

Math 131/135/194, Fall 2004

Applied Calculus

Profs. Kaplan & Flath

Macalester College

Lab 3: Derivatives on the Computer

Goal of this lab

To contrast the derivative of a function with the function itself and to compare the results of the symbolic derivative to that of the finite-difference approximation.

Software

We will use a special purpose R-stats function for this lab, called `D`. This function computes the finite-difference approximation to the derivative. Since the function was written by your diligent faculty mentors, it is not built in to the R system. You will need to instruct your R program to read in the function before you can use it.

To read in the `D` function, find the file `deriv.r` in the course folder.¹ Copy this file to your home directory in the same way that you copied the `whistle-scale.dat` file for Lab 2. Then use the FILE/CHANGE DIR menu command to move R to your home directory. Once you are there, you can give the simple command

```
> source('deriv.r')
```

Alternatively, you can use the FILE/SOURCE R CODE menu command and click on the `deriv.r` file in order to load it.

NOTE: Since the `deriv.r` file contains instructions to the R program, rather than data, you need to read it using a different command. The `source` program reads computer instructions into R, as opposed to the `scan` program that we used in Lab 2, which reads data into R.

The `D` function, like all functions, takes an input and produces an output. You are certainly used to the input to a function being a number. The `D` function is different. Like the mathematical derivative, the input is itself a function. This means that the input is something like `sin`, `exp`, and so on. The output of `D` is also a function. Like any other output in R, you can assign it to a variable in order to give the output a name.

Although you are very familiar with mathematical notation for numbers and variables, there is such a diversity of notations for functions that the situation can be confusing.

When we write $\sin(x)$ in mathematical notation, we are expressing the idea that the sine function takes a variable x as an input. The computer notation is subtly different. The computer statement `sin(x)` only makes sense when variable `x` has a specific numerical value. The statement means “apply the function `sin` to the values stored in the variable `x`.” This means that we should never write something like `D(sin(x))` since `sin(x)` is not a function; it is a number or set of numbers. The correct way to use `D` involves giving the function itself as an argument.

```
> newfun = D(sin)
```

We have saved the output of `D` in a variable called, in this example, `newfun`. The value of this variable is not a number, it is a function: the function that is the derivative of the sine function. We can evaluate this output function in the normal way, for instance

```
> newfun(0)
[1] 1
```

Note that there is no x here. Implicit in the use of `D` is that the function that it is being applied to takes only one argument; the derivative is with respect to that argument. If we want to take the derivative of a function that doesn't have a simple name, for instance $f(x) = 4x^2 - 3x + \cos(x/3)$, we would want first to define a function, as in

```
> f = function(x){ 4*x^2 -3*x + cos(x/3)}
> fderiv = D(f)
```

Mathematically, as you know, the derivative of a function $f(x)$ is defined as a limit

$$\lim_{\delta \rightarrow 0} \frac{f(x) - f(x - \delta)}{\delta}.$$

The algorithm used by `D` is very simple: rather than taking a mathematical limit, `D` merely sets δ to be a “small” number. The meaning of “small” is surprising subtle mathematically and took almost 200 years to work out rigorously *after* Newton had invented the calculus.

In this class, rather than focusing on the mathematically rigorous meaning of “small,” we're simply going to try various

¹Courses on Academic/Math194/anyone_readonly

different numerical values for “small,” and see what happens. To help us do this, we have arranged the `D` function to take a second, named argument, `delta`, that sets the numerical value of δ . For example:

```
> newfun = D(sin, delta=.1)
```

It's important to realize that taking the limit $\lim \delta \rightarrow 0$ cannot be accomplished numerically by setting `delta=0` in the function `D`. $\delta = 0$ is not a limit, but an undefined division by 0. If this is confusing to you, then you are in the same company as generations of mathematicians before and after Newton.

Multiple plots

Sometimes it's convenient to put multiple plots on the screen at the same time, so that you can compare them side-by-side. An easy way to do this in R-stats is to use the `layout` command. For example, to divide the plotting window into three sections, stacked vertically, give the command

```
layout( c(1,2,3) )
```

Each time you use the `plot` command, R-stats will move to the next section. The commands `lines` and `points` will add on to the current plot.

When you want to go back to a single plot, use

```
layout(1)
```

If you would prefer not to use `layout`, then you can always cut and paste the plots into a word processing program to line them up.

