Practical 1 Part 1

(R version)

7 August 2017

Introduction

In this first practical you will be introduced to the following tools and concepts, and will begin exploring two different example data sets.

Learning objectives:

1. Gain confidence with using the tagtools to load, plot and process data.
2. Gain understanding of different data representations in the tagtools: vectors vs sensor structures, raw vs corrected.
3. Learn how to use the tools to quantify aspects of behaviour.
4. Learn how to compute and evaluate data quality metrics for 3d field sensors.
5. Learn how to derive orientation from acceleration and magnetometer.
6. Compute and visualize a dead-reckoned track while appreciating some of the sources of error.

Section 1. Working with pressure data and data structures

1.1 Introduction:

Pressure data (e.g., depth or altitude) are useful for identifying behavioural epochs (dives or flights) but can be strongly affected by temperature and, in the case of altimeters, by the ambient pressure. Sensors on deep diving animals can also suffer from poor resolution because of the large pressures they have to handle. Here we are going to use tools to read in some example data, examine it and correct temperature errors. We will then quantify dive duration in the corrected data.

1.2 What you will need

1. Tag tools installed on your machine.
2. Test data sets installed in your working directory (or a known file location)
3. You will be using test data set mn12_186a
4. You will be exploring the data using the following tag tools: load_nc, sens2var, plot, crop, median_filter, crop_to, fix_pressure, find_dives.

1.3 Load the test data set

Load the test dataset mn12_186a. This dataset has already been converted from the source file that was offloaded from the tag into a NetCDF file. In doing so, some metadata was gleaned from the file and added to the data. Other metadata was added by hand. Use load_nc to load a NetCDF file:

Code below assumes the file is in your working directory. The example below sets the current working directory to work on Stacy's computer, but you will have to change it to match your machine.

```
setwd("C:/Users/Stacy DeRuiter/Dropbox/TagTools/data")
MN <- load_nc('mn12_186a_raw')
```

You may have to add the path for the file to the file name, e.g., if the file is in your directory:

/Users/Sebastian/tagtools/testdata

Then you need to enter:

```
MN <- load_nc('/Users/Sebastian/tagtools/testdata/mn12\_186a\_raw')
```

You may also need to escape slashes in the path specification (e.g., \) depending on your platform.

This creates an animaltag list object `MN` in your workspace. You can view it in the Environment tab if working in RStudio, or in the command line type:

```
names(MN)
str(MN$A)
#not run because output is very long!
#str(MN)
```

You should see that variables A, M, P, S, T and info are contained within the list MN.

1.4 Explore the test dataset

Now you can find out what each of the variables is by just typing their name followed by enter. For example, for info (tag-wide metadata)

```
MN$info
```

The info structure tells you where the data came from, when it was collected, on what animal, and with what type of tag. info is the general metadata store for a dataset. You can find out what is in a specific field in info, e.g., the to get the sensors_list

```
MN$info$sensors_list
```

1.5 Updating and adding metadata

You can also add new fields to the metadata, e.g., to add a field called 'data_owner' with value 'Gollum', you would enter:

```
MN$info$data_owner <- 'Gollum'
```

This is a way to add metadata to a dataset before sending it off to an archive (e.g., a dataset to accompany a journal paper). Although there are no rules for what you call each field, we have some suggestions and are working on a metadata browser/editor to help setup the metadata.

The other variables in the dataset contain data and are:

- A 3-axis acceleration
- M 3-axis magnetometry
- P depth
- T temperature
- S speed

1.6 Sensor data structures & extracting data vectors

Like info, these are structures that contain both the data and some information about it. Typing any of the variable names followed by enter will show what is in it, e.g., for P:

```
MN$P
```

Once again, in Matlab, Octave and R it will look a little different but will have all the same information. This tells you that the depth data were sampled at 1 Hz (1 sample per second) and are in units of meters. If you are more familiar with data vectors than with structures, you can easily get the data out of the structure using:

```
p <- MN$P$data
fs <- MN$P$sampling_rate
```

This makes a vector p with the depth data and a scalar fs with the sampling rate (1 in this case).

