How to Check Your Calculus Homework - Part 1

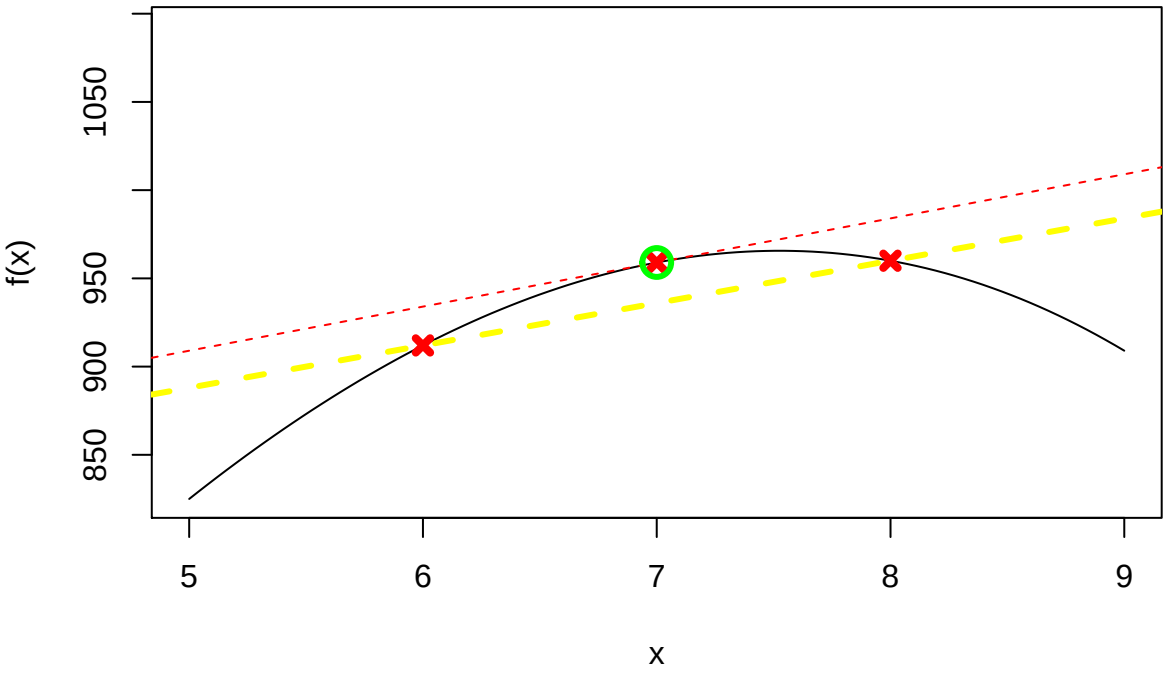
Robert Horton

August 31, 2014

Taking derivatives of functions is generally a matter of applying one or more formulas from a table of derivatives, such as those given in the back of a calculus textbook. This process can involve complicated algebra, so it is easy to make mistakes. Here we look at a simple way to check your work by graphing it against what the derivative should look like, based on a numerical approximation. If your analytical function for the derivative plots on top of the numerical approximation, that is good evidence that it is correct.

How it works

Say we have an equation for a curve, and we want to estimate the slope of this curve at a given point. Because we have the equation for the curve, we can use it as a function to find the position of the curve at any point. We can find the positions of two points, one slightly to the left of our point of interest, and another slightly to the right. If we draw a line through these two points, its slope will be an approximation of the slope of the tangent to the curve at the center point, assuming that the flanking points are close enough to the center point that the curve is reasonably smooth between them. This is shown in the figure below:



$$-x^3 - 2x^2 + 200x$$

Here the point where we want to estimate the slope is circled in green, and the flankig points are plain red "x"s. The red line is the tangent to the curve at the point of interest; this is what we are trying to estimate. The yellow line passes through the two flanking points. Note that it appears to parallel to the red tangent line quite closely. It is shifted below the red line, but we only care about the slope, so this shift doesn't matter.