To hand in

1. For the function $\sin(x)$, compute analytically the derivative. Plot out, one above the other, the function and its derivative, over the domain $0 \leq x \leq 10$. Then, compute the finite-difference approximation to the derivative using `delta=.00001` and overlay this (using `lines`) on the plot of the analytical derivative. Just to remind you, if you have two functions, `f1` and `f2`, then to plot both on the same graph you first define the sequence of x points at which you want to evaluate the functions, then use `plot` and `lines` as follows:

```
> x = seq(from, to, length=1000)
> plot(x, f1(x))
> lines(x, f2(x))
```

where `from` and `to` stand for the lower and upper end of the x -axis plotting range.

2. Make a plot of the finite-difference approximation to the derivative of $\cos(x)$ for $-10 \leq x \leq 10$ for different values of `delta`. Find some value of `delta` which is too

large to give a reasonable approximation to the derivative. (You can define “reasonable” in terms of the visual appearance of the plot.)

3. Due to the way computers store and manipulate numbers, it is possible to set `delta` too small. Experiment with various values of `delta` to find one that is greater than zero but still too small for a good approximation. It might help to know that you can write numbers in scientific notation. For example, 3×10^8 would be written `3e8` and 1×10^{-10} is `1e-10`.
4. Compare the analytic derivative of $\exp(t)$ to the function itself and to the finite-difference approximation (for any reasonable value of `delta`.)
5. Compute the second derivative of $\sin(x)$ both analytically and using the computer `D` operator. Graph these versus x on the same axes. (Note: Because of the way the numerical approximation is taken, the approximation to the second derivative is not very good, but you may not notice this visually. If you try the third or fourth derivative, however, you will see a dramatic failure of the finite-difference approximation.)

Extra Credit

A Note on Computer Notation

When we write a mathematical function like $e^{\alpha x}$ or $A \sin(\frac{2\pi(t-d)}{p})$, we know from experience and convention which symbols stand for variables and which for parameters. But, seen abstractly, the function $e^{\alpha x}$ is a function of two variables, x and α . It's just that when we hold quantities like `alpha` constant we call them parameters.

The computer equivalent of $e^{\alpha x}$ might be written as `exp(alpha*x)`. As explained above, this expression really means “apply the exponential function to a single argument, the quantity `alpha*x`.”

How do we express to the computer that we want to define a function of x for some fixed value of α ? We have to give R-stats the command that says, “Create a function.” This is done as we have seen in the previous labs with the `function` command. When parameters are involved, we simply need to give a numerical value to the parameter and then define the function using the parameter. Two examples should suffice:

```
> alpha = .1;
> f = function(x) { exp( alpha*x ) }
```

Now `f` is the function of variable x . The quantity named `alpha` is a parameter. We can apply the function to an argument, that is, give the input variable x a value and find the output of function:

```
> f(10)
[1] 2.718282
```

If we change the value of the parameter `alpha`, then the behavior of `f` changes correspondingly.

```
> alpha = 0
> f(10)
[1] 1
```

Alternatively, if you have a fixed value of the parameter that you want to “lock in” to the function, you can give that value explicitly when you define the function:

```
> g = function(x){ exp( 3*x ) }
```

Once you have defined a function in this way, you can plot it, apply the D operator, or whatever, just like you would for a “built-in” function like `cos`.

A common error is to forget to assign a specific numerical value to the parameters. For example, here is an exponential function with two parameters:

```
> h = function(x){A*exp(k*x)}
```

It’s perfectly fine to take the derivative of this function using `D`:

```
> newfun = D(h)
```

But you won’t be able to apply the function `newfun` since the parameters are not defined. For example,

```
> newfun(3)
Error in h(3) : Object "A" not found
```

The error message is telling us that we haven’t defined the parameter `A`. Let’s do so:

```
> A = 100
> newfun(3)
Error in h(3) : Object "k" not found
```

Still an error! This is because we hadn’t yet defined `k`. Doing this allows us to use

```
> k = -3
> newfun(3)
[1] -0.03702294
```

Always look at error messages carefully for clues about what has gone wrong. They are often cryptic, but provide the best hint about how to fix things.

To Hand In for Extra Credit

- Define the function $f(x) = 3 + 2x - 4.2x^2$. Plot the function, its analytic derivative, and the finite-difference derivative found with the `D` operator.