Working with EDEN data using `fireHydro`

Version 0.3.01

Troy Hill; Troy_Hill@nps.gov

updated: 2022-05-03

1. Introduction

This vignette demonstrates how to use R to access spatially-interpolated water surface data collected and maintained by the Everglades Depth Estimation Network (EDEN). The `fireHydro` R package includes functions that interact with EDEN's databases and provide access to EDEN data from an R session. `fireHydro` is hosted on GitHub and can be installed from the R console using the remotes package, as shown below. `fireHydro` ver. 0.3 exclusively uses the `terra` and `sf` spatial packages and includes S3 methods for routine operations on `eden` class objects.

```
# if remotes isn't already installed, install it
if (!"remotes" %in% installed.packages()) {install.packages("remotes")}

# install fireHydro from GitHub
remotes::install_github("troyhill/fireHydro@master")
```

`fireHydro` only needs to be installed once, but scripts that rely on `fireHydro` functions need to load the package at the beginning of each session:

```
library(fireHydro)
```

EDEN water surfaces are available in a range of formats and on multiple sites. Daily layers from the most recent two quarters are hosted on EDEN's real-time data page. Data older than the past two quarters are available separately in quarterly datasets. This presents challenges for accessing data spanning broad date ranges; pulling data for individual dates in the last two quarters will be rapid, whereas pulling a single day of data from a year ago requires downloading an entire quarter of data. `fireHydro` users should be aware of this nuance and use it to their advantage in pulling data. When downloading data for a large date range, it can be more time-efficient to download data for a broader time period than needed and trim it down to the desired subset.

---

2. Preparation

Because EDEN data can be large, I recommend setting `terraOptions` to store temp files on your hard drive rather than in RAM (`todisk=TRUE`), and setting your temp folder to a location that you can easily find and delete manually if your hard drive starts filling up.

```
### explicitly setting your temp directory makes it easier to delete temp files
temp_folder <- tmpDir() # consider changing this

### my preferred terraOptions
terraOptions(memfrac  = 0.5,
             tempdir  = temp_folder,
             todisk   = TRUE,
             progress = 1)
```

3. Downloading EDEN data: single dates

Often a single date of EDEN data is sufficient for a task. If that date is recent (e.g., today's water depth layer) the task could hardly be simpler. The default behavior is to download the most recent EDEN data available if no date is provided. If a date is provided, it should either be a `Date` object or, if a character element, should be formatted as `%Y%m%d` or `%Y-%m-%d`. We'll specify that we want the returned data to be `terra` objects here and throughout the vignette, because that is much faster than the alternative, converting data to an `sf` class.

```
edenDat <- getEDEN(EDEN_date = Sys.Date(), returnType = "terra")
```

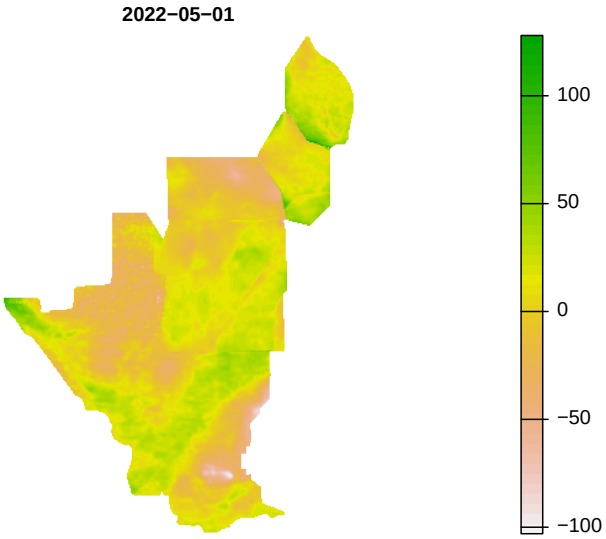
```
class(edenDat)
```

```
[1] "eden" "list"
```

```
names(edenDat)
```

```
[1] "date" "data"
```

```
plot.eden(edenDat) # provides a plot showing the first date and data layer
```



The object returned by `getEDEN` is a distinct class named `eden`. `eden` objects are lists with two elements: the water depth data, and the date(s) that those data correspond to. These are in elements names `data` and `date`, respectively. This structure can be useful because the date element provides important metadata about the data element that can simplify manipulations based on date ranges.

