University of Pennsylvania
EAS 105: Fall 2011
MATLAB PROJECT: Numerical Differentiation
Instructor: Donna Dietz

For this project, you will write a variety of two, three and five point routines which calculate derivatives. Your code will be tested for correctness against a sizeable bank of solutions, contained in a .mat file which will be provided. You will also be provided with a file called TestFunctionDiffList.m It is called using the command c=TestFunctionDiffList(); which results in a cellarray, c, being created. This has all your test problems in it. The cellarray c will have size 12, each cell having three elements. The first is the vector of t-values, the second is the vector of y-values, and the third (to be used for comparision purposes) is the 'actual' value of the derivative at that point. (Please note, that it is only the 'actual' value in the sense that this is the derivative of the function we started with. There is no way to conclusively find the 'actual' derivative based on a list of t-values and y-values! We will discuss this more in class.)

You will also be provided with Tester_NumDiff_key.mat.

The *Tester* will not work without these supplied files. Additonally, you will be creating plots with legends for each problem set, with each solution type having a different color.

If you don't know what a derivative is, check your class notes, or ask for help. The quickest way to visualize a derivative is to imagine a tiny car carrying a very long piece of lumber, strapped to the top of its roof. This tiny car goes up and down some steep hills. The derivative is the slope of that piece of lumber at any particular point on any particular hill.

For each of the eight NDiff functions, the input will be not only the y-values, but also the equally spaced t-values. This may seem unnecessary, but this is primarily for plotting purposes. The length of the vector of equally spaced t-values will be different than the input vector, hence, to avoid confusion, we will always keep these paired. We will use the plot command formats which expect both coordinates to be stated explicitly.

For this project, create these nine functions:

function [t_out, yp_out] = NDiff2A(t_in, y_in) For this function, the t_out will be shorter than the t_in by one, with the missing value being the *last* value of t_in. Let h be the common difference in the t-values. The derivative to be calculated at each t-value in t_out is

$$yp(i) = \frac{y(i+1) - y(i)}{h}.$$

However, you should not be using a loop to do this, as Matlab's native vector-handling abilites makes short work of this task!

function [t_out, yp_out] = NDiff2B(t_in, y_in) For this function, the t_out will be shorter than the t_in by one, with the missing value being the *first* value of t_in. Let h be the common difference in the t-values. The derivative to be calculated at each t-value in t_out is

$$yp(i) = \frac{y(i) - y(i-1)}{h}.$$

Again, you should not be using a loop to do this or any of these!

function [t_out, yp_out] = NDiff3A(t_in, y_in) For this function, the t_out will be shorter than the t_in by two, with the missing values being the *last two* values of t_in. Let h be the common difference in the t-values. The derivative to be calculated at each t-value in t_out is

$$yp(i) = \frac{-3y(i) + 4y(i+1) - y(i+2)}{2h}.$$

function [t_out, yp_out] = NDiff3B(t_in, y_in) For this function, the t_out will be shorter than the t_in by two, with the missing values being the *last two* values of t_in. Let h be the common difference in the t-values. The derivative to be calculated at each t-value in t_out is

$$yp(i) = \frac{y(i-2) - 4y(i-1) + 3y(i)}{2h}.$$

function [t_out, yp_out] = NDiff3C(t_in, y_in) For this function, the t_out will be shorter than the t_in by two, with the missing values being the *first and final* values of t_in. Let h be the common difference in the t-values. The derivative to be calculated at each t-value in t_out is

$$yp(i) = \frac{y(i+1) - y(i-1)}{2h}.$$

It is strange, isn't it, that this three-point formula involves only two points!? This is due to the way they are derived. The weight on the $y(i)$ component turns out to be zero! You should note that this looks like the definiation of the derivative, and that it looks just like NDiff2A averaged with NDiff2B!

Food for thought: Why do you think we are skipping the four-point formulas?!

function [t_out, yp_out] = NDiff5A(t_in, y_in) For this function, the t_out will be shorter than the t_in by four, with the missing values being the *last four* values of t_in. Let h be the common difference in the t-values. The derivative to be calculated at each t-value in t_out is

$$yp(i) = \frac{-25y(i) + 48y(i+1) - 36y(i+2) + 16y(i+3) - 3y(i+4)}{12h}.$$

function [t_out, yp_out] = NDiff5B(t_in, y_in) For this function, the t_out will be shorter than the t_in by four, with the missing values being the *first four* values of t_in. Let h be the common difference in the t-values. The derivative to be calculated at each t-value in t_out is

$$yp(i) = \frac{3y(i-4) - 16y(i-3) + 36y(i-2) - 48y(i-1) + 25y(i)}{12h}.$$

function [t_out, yp_out] = NDiff5C(t_in, y_in) For this function, the t_out will be shorter than the t_in by four, with the missing values being the *first two and last two* values of t_in. Let h be the common difference in the t-values. The derivative to be calculated at each t-value in t_out is

$$yp(i) = \frac{y(i-2) - 8y(i-1) + 8y(i+1) - y(i+2)}{12h}.$$

Did you notice the symmetry? Did you notice again the lack of $y(i)$ term? Does it make more sense this time that this formula has to be symmetric, and thus the middle term must drop out?

function PlotDiffTests(c) For each of the 12 provided test functions, for each of your 8 functions, you create a plot, plus you create one more for the 'actual' solution, and these will be bundled together by function. So, you will have 12 plots, each containing up to 9 functions. For the test functions in which the length of the t-vector is 5 or less, skip the three plots for the 5-point formulas, as they would be meaninless, and would probably crash your functions anyhow. To create a legend in your graph, you can use one of the following commands:
legend('2A','2B','3A','3B','3C','5A','5B','5C','Actual','Location','Best')
legend('2A','2B','3A','3B','3C','Actual','Location','Best')
The option shown here, 'Location', 'Best', is asking Matlab to place the legend where Matlab thinks the legend is least likely to interfere with the content of the plot. That works well for this project. You also need to put titles on your plots. You can use this command, title(['Example ' num2str(i)]); which is short and does the trick.(In this setting, 'i' is the counter.)

For this project, you will want to run the *Tester* early and often, to make sure you have everything just right!