ADR Introduction

Johannes Willkomm

March 10, 2020

This is an introduction to the ADR system for **A**utomatic **D**ifferentiation of **R** code, a method to evaluate derivatives of numerical functions written in R.

Consider a function in R like `myFunc`:

```
myFunc <- function(x, y) { print('.'); 2 * x * y }
```

Run the function, first creating variables for the actual parameters and then invoking the function with them:

```
a <- 12
b <- 4
z <- myFunc(a, b)

## [1] "."

z

## [1] 96
```

You can get the derivatives of `z` with respect to (w.r.t.) `a` and `b` with ADR by loading the package "adr" and then using the function `adrDiffFor`:

```
library('adr')
J <- adrDiffFor(myFunc, arguments = list(a, b))

## ADR: Post HTTR request for "r-ad-fm" to 'https://r-adr.de/adr/',
45 chars
## ADR: HTTR response after 0.658 s, 1139 chars

## [1] "."

J[[1]]

##      [,1]
## [1,]    8
```

This call returns the so called "Jacobian", which is the matrix of the first order derivatives of all values in the first argument `a` w.r.t. all return values, in this case a single scalar. Note that `df/dx=2*y` is 8, so the result is correct. The derivative is evaluated by the following steps behind the scenes:

1. The source code of `myFunc` is obtained and sent to the ADR server for differentiation.
2. The differentiated R source code returned by the ADR server is parsed and evaluated, which yields an "augmented function"
3. The augmented function is run to actually evaluate the derivatives

The first two steps are only done the first time you differentiate `myFunc`, the results are cached.

Continuing the example, ADR also works for vector functions. This means the R function code remains the same, we just change the argument, the point where to evaluate `myFunc`:

```
a <- c(1, 3, 12)
J <- adrDiffFor(myFunc, arguments = list(a, b))

## [1] "."
## [1] "."
## [1] "."

J[[1]]

##      [,1] [,2] [,3]
## [1,]    8    0    0
## [2,]    0    8    0
## [3,]    0    0    8
```

The differentiated function code also remains the same. However, when the definition of `myFunc` is changed, it obviously will have to be differentiated again:

```
myFunc <- function(x, y) { print('.'); 3 * x * y }
J <- adrDiffFor(myFunc, arguments = list(a, b))

## ADR: Post HTTR request for "r-ad-fm" to 'https://r-adr.de/adr/',
45 chars
## ADR: HTTR response after 0.511 s, 1139 chars

## [1] "."
## [1] "."
## [1] "."

J
```

```
## $J
##      [,1] [,2] [,3]
## [1,]   12    0    0
## [2,]    0   12    0
## [3,]    0    0   12
##
## $f
## [1]  12  36 144
```

Note how the message indicating the sending of the code to the ADR server appeared again this time. You can look at the differentiated source code by using the ADR function `d`:

```
d_myFunc <- d(myFunc, c(1))
d_myFunc

## function (f.p1, f.p0)
## {
##     x = f.p0[[1]]
##     y = f.p0[[2]]
##     dv_x = f.p1[[1]]
##     print(".")
##     d_f.ca2 = 3 * dv_x
##     f.ca2 = 3 * x
##     d_f.ca1 = d_f.ca2 * y
##     f.ca1 = f.ca2 * y
##     d_f.ridm16 <- d_f.ca1
##     f.ridm16 <- f.ca1
##     list(df = d_f.ridm16, f = f.ridm16)
## }
```

The function `d_myFunc` can perfectly well be run manually, for example:

```
d_a <- array(c(1,0,0), c(3,1))
d_myFunc(list(d_a), list(a, b))

## [1] "."
## $df
##      [,1]
## [1,]   12
## [2,]    0
## [3,]    0
##
## $f
## [1]  12  36 144
```

This yields the first column of `J`. The function `adrDiffFor` is basically just a driver which runs a series of calls like the one above and arranges the results in the Jacobian matrix. Finally, the argument `seed` can be set to a so called "seed matrix", for example:

```
S <- array(c(1,1,0, 0,0,1), c(3,2))
JtimesS <- adrDiffFor(myFunc, seed = S, arguments = list(a, b))

## [1] "."
## [1] "."

JtimesS[[1]]

##      [,1] [,2]
## [1,]   12    0
## [2,]   12    0
## [3,]    0   12
```

The result `JtimesS` is the Jacobian multiplied by the seed matrix, as can easily be checked:

```
J[[1]] %*% S

##      [,1] [,2]
## [1,]   12    0
## [2,]   12    0
## [3,]    0   12
```

It can be much faster and efficient to to give `S` to `adrDiffFor` than to multiply a-posteriorily. Coming up with and using a suitable seed matrix is possibly the most important thing to consider when using ADR or automatic differentiation in general. The time and memory needed to run `adrDiffFor` depends linearly on the number of columns in `S`, the so called "number of directional derivatives". This is because `adrDiffFor` will have to run the differentiated code once for each column in `S`, setting the derivative argument `d_a` to each column in turn. In particular, a matrix-vector product of the Jacobian can be obtained with a single run of the differentiated function.

The so-called vector mode is automatically activated when the number of directional derivatives exceeds the value in option `vectormode-switchover`. Then the differentiated function is run just once, but using the class `advec` to propagate bundles of derivatives simultaneously. With option `vectormode` the vectormode can be set manually.

```
S <- array(c(1,0,0, 0,1,0, 0,0,1), c(3,3))
options=adrOptions(vectormode=TRUE)
JtimesS <- adrDiffFor(myFunc, list(a, b), seed=S, options=options)
```

```
## [1] "."

JtimesS[[1]]

##      [,1] [,2] [,3]
## [1,]   12    0    0
## [2,]    0   12    0
## [3,]    0    0   12
```

The server URL is obtained from the function `adrServerURL()` which defaults to the environment variable `ADR_SERVER`, then to the option `adrserver` and finally to the static value

```
## [1] "https://r-adr.de/"
```