Let's also download the water depth layer from one week prior to the most recent data:

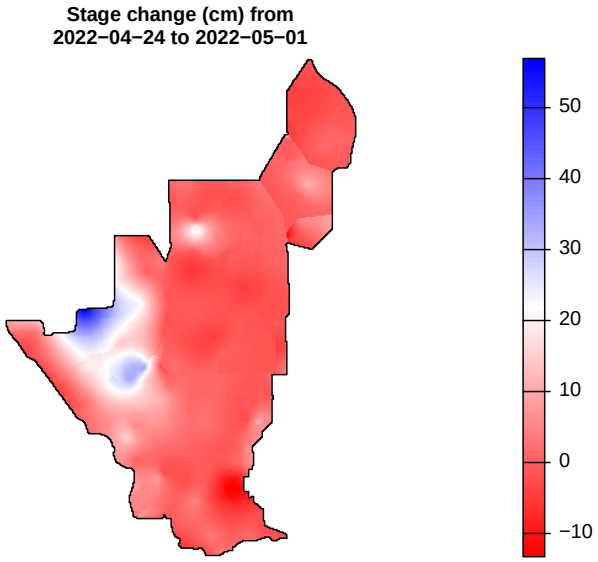
```
edenDat_1week <- getEDEN(EDEN_date  = edenDat$date - 7, returnType = "terra")
```

With these data we can easily visualize how stages across south Florida have changed during our one-week interval. Default units in the `getEDEN` output are water depths in centimeters with respect to the soil surface, so this plot will show stage changes in centimeters.

```
recessionRates <- edenDat$data - edenDat_1week$data
vals           <- values(recessionRates, na.rm = TRUE)
largestValue   <- round(max(abs(vals), na.rm = TRUE))
cuts           <- c((largestValue*-1):largestValue)
pal            <- colorRampPalette(c("red", "white","blue"))

### add an outline
eden_domain  <- as.polygons(classify(recessionRates, cbind(-Inf, Inf, 1)), dissolve=TRUE)


# positive values = ascension; negative = recession
plot(recessionRates, axes = FALSE,
     type = 'continuous', col = pal(length(cuts)), main =
       paste0('Stage change (cm) from\n', edenDat_1week$date, ' to ', edenDat$date))
plot(eden_domain, add = TRUE)
```



EDEN data are provided as files that have water surfaces expressed in centimeters NAVD88. `fireHydro` internally converts to water depths by subtracting the EDEN DEM. Water surfaces in cm NAVD88 can be returned by providing the argument `DEM = NULL`.

Converting between units and datums is a common task. `fireHydro` contains a function to quickly convert between water depths (centimeters relative to the soil surface) and stages in feet NGVD29. Note that these functions require the user to keep track of units - running the same conversion multiple times will lead to nonsensical results.

```
### convert to feet NGVD29
edenDat$data <- convertEDEN(EDENdata = edenDat$data, to = "ft_NGVD29")

### convert back to centimeters depth
edenDat$data <- convertEDEN(EDENdata = edenDat$data, to = "cm_soil")
```

3. Downloading EDEN data: multiple recent dates

Multiple days of EDEN data can be easily downloaded by passing a vector of dates to `getEDEN()`:

```
### make a vector of dates
dateSelect <- seq.Date(from = Sys.Date()-15, to = Sys.Date(), by = 1)

### download EDEN data
eden_list  <- getEDEN(EDEN_date = dateSelect, exact = TRUE, returnType = "terra")
```

Downloading individual days of data works nicely for recent data that are available as individual files. For data older than the past two quarters, obtaining a single day of data requires downloading an entire quarter of data, which naturally takes much longer. This means that the approach shown directly above, where a vector of dates was provided to `getEDEN()`, is extremely inefficient for data that are not available on EDEN's real-time data page. In the next section, we'll see how to download entire quarters or years of data, how to extract date ranges of interest and stitch together date ranges spanning multiple quarters.