1.7 Exploratory data plots

You can plot p using:

```
plot(c(1:length(p)), p, type='l')
grid()
```

Does this plot make sense for a diving animal? To reverse the direction of the Y-axis, alter it to:

```
ys <- range(p, na.omit=TRUE)
plot(c(1:length(p)), p, type='l', ylim=c(max(ys), min(ys)))
grid()
```

The tag toolbox also has a function for plotting the data without getting it out of the structure:

```
plott(list(P=MN$P))
```

(Note there are two t's in this function name: `plott`). `plott` is a general purpose data plotter that we will use a lot. For depth data, in Matlab it automatically plots increasing depths going downwards (in R you can set input r to TRUE for panels with depth data). It also automatically converts the horizontal axis to a convenient unit like hours or days instead of seconds, or a date-time if the recording start time is given. In both plott, if input `interactive` is TRUE, you can zoom in and out of data using keyboard entries. The first input to `plott.R` is the data, which must be in the form of a list (of sensor data lists or of matrices/vectors). This is because `plott` is usually used to make multi-panel plots with multiple data inputs.

Exercise 1.

Calculate the mean duration of dives deeper than 5m Our goal with these data is to calculate the mean duration of dives deeper than 5 m. If you can think of a way to do this already, go ahead and try - you can compare your answer to the step-by-step procedure below.

As with all raw depth data, there are some problems with this dive profile. See if you can find evidence for each of these in the plot: 1. Incorrect calibration of the sensor 2. Occasional outliers 3. Coarse depth resolution 4. Temperature sensitivity

E1.1 Hints & Tips

1. look in `info` to find what species the data come from - are the depth values reasonable for this species?
2. Zoom in and see what size depth steps there are in the data
3. Use `plott` to plot both the depth and temperature:

```
plott(X=list(P=MN$P, T=MN$T), r=c(TRUE,FALSE))
```

E1.2 What to do about periods of data when the tag is not on the animal.

Not all tags have a way to start logging as soon as the tag has been deployed on the animal. Often data logging is started by a time trigger or alarm, and the researcher has to make a guess as to when the tag will be deployed to set it's start time appropriately. Often this means that a tag is logging data before it is put on an animal. Equally tags have no means of detecting when they release from the tagged animal and as a consequence may continue to log data after they release. In most cases the logged data from before and after deployment has no use. To reduce the data to just the periods when the tag is on the animal, use the tool `crop`:

```
Pc = crop(MN$P)$Y
```

This displays an interactive depth plot. Follow the instructions to select the obvious diving section of the data and then click finish (actually, if you click once on your desired left limit and twice on your desired right limit, finish will click itself and the cropping will be complete). The function returns a new data structure which contains just the selected part of the dive profile.

The resulting sensor data list also contains fields that document what you just did. They should look like:

The history keeps track of the operations that you perform on a data structure. This helps with traceability if you make the processed data available in an archive. The crop and start_time fields show how the original data was changed: the start_time is with respect to the field 'dephist_device_datetime_start' in the info structure which says when the tag recording started.

Use plott to plot Pc to make sure you cropped it correctly.

E1.3 Removing outliers.

Outliers or spikes in the data may result from errors in the tag or poor sensor performance under rapidly changing environmental conditions. For example in this data set, rapid changes in temperature and pressure as the animal surfaces cause small glitches in the data. These are not representative of the animal's behaviour so we need to remove them. A good way to do this is with a median filter. Type:

```
?median_filter
```

to find out how this function works. You call it using:

```
Pcm = median_filter(Pc,n=3)
```

Variable Pcm now contains the median filtered, cropped depth data.

Check its history to verify that the median filtering has been added.

Compare it against the unfiltered data using:

```
plott(X=list(Pc=Pc,Pcm=Pcm), r=TRUE)
```

This plots Pc in the upper panel and Pcm in the lower one.

E1.4 Correcting pressure offsets & temperature effects

The next step is to correct the '0' pressure offset of the depth sensor (so that the animal is not 10 m out of the water when it is really at the surface). We can also compensate for temperature at the same time. To do this we have to first crop the temperature data to match the pressure data. You can do this using:

```
Tc <-  crop_to(MN$T,tcues=Pc$crop)$X
```

This uses the crop information stored in Pc to do the same operation on T. The tag toolbox has a function to correct pressure data called 'fix_pressure'. Type

```
?fix_pressure
```

to find out what it does and what assumptions it makes about the data. Use this function by:

```
Pcmf <- fix_pressure(Pcm,Tc)$p
```

Compare the compensated dive profile to the uncompensated cropped one using plott. Which of the problems that we listed above have been taken care of? Any ideas what you could do about the remaining one(s)?

