Assignment: High performance R with Rcpp

EDH7916

Benjamin Skinner

Using `collegeloc.RDS` and `cblocks.RDS`, please answer the following question. Use comments to respond as necessary.

Questions

1. Below is a function that computes the great circle distance using Vincenty's formula[1]. It's more accurate than the haversine version, but can be much more computationally intensive. Try to convert the base R function into an Rcpp function. You'll need to start a new script and then use `sourceCpp()` to read it in and test. Once you've got, substitute the respective Vincenty formula functions into the `dist_min_*()` functions and compare times.

   A few things to keep in mind:

   1. You'll need to declare your variables and types (lot's of `double`); don't forget that double numbers need a decimal, otherwise C++ thinks they are integers.

   2. Don't forget your semi-colon line endings!

   3. `abs()` in C++ is `fabs()`

   4. Remember: `a^2 = a * a`

```r
## base R distance function using Vincenty formula
dist_vincenty <- function(xlon,
                          xlat,
                          ylon,
                          ylat) {

  ## return 0 if same point
  if (xlon == ylon && xlat == ylat) { return(0) }
  ## convert degrees to radians
  xlon <- deg_to_rad(xlon)
  xlat <- deg_to_rad(xlat)
  ylon <- deg_to_rad(ylon)
  ylat <- deg_to_rad(ylat)
  ## ------------------------------------------------
  ## https:##en.wikipedia.org/wiki/Vincenty%27s_formulae
  ## ------------------------------------------------
  ## some constants
  a <- 6378137
  f <- 1 / 298.257223563
  b <- (1 - f) * a
```

[1] https://en.wikipedia.org/wiki/Vincenty%27s_formulae

```
U1 <- atan((1 - f) * tan(xlat))
U2 <- atan((1 - f) * tan(ylat))
sinU1 <- sin(U1)
sinU2 <- sin(U2)
cosU1 <- cos(U1)
cosU2 <- cos(U2)
L <- ylon - xlon
lambda <- L                          # initial value
## set up loop
iters <- 100                         # no more than 100 loops
tol <- 1.0e-12                       # tolerance level
again <- TRUE
## while loop...
while (again) {
  ## sin sigma
  sinLambda <- sin(lambda)
  cosLambda <- cos(lambda)
  p1 <- cosU2 * sinLambda
  p2 <- cosU1 * sinU2 - sinU1 * cosU2 * cosLambda
  sinsig <- sqrt(p1^2 + p2^2)
  ## cos sigma
  cossig <- sinU1 * sinU2 + cosU1 * cosU2 * cosLambda
  ## plain ol' sigma
  sigma <- atan2(sinsig, cossig)
  ## sin alpha
  sina <- cosU1 * cosU2 * sinLambda / sinsig
  ## cos^2 alpha
  cos2a <- 1 - (sina * sina)
  ## cos 2*sig_m
  cos2sigm <- cossig - 2 * sinU1 * sinU2 / cos2a
  ## C
  C <- f / 16 * cos2a * (4 + f * (4 - 3 * cos2a))
  ## store old lambda
  lambdaOld <- lambda
  ## update lambda
  lambda <- L + (1 - C) * f * sina *
    (sigma + C * sinsig * (cos2sigm + C * cossig *
                           (-1 + 2 * cos2sigm^2)))
  ## subtract from iteration total
  iters <- iters - 1
  ## go again if lambda diff is > tolerance and still have iterations
  again <- (abs(lambda - lambdaOld) > tol && iters > 0)
  }
  ## if iteration count runs out, stop and send message
  if (iters == 0) {
    stop("Failed to converge!", call. = FALSE)
  }
  else {
    ## u^2
    Usq <- cos2a * (a^2 - b^2) / (b^2)
    ## A
    A <- 1 + Usq / 16384 * (4096 + Usq * (-768 + Usq * (320 - 175 * Usq)))
    ## B
```

```
    B <- Usq / 1024 * (256 + Usq * (-128 + Usq * (74 - 47 * Usq)))
    ## delta sigma
    dsigma <- B * sinsig *
      (cos2sigm + B / 4 *
         (cossig * (-1 + 2 * cos2sigm^2)
           - B / 6 * cos2sigm * (-3 + 4 * sinsig^2)
           * (-3 + 4 * cos2sigm^2)))
    ## return the distance
    return(b * A * (sigma - dsigma))
  }
}
```

Submission details

- Save your script (`<lastname>_assignment_rcpp.R`) in your `scripts` directory.
- Push changes to your repo (the new script and new folder) to GitHub prior to the next class session.