screen above, you simply need to press ENTER repeatedly to get the next approximations. You need to write these down as you compute them, and you need to keep count to know which term in the sequence is the next one.

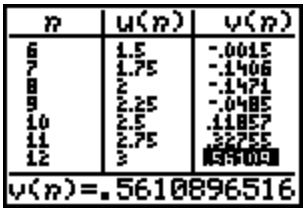
A better way is to use the sequence graphing mode on the TI-83. We use one sequence **u** to represent the t-values we desire and a second sequence **v** to represent the computed y-approximations.

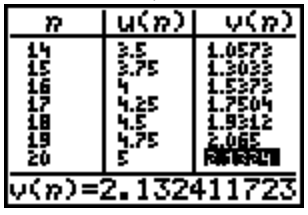


Y= Editor, H=0.25

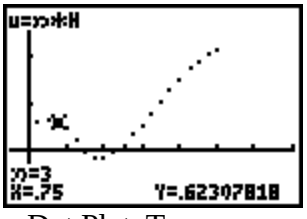


Table

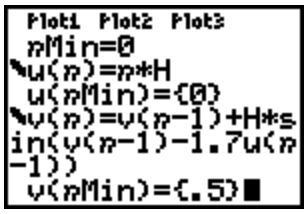




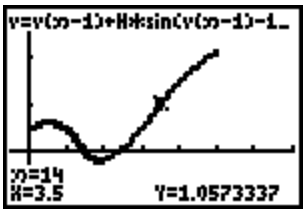
Graph FORMAT



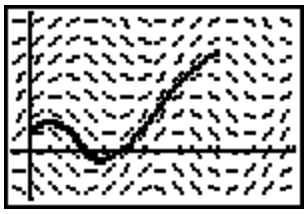
Dot Plot, Trace, **u**



Thick Style



Thick Plot, Trace, **v**


Thick Plot, prgmSLOPE

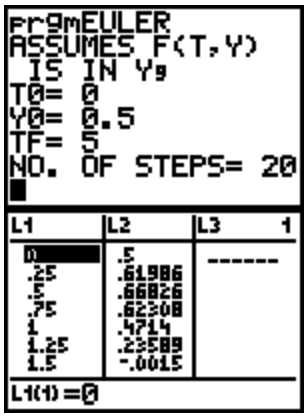
Tip: If you try to plot a single sequence later with the Graph FORMAT still set to **uv** plot, the error message: INVALID might not be enough information to tell you to change back to a **Time** plot.

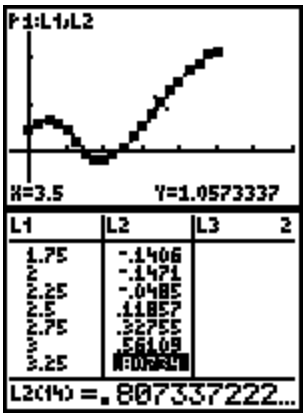
A third way to compute the Euler approximations is to do the computations in a program. If we store the results in lists, we can get a nice plot by turning on a statistical plot. This has two advantages over the sequence graphing. One is that we can stay in FUNCTION graphing mode, where we can also plot an exact solution (if the differential equation has one that we wish to compare to the numerical solution). A second advantage is that we can to do error analysis with the lists (again assuming an exact solution).

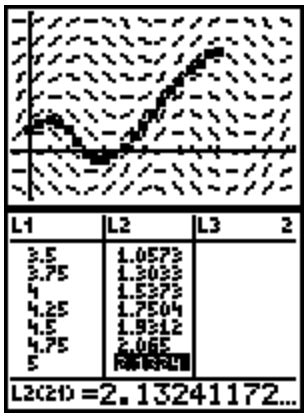
The following TI-83 program stores the *t*-values in list $\mathbf{L}_1$ and the corresponding *y*-values in $\mathbf{L}_2$. Then it turns on a statistical plot to display the graph of the computations. After running this you can again run prgmSLOPE to have the "DRAWN" slope lines added to the plot. (Since the line segments are drawn objects, they disappear whenever something changes in the plot.)

