**Numerical differentiation**

The need to differentiate a function in statistics is very common. Numerical differentiation can be used to make the process easier. The purpose of this section is to examine the basics of numerical differentiation.

Simple methods

You first learn in a calculus class that the derivative of a function f(x) with respect to x is



The numerical derivative then can be taken using



for some small number h > 0. Using this expression, we are trying to find the slope of a secant line (blue) to approximate the slope of the line of interest (red):



Ideally, we would like to find the slope of a tangent line (green) at f(x). It is obvious that the secant line slope is not necessarily equal to the tangent line slope. One could also work with



instead and perhaps have similar inaccuracies.

Alternatively, we can take use the central difference formula:



to find a secant line through the points (x-h, f(x-h)) and (x+h, f(x+h)). This accounts for the slope of f(x) with both a positive or negative h close to 0, and it is more accurate. The amount of error left from this alternative approach can be seen by first performing a Taylor series expansion of f(x) about some point x0:



If we take x to be x0 + h or x0 – h and find the difference f(x0 + h) – f(x0 – h), we have



 Dividing by 2h leads to



The resulting error with approximating  by  is approximately .

Example: Simple function (derivative\_examples.R)

Suppose the function of interest is

f(x) = x3 – 5

for x = 1 to 4.5. The first derivative of f(x) is



Below is how this can be found in R:

> fx <- function(x, a = 5) {

 x^3 - a

 }

> curve(expr = fx(x), xlim = c(1, 4.5), col = "red", lwd =

 2, ylim = c(-10, 100), xlab = "x", ylab = "f(x)")

> abline(h = 0, lty = "dotted")



> fx.dot <- function(x) {

 3\*x^2

 }

> x0 <- seq(from = 1, to = 4.5, by = 0.5)

> h <- 0.0001

> data.frame(true.der = fx.dot(x0),

 app.der1 = (fx(x0 + h) - fx(x0))/h,

 app.der2 = (fx(x0 + h) - fx(x0 - h))/(2\*h))

 x0 true.der app.der1 app.der2

1 1.0 3.00 3.00030 3.00

2 1.5 6.75 6.75045 6.75

3 2.0 12.00 12.00060 12.00

4 2.5 18.75 18.75075 18.75

5 3.0 27.00 27.00090 27.00

6 3.5 36.75 36.75105 36.75

7 4.0 48.00 48.00120 48.00

8 4.5 60.75 60.75135 60.75

Notice that the approximate error from the central difference formula is



This error is small enough that R indicates the calculations are exactly correct.

A smaller h does not necessarily result in more accuracy:

> h <- 10^(-7)

> data.frame(true.der = fx.dot(x0),

 app.der1 = (fx(x0 + h) - fx(x0))/h,

 app.der2 = (fx(x0 + h) - fx(x0 - h))/(2\*h))

 x0 true.der app.der1 app.der2

1 1.0 3.00 3.00 3.00

2 1.5 6.75 6.75 6.75

3 2.0 12.00 12.00 12.00

4 2.5 18.75 18.75 18.75

5 3.0 27.00 27.00 27.00

6 3.5 36.75 36.75 36.75

7 4.0 48.00 48.00 48.00

8 4.5 60.75 60.75 60.75

> h <- 10^(-14)

> data.frame(true.der = fx.dot(x0),

 app.der1 = (fx(x0 + h) - fx(x0))/h,

 app.der2 = (fx(x0 + h) - fx(x0 - h))/(2\*h))

 x0 true.der app.der1 app.der2

1 1.0 3.00 3.019807 3.019807

2 1.5 6.75 6.794565 6.750156

3 2.0 12.00 12.256862 12.123635

4 2.5 18.75 19.184654 19.095836

5 3.0 27.00 27.711167 27.533531

6 3.5 36.75 37.658765 37.658765

7 4.0 48.00 46.895821 47.961635

8 4.5 60.75 59.685590 59.685590

> h <- 10^(-16)

> data.frame(true.der = fx.dot(x0),

 app.der1 = (fx(x0 + h) - fx(x0))/h,

 app.der2 = (fx(x0 + h) - fx(x0 - h))/(2\*h))

 x0 true.der app.der1 app.der2

1 1.0 3.00 0 0

2 1.5 6.75 0 0

3 2.0 12.00 0 0

4 2.5 18.75 0 0

5 3.0 27.00 0 0

6 3.5 36.75 0 0

7 4.0 48.00 0 0

8 4.5 60.75 0 0

The reduction in accuracy occurs due to a computer’s ability to perform numerical calculations. There are only so many digits that a computer can track before it rounds. When x = 1 to 4.5 and h = 10-16, we see that the computer treats x0 + h is equal to x0 – h.

pracma package

This package is named for “PRACtical numerical Math functions.” It contains a number of functions for differentiation. In particular, fderiv() finds derivatives up to the 8th order using the central difference formula.

