radiant.R

mcint

2021-09-27

```r
# Converts dates to an easily comparably float
numeric.date <- function(date, format = "%Y%m%d", tz = "UTC") {
  date <- strptime(date, format, tz = tz)
  date <- strftime(date, format, tz = tz)
  date <- as.numeric(date)
  return(date)
}

# Returns all events that fall between the start date and end date. If no end
#  date is provided, return events that occured on the start.date.
find.events <- function(events,
                        start.date,
                        end.date = NULL,
                        format = "%Y%m%d",
                        tz = "UTC") {
  # Sanity check
  if (length(start.date) != 1) {
    stop("Invalid length of start.date in find.events ")
  }
  else if (!is.null(end.date) && length(end.date) != 1) {
    stop("Invalid length of end.date in find.eventss ")
  }
  else if (length(start.date == 1) && is.null(end.date)){
    end.date = start.date
  }

  start.date <- numeric.date(start.date, format = format, tz = tz)
  end.date <- numeric.date(end.date, format = format, tz = tz)
  event.dates <- as.numeric(event.calendar.date(events))

  return(events[(event.dates >= start.date) &
                (event.dates <= end.date)])
}

# From a list of showers, return that with the given name and year
find.shower <- function(name, year, showers) {
  return(showers[(((name == showers$name) |
                     (name == showers$abbrev) |
                     name == showers$number
  )
  & year == substr(showers$start.date, 1, 4)), ])
}
```

```r
#Returns the cross product of two 3 dimensional vectors
cross.product.3d <- function(vector1, vector2) {
  if ((length(vector1) != 3) || (length(vector2) != 3)) {
    stop("radiant - cross.product.3d: Invalid dimensions (D[3] X D[3])")
  }

  i = vector1[[2]] * vector2[[3]] - vector1[[3]] * vector2[[2]]
  j = vector1[[3]] * vector2[[1]] - vector1[[1]] * vector2[[3]]
  k = vector1[[1]] * vector2[[2]] - vector1[[2]] * vector2[[1]]

  return(c(i, j, k))
}

#Converts from equatorial coordinates to Cartesian coordinates
equatorial.to.cartesian <- function(ra, dec = NULL) {
  #If the data is being passed as a table, break it into two vectors
  if ((length(dec) == 0) && (length(ra) == 2)) {
    dec <- sapply(ra, function(ra) {
      return(ra[2])
    })
    ra  <- sapply(ra, function(ra) {
      return(ra[1])
    })
  }
  #If the ha is provided, break it off
  else if (!is.null(names(ra)) &&
           all(names(ra) == c("ra", "dec", "ha"))) {
    dec <- ra$dec
    ha <- ra$ha
    ra <- ra$ra
  }
  #Null case
  else if ((length(ra) == 0) && (length(dec)) == 0) {
    return()
  }
  else if (length(ra) != length(dec)) {
    stop("Invalid length of ra or dec. Haulting execution.")
  }

  r <- sapply(1:length(ra), function(i, ra, dec) {
    x = cos(dec[[i]] * pi / 180) * cos(ra[[i]] * pi / 180)
    y = cos(dec[[i]] * pi / 180) * sin(ra[[i]] * pi / 180)
    z = sin(dec[[i]] * pi / 180)
    return(c(x, y, z))
  }, ra, dec)

  return(r)
}

# Returns the radiant given 2 events of the form (ra, dec, antirad), where
#  antirad = 0 means true radiant, antirad = 1 means antiradiant, and
#  antirad = -1 is an indication that one of these events is an outlier
```

```r
radiant <- function(event1, event2) {
  #Get Cartesian coordinates for the start and end points of the events
  r1.start <-
    equatorial.to.cartesian(event.start.equatorial(event1))
  r1.end   <- equatorial.to.cartesian(event.end.equatorial(event1))

  r2.start <-
    equatorial.to.cartesian(event.start.equatorial(event2))
  r2.end   <- equatorial.to.cartesian(event.end.equatorial(event2))

  #Crossing the start point with the end point gives a norm for the plane
  #  in which the movement exists
  n1 <- cross.product.3d(r1.start, r1.end)
  n2 <- cross.product.3d(r2.start, r2.end)

  #Crossing the norm of the resulting plane above yields the vector of the
  #  line intersecting the two plance, which points towards (or directly away
  #  from) the point of intersection
  rad <- cross.product.3d(n1, n2)
  rad <- rad / norm(rad)