TI-83 prgmEULER

`Disp "ASSUMES F(T,Y)"`	`T→L1(1):Y→L2(1)`
`Input "T0= ",T`	`For(I,1,N,1)`
`Disp " IS IN Y9"`	`Y9→M`
`Input "Y0= ",Y`	`Y+H*M→Y:T+H→T`
`Input "TF= ",F`	`T→L1(I+1):Y→L2(I+1)`
`Input "NO. OF STEPS= ",N`	`End`
`(F-T)/N→H`	`Plot1(xyLine,L1,L2,?)`
`N+1→dim(L1):N+1→dim(L2)`	`DispGraph`



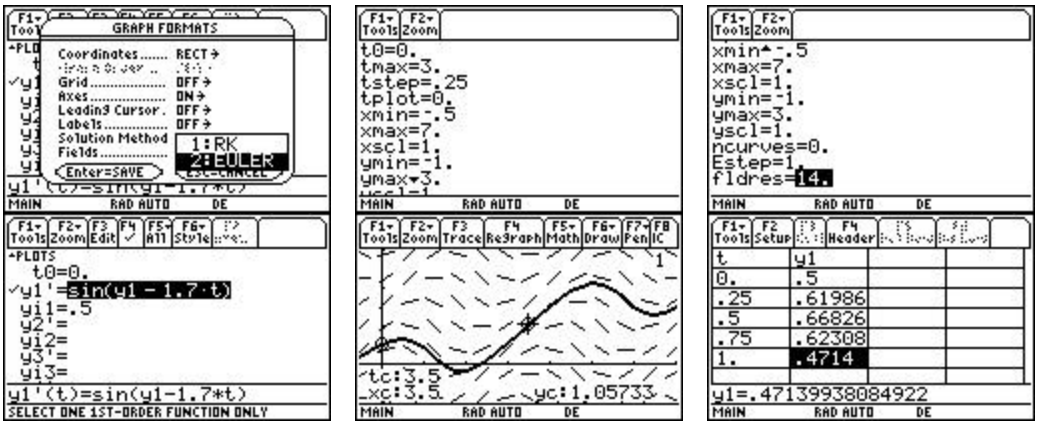




Long ago when these computations were done by hand, there would be a great deal of effort make to determine how small *h* needed to be to have some accuracy in the final approximation $y_N \approx y(t_f)$. Some calculus texts still place more emphasis on using the error estimates for doing this than is probably deserved. Given how easy it is to re-do the computations, we can simply try a few smaller values for *h* (or equivalently larger values for *N* with t_f fixed) to see the effects. More advanced methods than Euler's method incorporate internal computations within the algorithm to estimate the errors.