4. Downloading EDEN data: Entire quarters and years

To download a full quarter of data, input a date in the desired quarter and set `quarterly = TRUE`:

```
q1_2021 <- getEDEN(EDEN_date = "20210115", quarterly = TRUE)
q2_2021 <- getEDEN(EDEN_date = "20210315", quarterly = TRUE)

### note that the date element includes all dates in the quarter
q1_2021$date
```

To download entire years of EDEN data there is a separate function that downloads and combines quarterly data.

```
eden2020 <- getAnnualEDEN(years = 2020)
```

5. Subsetting and merging EDEN lists

Subsetting and merging EDEN lists is simplified using provided functions. The `subset.eden` function accepts either an index or date range as the subsetting criterion.

```
# combine Q1 and Q2 data
H1_2021 <- merge(q1_2021, q2_2021)
```

A useful approach to subsetting is to identify elements that are in a target time period, and then subset both the `data` and `date` elements in the EDEN list.

```
# an example where we want all data after some date
startDate       <- "2021-03-01"
target_elements <- which(H1_2021$date > startDate)

currentYear_sub <- subset.eden(H1_2021, subset = target_elements)
```

6. Saving and loading EDEN objects

Instead of constantly downloading EDEN data needed for an analysis, `eden` objects can be saved and loaded from a local directory. These operations are still somewhat sub-optimal, partly because of complexities related to `SpatRaster` objects being a pointer to a file on disk, rather than the data itself. An approach that I've found useful is to have a dedicated folder on my hard drive that I exclusively use for storing annual EDEN data objects. I then load years of EDEN data and manipulate as needed.

I highly recommend having a dedicated folder because `write.eden` creates two files: a .tif file with the geospatial data and a .txt file with the same name holding the corresponding dates. `read.eden` looks for those two files and combines them to reproduce the `eden` object that was saved.

```r
### write data
write.eden(x = eden2020, filename = "2020.tif")

### read data
newEden <- read.eden(filename = "2020.tif")
```

7. A closing note

At this point the entirety of the EDEN data stream is available to you via the R command line. You have everything you need to manipulate and analyze one of the most expansive water level datasets in the world. As you use `fireHydro`, don't hesitate to let me know how you're using it, what you struggle with, and any ideas for how it might better serve the South Florida restoration science community. If you identify a problem with the code, please submit an issue on `fireHydro`'s GitHub page.

8. Going further

This final section includes some potentially useful, tangentially related code. This section assumes the code in previous sections has been run. Parts of this section highlight how `fireHydro` can be complemented by the GitHub packages 'SFNRC' and 'EvergladesEBM'. Those packages can be installed via:

```
remotes::install_github("troyhill/SFNRC@master")
remotes::install_github("troyhill/EvergladesEBM@main")

library(SFNRC)
library(EvergladesEBM)
```

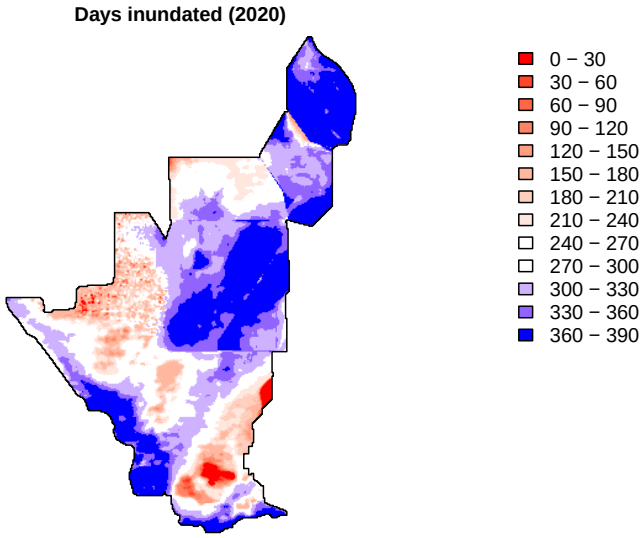
A. Aggregating and summarizing

After pulling a year of EDEN data, pixel-level summaries can be easily calculated using functions in the `terra` package. For example, hydroperiod can be calculated as the number of days each pixel was inundated:

```
daysWet <- app(eden2020$data, fun=function(x){sum(x > 0)})

cols <- getColors(uniqueValues = c(0, 366), binSize = 30,
                  colorGradient = c("red",'white', "blue"), # "#f5f5f5"
                  threshold = 270)

plot(daysWet, axes=FALSE, main = "Days inundated (2020)",
     col = cols$colors, breaks = cols$key)
plot(eden_domain, add = TRUE)
```