This approximation only works if the flanking points are close together. If the flanking points are far apart and the curve has a complicated shape between these points, then the line between the flanking points might not reflect the slope of the tangent at the center point very well at all. But if the flanking points are close enough toghether that the curve between them is almost a straight line, the approximation will be very good.

Computing numerical derivatives

We can use the approach described above to create a function to compute numerical derivatives:

```
d <- function(FUN, X){
  Y <- FUN(X)
  numerical_derivative <- function(i) (Y[i+1]-Y[i-1])/(X[i+1]-X[i-1])
  interiorPoints <- 2:(length(X) - 1)
  c(NA, vapply(interiorPoints, numerical_derivative, numeric(1)), NA)
}
```

This function, named "d", takes two arguments: the function ("FUN") for which we want to estimate the derivative, and a vector of X values over which we will produce estimated slopes. The first step is to call the function on the X values; these are stored in the vector Y. Now we can find (x,y) points by looking in these vectors. Say we want the third (x,y) point; we set `i = 3`, then `X[i]` is the third x-value, and `Y[i]` is the third y-value; we can find the point before it with `i = 2` and the point after it with `i = 4`, repsectively. For any point `i`, we will estimate the slope of its tangent by rise over run of a line through its flanking points, or `(Y[i+1]-Y[i-1])/(X[i+1]-X[i-1])`; we will use the local function `numerical_derivative` to apply that formula to a set of `i` values. There is one minor issue; the very first and the very last points do not have flanking points, so we can only apply our slope-estimating function to the interior points; this is just a sequence of i-values from 2 to one less than the total number of points. Now when we apply our slope function to the indexes of the interior points (using `vapply`), we get a vector of slope values. To make plotting simpler, we add back "not available" values to both ends of this slope vector; this way it will be the same length as the original vector of x values, and this will keep the plot function from complaining that it can't match up the (x,y) pairs for our slope values.

Plotting functions

Here is some R code to plot a given function, the approximation of its derivative, and the user's guess of the derivative function on a single graph. You can use it to check whether your derivative function is close to the true answer.

```
plotFdF <- function(x, fun, dfun){
  ylim <- range(c(fun(x), d(fun,x), dfun(x)), na.rm=TRUE)
  plot(x, fun(x), type="l", ylim=ylim)  # original function for which we want derivative
  lines(x, d(fun,x), col="yellow", lwd=4)  # numerical approximation
  lines(x, dfun(x), col="red", lty=2)  # candidate analytical derivative
  abline(h=0, lty=3, col="gray")  # gray line showing zero on y-axis
}
```

