HW3: writing a Newton optimizer

This homework is extremely difficult to complete without having watched lecture 13 (as well as 11 and 12), and attempted the week 6 exercises.

Programming Task: to write an R function, `newton`, implementing Newton's method for minimization of functions, and to provide example code using it to optimize Rosenbrock's function. Due 17:00 13/11/20.

Specification: Your `newton` optimization function should operate broadly in the same way as `nlm`. Note that the purpose is to have an independent implementation: you must code the optimization yourself, not simply call optimization code written by someone else. The function arguments should be as follows:

```
newton(theta,f,...,tol=1e-8,fscale=1,maxit=100,max.half=20)
```

`theta` is a vector of initial values for the optimization parameters.

`f` is the objective function to minimize. Its first argument is the vector of optimization parameters. Remaining arguments will be passed from `newton` using '`...`'. The scalar value returned by `f` will have 2 attributes, a `gradient` vector and, optionally, a `hessian` matrix.

`...` any arguments of `f` after the first (the parameter vector) are passed using this (see hints below).

`tol` the convergence tolerance.

`fscale` a rough estimate of the magnitude of f at the optimum - used in convergence testing.

`maxit` the maximum number of Newton iterations to try before giving up.

`max.half` the maximum number of times a step should be halved before concluding that the step has failed to improve the objective.

`newton` should return a list containing:

`f` the value of the objective function at the minimum.

`theta` the value of the parameters at the minimum.

`iter` the number of iterations taken to reach the minimum.

`g` the gradient vector at the minimum (so the user can judge closeness to numerical zero).

`Hi` the inverse of the Hessian matrix at the minimum (useful if the objective is a negative log likelihood).

The function should issue errors or warnings (using `stop` or `warning` as appropriate) in at least the following cases. 1. If the objective or derivatives are not finite at the initial `theta`; 2. If the step fails to reduce the objective despite trying `max.half` step halvings; 3. If `maxit` is reached without convergence; 4. If the Hessian is not positive definite at convergence.

Other considerations:

1. You can test whether your Hessian is positive definite, by seeing if `chol` succeeds in finding its Cholesky factor. Read the documentation for `try` to find out how to trap the error generated by `chol` if it fails. If the Hessian is not positive definite, add a small multiple of the identity matrix to it and try again. One approach is to start by adding $\epsilon\mathbf{I}$ where ϵ is the largest absolute value in your Hessian, multiplied by 10^{-8}. If the perturbed Hessian is still not positive definite, keep multiplying ϵ by 10 until it is.

2. If your Newton step does not reduce the objective, or leads to a non-finite objective or derivatives, you will need to repeatedly half the step until the objective is reduced.

3. To judge whether the gradient vector is close enough to zero, you will need to consider the magnitude of the objective (you can't expect gradients to to be down at 10^{-10} if the objective is of order 10^{10}, for example). So the gradients are judged to be zero when they are smaller than `tol` multiplied by the objective. But then there is a problem it the objective is zero at the minimum - we can then never succeed in making the magnitude of the gradient less than the magnitude of the objective. So `fscale` is provided. Then if `f0` is the current value of the objective and `g` the current gradient,

```
max(abs(g)) < (abs(f0)+fscale)*tol
```

is a suitable condition for convergence.

4. If no Hessian matrix is supplied, your code should generate one by finite differencing the gradient vector. Such an approximate Hessian will be asymmetric: `H <- 0.5 * (t(H) + H)` fixes that.

Hints:

1. Recall from the lectures that Rosenbrock's function is

   ```
   { k*(z-x^2)^2 + (1-x)^2}
   ```

   and that you can use `deriv` to get the required gradient and Hessian. Note however that if you use `deriv` you will need to write a wrapper function to call the function it creates. This is necessary because the objective function expected by `newton` expects the optimization parameters (z and x, here) to be in a single parameter vector. Also the gradient and Hessian returned by the function `deriv` creates are not returned as a vector and matrix, so you will need to modify them slightly. $z = -.5$, $x = 2$ are reasonable starting values to use as illustrative examples.

2. The following code gives a simple example of passing named arguments 'through' a function using the '`...`' argument. This illustrates how '`...`' can be used to pass the named arguments to `f`, through `newton`, without `newton` having to 'know' what those arguments are.

   ```
   printxt <- function(n,txt1,txt2) { cat(n,txt1,txt2,"\n") }
   foo <- function(a=1,...) { printxt(10*a,...) }
   foo(3,txt1="pizzas",txt2="please!")
   ```

3. It is a good idea to write the code to deal with the case in which gradient and Hessian are available first. Then modify it to add the code to deal with the case in which the Hessian is not available.

4. It is a good idea to test your code on more than just Rosenbrock's function.

5. In addition to the R Functions already mentioned, some of the following may be useful: `abs`, `attr`, `backsolve`, `chol`, `diag`, `forwardsolve`, `inherits`, `is.finite`, `is.null`, `length`, `matrix`, `max`, `ncol`, `nrow`.

6. We will look at your comments when marking. Good comments explain the purpose of each function and clarify what the code is supposed to do, so that someone reading the code can rapidly understand its purpose and why it is written as it is. Comments that simply describe mechanically what each line of code does are useless. For example

   ```
   f <- A %*% y ## multiplies y by A to get f
   ```

   simply describes what is obvious from the code itself and is useless, whereas

   ```
   f <- A %*% y ## compute the fitted values, f, from the response, y
   ```

   tells the reader the purpose of the code.

What to submit: Within the `github` repository for your group you should create a single text file `newton.R` containing the implementation of your `newton` function. A second file `rosenbrock.R` should contain the code illustrating use of `newton` to optimize the Rosenbrock function, with and without the exact Hessian supplied. `rosenbrock.R` should source `newton.R`, and sourcing `rosenbrock.R` in R studio should simply cause both examples to run. What is submitted should not contain other testing code, or code not required by `newton` or the 2 examples. Other files in your repository will not be marked - you might want to put any such files in a subfolder of the repository to keep things tidy.

Marking: Marks will be awarded for the supplied code running properly, and for it behaving correctly on some further test objective functions that you will not be given. Marks will also be awarded for code structure and commenting. Highest marks will be awarded for concise, efficient, well structured, well commented code behaving according to the specification.