Euler's Method and Better Methods Built-In to Other TI Calculators

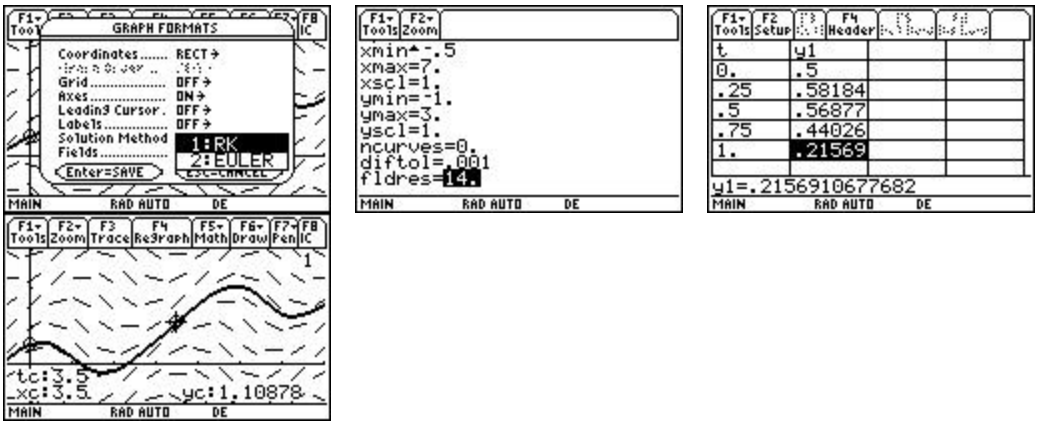
The TI-85 did not have Euler's method as an option. It simply did the numerical computations with a better method. Unfortunately Euler's method started to appear in calculus texts, and instructors demanded that it be an option in the TI-86, which it is. Certainly Euler's method is the simplest numerical method for solving a first order differential equation, making it nice for textbooks. Being so simple, it is not very accurate or efficient. To have any reasonable accuracy, you need to take hundreds (often thousands or millions) of steps. Thus when you select Euler's method in the formatting of differential equations on a TI-86, TI-89, or TI-92 PLus, the window editor then includes the setting **Estep** near the bottom. If you choose Estep=5, then the calculator will take five Euler steps before it plots again. For example, if you have 101 points plotted (from t_0 to t_f in increments of *tstep*), the calculator will actually do 500 Euler steps. While this improves accuracy a little (but takes a long time), far better is to choose the other solution method.



My advice is to select the **EULER** solution method only when the assignment specifies that Euler's method must be used. Then set Estep=1 to be able to record all of the steps for the assignment. When you trace, you will see all of the Euler steps computed. The window setting **tstep** is actually *h* if the assignment specifies that quantity in the method. If the assignment specifies the total number of steps *N*, then calculate **tstep** = *h* = (**tmax** – **tmin**)/*N* = $(t_f - t_0)/N$.

At all other times, select the method labeled **RK**. The letters stand for Runge-Kutta, and there are many methods of this type. The TI-89 Guidebook has more details in an

appendix if you wish to know more about the one implemented here. The **RK** solution method used is much better than Euler's method for several reasons. First, for any given *h* step, the calculator uses a more complicated procedure to find a more accurate approximation (using the ideas of both Runge and Kutta). Second, the algorithm also computes a local error estimate for this approximation during each step. If the local error estimate is too large (based upon the given tolerance), the algorithm will automatically back up to use a smaller *h* value. If the local error estimate is too small (based upon the tolerance), it will try a larger *h* on the next step. An algorithm that makes changes based upon computed local error estimates is called *adaptive*. Overall, it tries to give approximations that satisfy the tolerance you have given in the window editor as **diftol**. The adaptive RK algorithm will work harder (and take longer) with a smaller setting of diftol.



A final feature of this algorithm is the interpolation used to determine plotted points (or any other evaluations). If the calculator just plotted the computed points of the adaptive RK steps, the pattern of the points would be very irregular and unattractive (but accurate). Instead, the algorithm fits a cubic polynomial between these points in the viewing window (also matching the slopes at these points). It then evaluates the cubic at regular *tstep*-intervals. Set **diftol** to something large (like 0.5) and you can see the effect of this cubic interpolation. After a (large) **RK** step is computed by the algorithm, the grapher will plot several points (from the same cubic piece) in a hurry. In the **EULER** solution method, the plotted points will appear at regular time intervals because exactly the same computation takes place between plotted points.

References

I would, of course, highly recommend the TI Explorations book *Differential Equations with the TI-86*, by Ray Barton and Dennis Pence. The examples and activities there are also appropriate for users with a TI-89 or TI-92 Plus (but the commands will be slightly different to implement differential equation graphing). With the above TI-83 programs, you can also do most of the activities on a TI-83. For more general information about numerical mathematics (including Newton's method, trapezoid rule, and Simpson's rule, and Euler's method as well as more advanced methods), my favorite is *Numerical Mathematics and Computing*, by Ward Cheney and David Kincaid (Brooks/Cole Publishing Company). Cheney/Kincaid assume only knowledge of calculus.