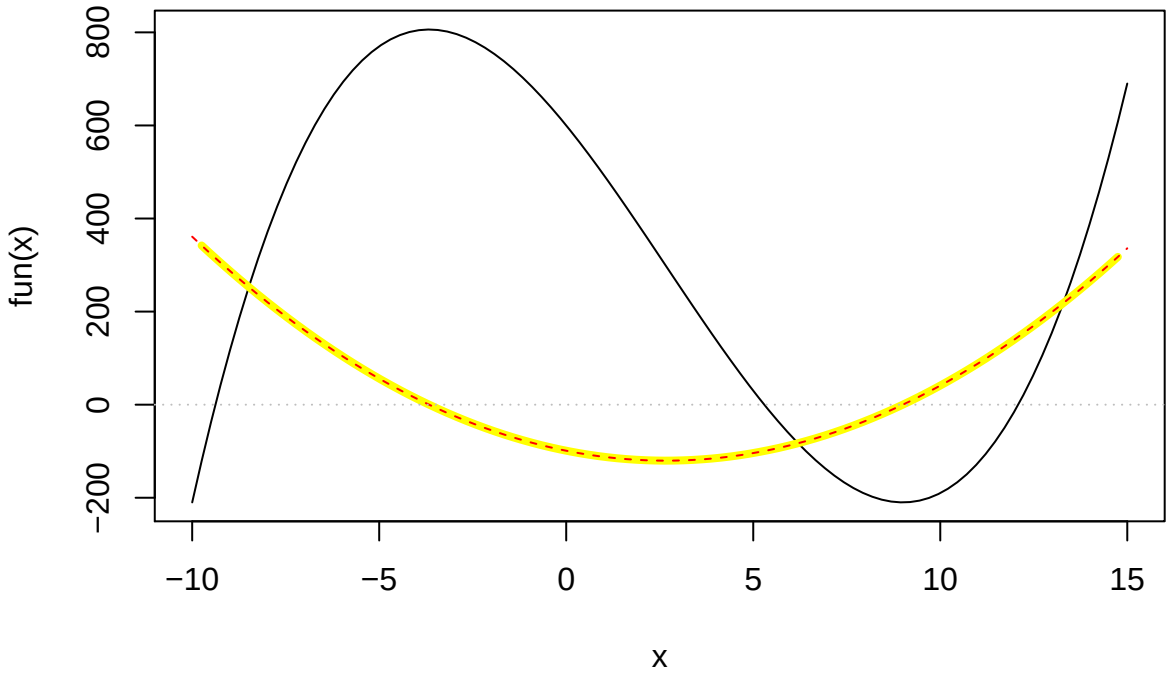
A simple example

Say we're given the following cubic polynomial function:

$$f(x) = -x^3 - 2x^2 + 200x$$

We'll code that equation as function `f`, and then use our calculus skills to write it's derivatove as function `df`. We provide some reasonable range of x-values to operate over, then call our plotting code. Note that when we make the sequence of x-values, we need to make them close enough together that our approximation will be reasonable; this spacing is specified in the "`by`"" parameter.

```
f <- function(x) x^3 - 8*x^2 - 99*x + 600
df <- function(x) 3*x^2 - 16*x - 99
x <- seq(-10, 15, length=100)
plotFdF(x, f, df)
```



A more complicated example

Say we need to take the derivative of a ratio of functions:

