Loss function demonstration

Peter Chi

April 16, 2020

The following is the output without adjusting `rel.tol`:

```
optim.out
```

```
##                p1        p2        p3        p4        p5        p6         p7
## nlminb -0.8086033 -3.141873 -1.088816 -5.273553 -2.622657 -4.583462 -0.6572162
##                p8         p9        value fevals gevals niter convcode kkt1 kkt2
## nlminb -2.149927 -0.4993164 1.274033e-06     99    686    73        1 TRUE TRUE
##        xtime
## nlminb  21.5
```

The value of 1 under `convcode` is what indicates that convergence was not attained.

Originally, what would happen now is that the optimization would be re-run with new, random start values. Like the following:

```
optim.out.r
```

```
##                p1        p2       p3        p4 p5        p6         p7
## nlminb -1.401877 -1.416569 1.999496 -1.097514  2 -9.088163 -0.6377611
##                p8        p9        value fevals gevals niter convcode kkt1
## nlminb -0.9641644 -4.332943 1.000321e-06     61    300    26        1 TRUE
##         kkt2 xtime
## nlminb FALSE  9.03
```

In this particular case, `p5` hit the max of 2. The loss function (under `value`) here is lower than that of the first run, and `convcode` is equal to 1. But in other scenarios that I investigated, neither of these were necessarily true.

Regardless, after adjusting `rel.tol=1e-4`, we obtain:

```
optim.out.1e4
```

```
##                p1        p2        p3        p4        p5        p6         p7
## nlminb -0.8086033 -3.141873 -1.088816 -5.273553 -2.622657 -4.583462 -0.6572162
##                p8         p9        value fevals gevals niter convcode kkt1 kkt2
## nlminb -2.149927 -0.4993164 1.274033e-06     72    639    69        0 TRUE TRUE
##        xtime
## nlminb 17.61
```

All of the values are the as that of the original run. The only difference is that now `convcode=0` so it is happy to stay here.

Now, plugging the MLE values into the loss function gives:

```
new.ls.loss(log(ml.tree$edge.length[c(7,8,5,6,9,3,1,2,4)]), nodist.tree,
            read.phylosim.nuc(as.character(my.align)))
```

```
## [1] 2.82606e-06
```

So, about twice the size of what we get through optimization, but still very small.