  #Check if the antiradiant was found. Test if the result is
  #  closer to the start of the events (thus its source) or the end of the
  #   events (thus its termination).
  if ((norm(r1.start - rad) > norm(r1.end - rad)) &&
      (norm(r2.start - rad) > norm(r2.end - rad))) {
    #confirmed antiradiant
    rad <- -rad
    antirad <- 1
  } else if ((norm(r1.start - rad) > norm(r1.end - rad)) ||
             (norm(r2.start - rad) > norm(r2.end - rad))) {
    #An error flag, this result doesn't correlate to anything that would
    #  be expected to happen if both events were from a meteor shower.
    antirad <- -1
  } else {
    antirad <- 0 #confirmed radiant
  }

  x <- rad[[1]]
  y <- rad[[2]]
  z <- rad[[3]]

  #Convert cartesian coordinates to equatorial
  rho <- norm(c(x, y))
  ra <- (acos(x / rho) * 180 / pi) %% 360

  if (y < 0) {
    ra <- 360 - ra
  }

  rho <- norm(rad)
  dec <- asin(z / rho) * 180 / pi
```

```r
  return(data.frame(
    ra = ra,
    dec = dec,
    antirad = antirad
  ))
}

# Generates the radiants for all intersections of the given events
shower.radiants <- function(events, verbose = TRUE) {
  n.events <- length(events)
  if (n.events < 2) {
    # Cannot even be considered a shower with under 2 events
    print("Too few events passed to shower.radiants")
    return(data.frame(
      ra = NULL,
      dec = NULL,
      antirad = NULL,
      event1 = NULL,
      event2 = NULL
    ))
  }

  # Go over every event (except the last) and check the radiant with every
  #   event that comes after (including the last)
  radiants <- lapply(events[-n.events], function(event1) {
    next.event <- match(event.name(event1), event.name(events)) + 1
    if (verbose) {
      cat(next.event - 1, "/", n.events - 1, " events ")
    }
    radiant <- lapply(events[next.event:n.events], function(event2) {
      rad <- radiant(event1, event2)
      rad$event1 <- event.name(event1)
      rad$event2 <- event.name(event2)
      if (verbose) {
        cat(".")
      }
      return(rad)
    })
    if (verbose) {
      cat("X\n")
    }
    return(radiant)
  })

  radiants <- bind_rows(radiants)

  return(radiants)
}

# Uses a Fisher distribution fit (vmf.mle) to determine the mean direction of
#  a shower's events' radiants. Returns a lot of info. Option to recalculate
#  radiant data if the formula has changed, along with the option to remove
#  remove outliers based on the given aggression.
```

```r
mean.radiant <- function(events,
                         shower = NULL,
                         recalc.radiants = FALSE,
                         aggression = 0.0,
                         verbose = TRUE){
  if (is.null(shower)){
    shower.events <- events
    recalc.radiants <- TRUE
  } else{
    shower.events <- find.events(events, shower$peak.date)
  }
  n.events <- length(shower.events)
  if (n.events < 2){
    return(data.frame(n.events = n.events, n.radiants = 0))
  }
  if(!recalc.radiants && file.exists(paste("./save-files/radiants/",
                                          shower$abbrev,
                                          substr(shower$start.date, 1, 4),
                                          ".txt",
                                          sep=""))){
    radiants <- load.radiant(shower)
    n.events <- 0.5 * (sqrt(8 * nrow(radiants) + 1) - 1)
  } else{
    radiants <- shower.radiants(shower.events)
  }
  n.radiants <- nrow(radiants)
  if (n.radiants < 2){
    return(data.frame(n.events = n.events, n.radiants = n.radiants))
  }

  r <- equatorial.to.cartesian(radiants$ra, radiants$dec)
  r <- t(r)

  if(aggression > 0.0){
    bad.events <- suspect.events(radiants, aggression = aggression)
    n.bad.events <- length(bad.events)
    radiants <- remove.event(radiants, bad.events)
    n.bad.radiants <- n.radiants - nrow(radiants)
    n.radiants <- nrow(radiants)
    if (n.radiants < 2){
      return(data.frame(n.events = n.events,
                        n.radiants = n.radiants,
                        n.bad.events = n.bad.events,
                        n.bad.radiants = n.bad.radiants))
    }
  } else{
    n.bad.events <- -1
    n.bad.radiants <- -1
  }

  r <- equatorial.to.cartesian(radiants$ra, radiants$dec)
  r <- t(r)
```

```
  fishkent.p.value <- fishkent(r, B = 1)[2]
  kent <- kent.mle(r)
  vmf <- vmf.mle(r)

  # mu represents the mean direction of the radiants
  x <- vmf$mu[1]
  y <- vmf$mu[2]
  z <- vmf$mu[3]


  #Convert cartesian coordinates to equatorial
  rho <- norm(c(x, y))
  mean.ra <- (acos(x / rho) * 180 / pi) %% 360

  if(y < 0){
    mean.ra <- 360 - mean.ra
  }

  rho <- norm(mean)
  mean.dec <- asin(z / rho) * 180 / pi
  #Convert cartesian coordinates to equatorial
  rho <- norm(c(x, y))
  mean.ra <- (acos(x / rho) * 180 / pi) %% 360

  if(y < 0){
    mean.ra <- 360 - mean.ra
  }

  kent.mu <- kent$G[,1]
  sdom <- sqrt(vmf$kappa ^ -1) / sqrt(n.events - n.bad.events) * 180 / pi

  return(data.frame(mean.ra = mean.ra,
                    mean.dec = mean.dec,
                    sdom = sdom,
                    n.events = n.events,
                    n.radiants = n.radiants,
                    n.bad.events = n.bad.events,
                    n.bad.radiants = n.bad.radiants,
                    fishkent.p.value = fishkent.p.value,
                    vmf.mu.x = vmf$mu[1],
                    vmf.mu.y = vmf$mu[2],
                    vmf.mu.z = vmf$mu[3],
                    vmf.kappa = vmf$kappa,
                    kent.mu.x = kent.mu[1],
                    kent.mu.y = kent.mu[2],
                    kent.mu.z = kent.mu[3],
                    kent.kappa = kent$param[1]))
}
```