$$f(x) = (2sin(x) - sin(2x))/(x - sin(x))$$

Here we apply the quotient rule: if

$$f(x) = g(x)/h(x)$$

then

$$f'(x) = (g'(x) * h(x) - g(x) * h'(x))/(h(x))^2$$

Set

$$g(x) = (2sin(x) - sin(2x))$$

and

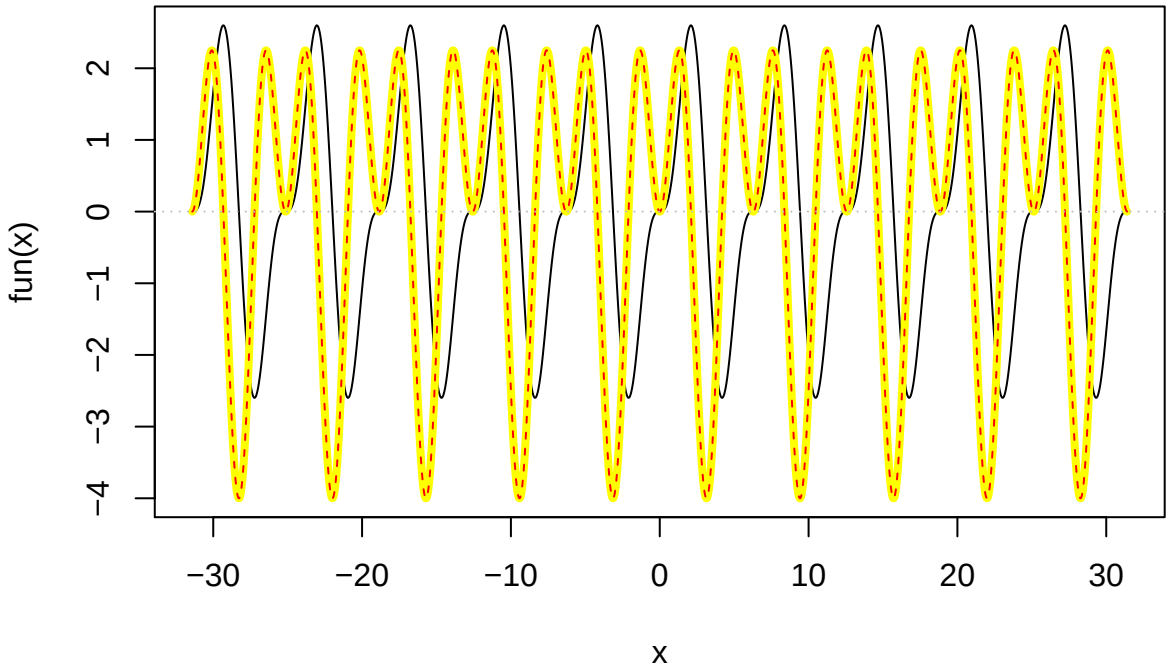
$$h(x) = (x - sin(x))$$

So we can break the problem down into three parts:

1. find the derivative of g(x)
2. find the derivative of h(x)
3. use these together with the quotient rule to find the derivative of f(x)

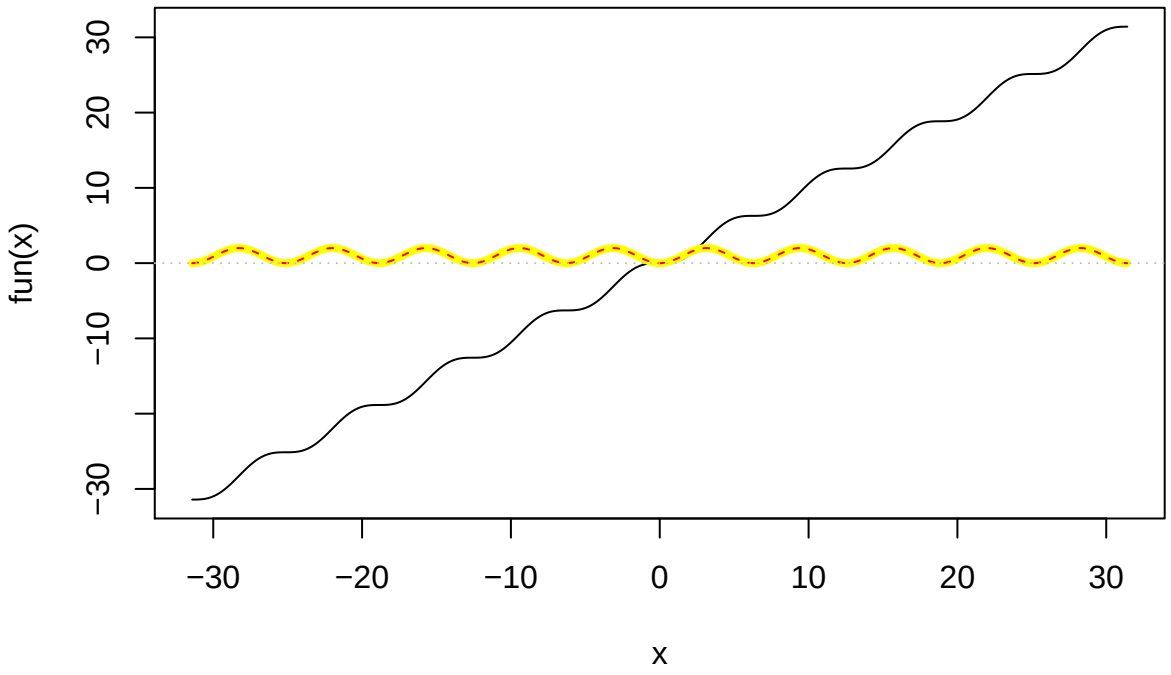
1. derivative of g(x)

```
x <- seq(-10 * pi, 10 * pi, , by=0.01)
g <- function(x) 2 * sin(x) - sin(2*x)
dg <- function(x) 2*(cos(x) - cos(2*x))
plotFdF(x, g, dg)
```