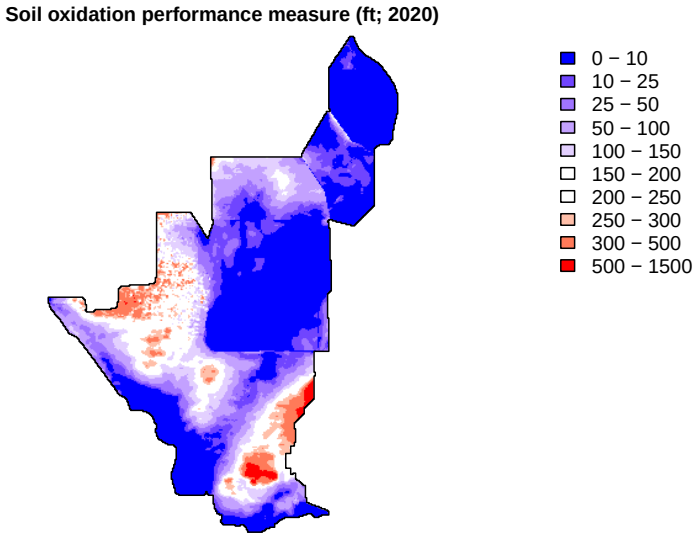


Other measures can be calculated by adjusting the function supplied to `terra::app`. Soil oxidation is a common performance measure with units of feet; when water is belowground, the depth accumulates on a daily time step.

```
soilOxidation <- app(eden2020$data, fun = function(x){sum(x[x < 0]) / 30.48})

cols <- getColors(uniqueValues = c(0, 10, 25, 50, 100, 150, 200, 250, 300, 500, 1500),
                  colorGradient = c("blue",'white', "red"), # "#f5f5f5"
                  threshold = 200, type = 'interval')

plot(soilOxidation*-1, axes = FALSE, box = FALSE,
     main = "Soil oxidation performance measure (ft; 2020)",
     col = cols$colors, breaks = cols$key)
plot(eden_domain, add = TRUE)
```

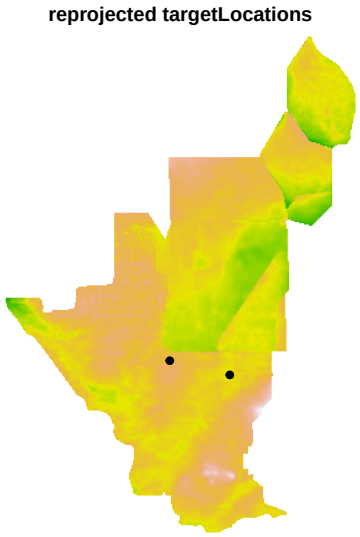


B. Extracting data at locations of interest

In situations where there are critical locations of interest, it can be helpful to extract and plot the time series for those locations. This can also reveal quality control issues. In the code block below we pull station coordinates from the EDEN website, make the coordinates spatial data, and then use them to extract EDEN data. The `extractEDEN` function is from the `EvergladesEBM` R package. If a polygon were used instead, the function would return the average daily value across all EDEN pixels in the polygon.

```
### pull some station coordinates from the EDEN site
stns <- c("SPARO", "NESRS1")
stn.coords <- do.call(rbind, lapply(X = stns, FUN = getCoords_EDEN))


stages2020 <- extractEDEN(targetLocations      = stn.coords,
                          targetLocationNames = stn.coords$stn,
                          EDEN_data           = eden2020)
```




```
Coordinate reference systems did not match. Conversion has been performed internally but may be incorrec
```

```
ggplot(stages2020, aes(y = ave, x = date, col = name)) + geom_line() +
  ylab("Water depth (cm)")
```