Example: Simple function (derivative\_examples.R)

Below is how I use fderiv():

> library(pracma)

> save1 <- fderiv(f = fx, x = x0, n = 1, h = 0.0001, method

 = "central") #Same as my central difference

> save2 <- fderiv(f = fx, x = x0, n = 1, h = 10^(-14),

 method = "central") #Same as my central difference

> save3 <- fderiv(f = fx, x = x0, n = 1) #Default - central

 difference and finds h on its own

 (.Machine$double.eps^(1/(n + 2)))

> data.frame(x0, save1, save2, save3)

 x0 save1 save2 save3

1 1.0 3.00 3.019807 3.00

2 1.5 6.75 6.750156 6.75

3 2.0 12.00 12.123635 12.00

4 2.5 18.75 19.095836 18.75

5 3.0 27.00 27.533531 27.00

6 3.5 36.75 37.658765 36.75

7 4.0 48.00 47.961635 48.00

8 4.5 60.75 59.685590 60.75

> #Higher order derivatives

> der2 <- fderiv(f = fx, x = x0, n = 2) #6x

> der3 <- fderiv(f = fx, x = x0, n = 3) #6

> der4 <- fderiv(f = fx, x = x0, n = 4) #0

> data.frame(x0, der2, der3, der4)

 x0 der2 der3 der4

1 1.0 6 6.000000 9.688727e-05

2 1.5 9 6.000000 0.000000e+00

3 2.0 12 6.000002 -4.602145e-04

4 2.5 15 5.999997 1.453309e-04

5 3.0 18 5.999997 5.813236e-04

6 3.5 21 5.999984 -1.937745e-04

7 4.0 24 6.000063 3.100393e-03

8 4.5 27 6.000080 7.750982e-04

The package also contains a function that performs an iterative Richardson’s extrapolation method in an attempt to compute a more accurate derivative. Details on this method are available at <http://dafeda.wordpress.com/2010/10/18/numerical-differentiation-richardsons-extrapolation/>.

> numderiv(f = fx, x0 = 1)

$df

[1] 3

$rel.err

[1] 0

$niter

[1] 3

> numderiv(f = fx, x0 = x0) #Does not work - the help makes

 me think it should work for a vector

Error in numderiv(f = fx, x0 = x0) :

 Argument 'x0' must be a numeric scalar.

Example: Bernoulli distribution (Bernoulli.R)

The purpose here is to find the maximum likelihood estimate (MLE) of the probability of success parameter from a Bernoulli distribution. Obviously, the MLE can be simply derived, but this provides us a simple example where we can compare the actually MLE to a numerically estimated MLE.

Suppose w = 4 and n = 10. Define  Note that



We can perform the function maximization by first finding the partial derivative of (π|w) with respect to π:



set this equal to 0, and solve for π. The MLE of π is w/n.

Below are some calculations performed in R:

> w <- 4

> n1 <- 10 #Using n1 rather than n to avoid confusion with

 n argument in fderiv()

> logL <- function(pi, w, n1) {

 w \* log(pi) + (n1 - w) \* log(1 - pi)

 }

> curve(expr = logL(pi = x, w = w, n1 = n1), xlim = c(0,1),

 xlab = expression(pi), ylab = "log(L)", col = "red")

> abline(v = w/n1, lty = "dotted")



> library(pracma)

> partial.logL <- function(pi, w, n1) {

 w/pi - (n1 - w)/(1 - pi)

 }

> curve(expr = partial.logL(pi = x, w = w, n1 = n1), xlim =

 c(0,1), xlab = expression(pi), ylab = "Partial

 derivative of log(L)", col = "red", lwd = 2)

> abline(v = w/n1, lty = "dotted")

> abline(h = 0, lty = "dotdash")

> partial.logL2 <- function(pi, w, n1) {

 fderiv(f = logL, x = pi, n = 1, w = 4, n1 = 10)

 }

> partial.logL2(pi = 0.4, w = 4, n1 = 10) #At MLE

[1] 3.666853e-10

> partial.logL2(pi = c(0.4,0.5), w = 4, n1 = 10)

[1] 3.666853e-10 -4.000000e+00

> # Confirms numerical derivative is same

> curve(expr = partial.logL2(pi = x, w = w, n1 = n1), col =

 "blue", add = TRUE, xlim = c(0.05, 0.95), lty =

 "dashed", lwd = 2)



> uniroot(f = partial.logL2, interval = c(0.01,0.99), w =

 4, n1 = 10)

$root

[1] 0.4

$f.root

[1] -9.827166e-09

$iter

[1] 2

$estim.prec

[1] 6.103516e-05

The uniroot() function is a simple function that finds the root of an equation. In the optimization section of this course, we will examine functions that will combine the derivative and solving for roots process for us.

There are a number of other functions that can be useful for finding partial derivatives and higher order derivatives. Below is a simple example involving the gamma distribution. We will discuss this example further in the optimization section.