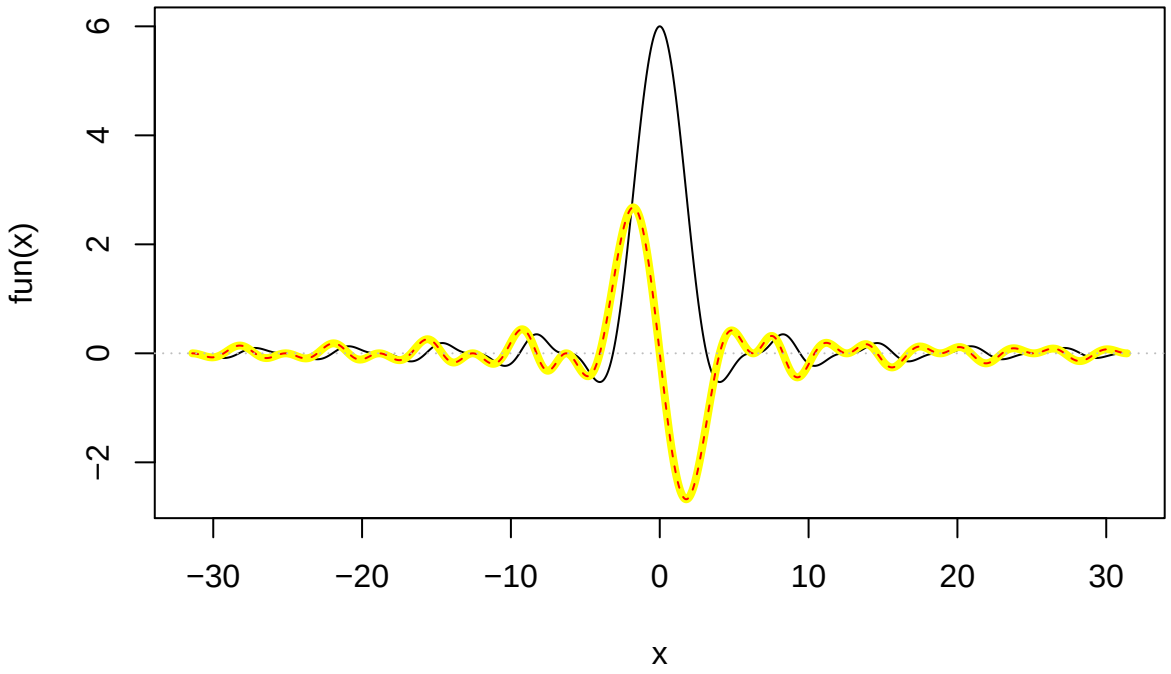
2. derivative of h(x)

```
# use the same x values as before
h <- function(x) x - sin(x)
dh <- function(x) 1 - cos(x)
plotFdF(x, h, dh)
```



3. Apply the quotient rule

```
f <- function(x) (2 * sin(x) - sin(2*x)) / (x - sin(x))
df1 = function(x) (dg(x)*h(x) - g(x)*dh(x)) / (h(x))^2
df2 = function(x) ((2*(cos(x) - cos(2*x)) * (x - sin(x))) - (2 * sin(x) - sin(2*x)) * (1 - cos(x)) )/ (
plotFdF(x, f, df2)  # df1 and df2 plot right on top of one another; try it!
```



$$f'(x) = ((2 * (cos(x) - cos(2 * x)) * (x - sin(x))) - (2 * sin(x) - sin(2 * x)) * (1 - cos(x)))/(x - sin(x))^2$$