E1.5 Finding dives & the mean dive duration

To find the mean dive duration for dives over 5 m depth, you could measure each dive by hand on the depth plot (ginput is a useful function in Matlab and Octave for measuring data on a plot – there isn't a great equivalent in R, where interactive plots are not really commonly used).

But there is a toolbox function for this called `find_dives`. See the help on this function to find out what it does and what options it has. To find dives deeper than 5 m in your compensated dive data, type:

```
d <- find_dives(Pcmf,mindepth=5)
head(d)
```

d should return a data frame with the start, end, and maximum depth of about 51 dives (depending on where you cropped the data). How can you get the mean dive duration from this structure?

When you have got the mean dive depth, try plotting the start and end of the dives on the depth plot:

```
plott(X=list(Pcmf=Pcmf), r=TRUE)
points(d$start/(3600*24),rep(0,nrow(d)),col='green', pch=19)
points(d$end/(3600*24), rep(0,nrow(d)),col='red', pch=17)
```

Note: if you cropped the time such that the units of the x-axis are not in days, you will have to adjust the multipliers in the `points` code accordingly.

In the example above, the start and end times returned by find_dives are in seconds so we needed to divide them by 3600*24 to match the unit (days) automatically selected for time by `plott`.

Section 2: Working with accelerometers and magnetometers

2.1 Introduction

Data quality checking is fairly easy with pressure data, at least for some species, because we know what to expect - e.g., breathing at the surface in an aquatic mammal, periodic landing in a bird. Data from accelerometers and magnetometers are more difficult not only because there are three axes but also because we don't have such an intuitive feel for what they should look like. However, there are a number of quality checks we can do with A and M data that help to catch and correct problems.

Our objective here is to estimate the orientation of an animal (its pitch, roll and heading) as a function of time. We will then use these derived data to make a dead-reckoned track. To do so we first need to correct sensor offsets in A and M and also correct for how the tag is oriented on the animal. The work flow is:

1. load data and check it visually
2. reduce the sampling rate
3. make quality checks
4. correct sensor offset
5. correct for the tag orientation on the animal
6. estimate animal orientation

2.2 Loading and decimating the data

We will use data set: `testset3`. Use `load_nc` to read this file and look at info to see where it came from. You can plot the depth, acceleration and magnetometer data in the same figure using:

```
plott(X=list(P=ts3$P,A=ts3$A,M=ts3$M),
      r=c(TRUE, FALSE, FALSE))
```

Find out the sampling rate of each of A and M (by displaying each of these structures). For a large animal, this sampling rate is higher than needed to characterize orientation. We are going to reduce the data rate to 5 Hz. We do this by a signal processing operation called decimation which reduces the sampling rate by an integer factor. What 'decimation factor' (i.e., the amount we need to divide the original sampling rate) is needed to get to 5 Hz? The tag toolbox function decdc performs decimation. Type help decdc to find out how it is called. Then use it to make variables Ad and Md with a sampling rate of 5 Hz. e.g, for Ad you will enter:

(Note this will take a *long* time – the decdc function in R needs work to speed it up unfortunately.)

```
df <- ts3$A$sampling_rate/5
Ad <- decdc(ts3$A$data,df)
```

where df is the decimation factor you chose above. Display the metadata of Ad to make sure that the sampling rate is now 5 Hz.

2.3 Data quality checks

When there is not much specific acceleration, the vector magnitude of each acceleration measurement in Ad should be close to the magnitude of the gravity vector, i.e., 9.8 m/s2. You can compute the magnitude of each measurement in Ad using norm2:

```
nad <- norm2(Ad)
```

Plot nad to see if, most of the time, it is fairly close to 9.8. If not, it is an indication that the acceleration data needs re-calibration.

```
plot(c(1:length(nad)), nad, type='l')
grid()
```

The magnetometer data should also have a fairly constant vector magnitude. In this case it should be equal to the field strength where the data came from. A useful website to find the parameters of the geomagnetic field is:

https://www.ngdc.noaa.gov/geomag-web/#igrfwmm

You can find the latitude, longitude and date of the tag deployment in the info structure. You will need to select the IGRF model on the NOAA webpage as this is the only one that covers the year of the deployment.