Example: Gamma distribution (MLE\_gamma.R)

Using the Casella and Berger (2002) parameterization, below are the likelihood and log-likelihood functions:





> set.seed(4869)

> n <- 30

> alpha <- 7/2

> beta <- 2 #Chisq(7)

> y <- rgamma(n = n, shape = alpha, scale = beta)

> round(y, 2)

 [1] 3.61 10.08 1.90 3.75 10.43 5.52 9.86 1.84 12.53

[10] 6.09 12.40 3.31 16.48 6.34 6.65 10.11 9.18 7.98

[19] 1.84 2.14 3.65 10.34 10.88 14.31 5.82 10.70 1.58

[28] 13.48 5.37 11.31

> logL <- function(theta, y) {

 n <- length(y)

 alpha <- theta[1]

 beta <- theta[2]

 sum(dgamma(x = y, shape = alpha, scale = beta, log =

 TRUE))

 }

> library(pracma)

> #At parameter values

> grad(f = logL, x0 = c(alpha, beta), y = y)

[1] 1.168529 4.876178

> #At MLEs – found later in the course

> grad(f = logL, x0 = c(2.661508, 2.874372), y = y)

[1] -2.717343e-05 -2.557997e-05

> obs.Fisher.info <- hessian.mat <- hessian(f = logL, x0 =

 c(2.661508, 2.874372), y = y)

> -solve(obs.Fisher.info) #Estimated covariance matrix

 [,1] [,2]

[1,] 0.4208934 -0.4545567

[2,] -0.4545567 0.5943881

Symbolic

R can perform some symbolic differentiations. The two main functions are D() and deriv(). Below is part of the shared help file for these functions:



Do not expect that these functions will work for all possible functions!

Example: Simple function (derivative\_examples.R)

We used the function f(x) = x3 – 5 for this example earlier. The D() function below will return the symbolic derivative to the R Console window. The deriv() function will return a function that can be used to evaluate the derivative at specific values. Notice how I used the expression() function a few times to make sure that R recognizes that I am not wanting to evaluate a particular set of code at the values previously defined for some objects.

> fx <- function(x, a = 5) {

 x^3 - a

 }

> D(expr = expression(x^3 - 5), "x")

3 \* x^2

> deriv(expr = ~ fx, namevec = "x")

expression({

 .value <- fx

 .grad <- array(0, c(length(.value), 1L), list(NULL,

 c("x")))

 .grad[, "x"] <- 0

 attr(.value, "gradient") <- .grad

 .value

})

> deriv(expr = ~ x^3 - 5, namevec = "x")

expression({

 .value <- x^3 - 5

 .grad <- array(0, c(length(.value), 1L), list(NULL,

 c("x")))

 .grad[, "x"] <- 3 \* x^2

 attr(.value, "gradient") <- .grad

 .value

})

> save1 <- deriv(expr = ~ x^3 - 5, namevec = "x")

> save2 <- deriv(expr = ~ x^3 - 5, namevec = "x", hessian =

 TRUE)

> x <- seq(from = 1, to = 4.5, by = 0.5)

> eval(save1)

[1] -4.000 -1.625 3.000 10.625 22.000 37.875 59.000 86.125

attr(,"gradient")

 x

[1,] 3.00

[2,] 6.75

[3,] 12.00

[4,] 18.75

[5,] 27.00

[6,] 36.75

[7,] 48.00

[8,] 60.75

> eval(save2)

[1] -4.000 -1.625 3.000 10.625 22.000 37.875 59.000 86.125

attr(,"gradient")

 x

[1,] 3.00

[2,] 6.75

[3,] 12.00

[4,] 18.75

[5,] 27.00

[6,] 36.75

[7,] 48.00

[8,] 60.75

attr(,"hessian")

, , x

 x

[1,] 6

[2,] 9

[3,] 12

[4,] 15

[5,] 18

[6,] 21

[7,] 24

[8,] 27

The Hessian is just a vector because there is only one variable for which a derivative is found.

Suppose f(x,y) = x3 – 5 + y2. Below are the results from deriv():

> save3 <- deriv(expr = ~ x^3 - 5 + y^2, namevec = c("x",

 "y"), hessian = TRUE)

> x <- 1

> y <- 1

> eval(save3)

[1] -3

attr(,"gradient")

 x y

[1,] 3 2

attr(,"hessian")

, , x

 x y

[1,] 6 0

, , y

 x y

[1,] 0 2

The first row of the Hessian matrix is given in the , , x part of the hessian attribute and the second row of the Hessian matrix is given in the , , y part of the attribute.

My tool of choice for symbolic differentiation completely outside of R is Maple. Below is a simple example:

> fx:=x^3 - 5;



> diff(fx, x);



I wish that I would have known about these types of software packages when I was taking courses like STAT 882 and STAT 883!

Final notes on numerical differentiation

* diff() does not find a derivative. Rather it finds lagged differences, such as xt – xt-1 at time t.
* The numDeriv package contains functions similar to those in pracma for differentiation.