Now calculate the norm2 of the magnetometer data, plot it and compare to the expected field strength.

Does it look reliable? The final data quality check we can do on the accelerometer and magnetometer data is to compute the inclination angle between them. The toolbox function for this is:

```
inc <- inclination(Ad,Md)
inc
```

Once again, check what the value should be from the website and then plot inc to see if it is fairly close. Bear in mind that inc is in radians (the tagtools keeps all angles in radians except for latitude and longitude) so you will need to convert to degrees before plotting it. If you have forgotten how to, there is a function in R's `pracma` and `calibrate` packages called `deg2rad` and `rad2degree` which will save you from embarrasment.

2.4 Correcting offsets in the data

You may have found that the magnetometer data does not seem to have a constant field strength. It is very common that tags acquire a small residual magnetic field and this changes the offset of the magnetometer data from the calibration values. There is a tagtool called fix_offset_3d that can determine the offset correction for each axis of a vector field sensor (i.e., accelerometers and magnetometers but not gyroscopes). Use fix_offset_3d to improve the offset of Md:

```
Mdc <- fix_offset_3d(Md)
```

Have a look at the sensor structure Mdc. You will see that a new field has been added called cal_poly. This contains the offset corrections that have been calculated by fix_offset_3d. Now we need to re-run the data quality checks. A shortcut for doing this is the function check_AM. This computes the field strength of A and M and their inclination angle in one step:

```
AMchecks <- check_AM(Ad,Mdc)
```

Plot the results to see if they have improved. Note that v is a two column matrix. The first column is the norm2 of Ad and the second column is the norm2 of Mdc. Also bear in mind that the angle between accelerometer and magnetometer measurements is always a bit noisy so don't expect inc to be right on the expected inclination angle.

2.5 Estimating and checking pitch and roll

If the data now pass the quality checks, we can go on to calculate the Euler angles: pitch, roll and heading from Ad and Mdc. Pitch and roll can be estimated from Ad and the tagtool a2pr does this:

```
pitchroll <- a2pr(Ad)
```

Pitch and roll are each vectors at the same sampling rate as Ad. Plot these and check if they are reasonable, remembering that angles are in radians. In particular, is the animal in a realistic orientation when it is at the surface? You can plot the pitch and roll data along with the dive profile using plott...

```
plott(X=list(P=ts3$P$data, pitch=pr$p,roll=pr$r),fsx=c(25,5,5))
```

The 5 here is the sampling rate of the pitch and roll which are derived from your decimated (5Hz) accelerometer data.

2.6 Correcting for the tag orientation on the animal

The tag in this case slid to an unusual location and we had to infer where it was on the animal from the tag data. The inferred orientation of the tag with respect to the animal is:

pitch = -21 degrees roll = -177 degrees yaw = 8 degrees

Think about where the tag must be on the animal to have these orientations. We need to use these angles to rotate Ad and Mdc so as to correct for orientation. We do this by first computing the rotation matrix that will rotate by these angles. This rotation matrix is a transformation the maps tag-frame data into animal frame data. We compute the matrix using:

```
Q <- euler2rotmat(pi/180*-21, pi/180*-177, pi/180*8)
```

Note that the angles are entered in radians, hence the pi/180 correction. Q is a 3x3 matrix. Each accelerometer and magnetometer measurement must be multiplied by this matrix to convert them into the animal frame. This is done using rotate_vecs:

```
Adr = rotate_vecs(Ad,Q)
Mdr = rotate_vecs(Mdc,Q)
```

Adr and Mdr should now be as if the data were recorded by a tag oriented with its axes coinciding with the animal's axes. Re-compute pitch and roll using Adr. Plot the results and check if they make more sense.

2.7 Estimating heading

If we are now ok with pitch and roll, we need to also compute heading. To get heading, use the tagtool m2h. This requires both accelerometer and magnetometer data. This is because the magnetometer data have to first be 'gimballed' by the pitch and roll to make the measurement that would have been made by a horizontal tag. The heading is then calculated from that. Tagtool m2h does both of these steps:

```
head <- m2h(Adr,Mdr)
```

Plot the heading along with the pitch and roll. Again, remember that these angles are in radians and that heading is also with respect to magnetic north, i.e., it is not corrected for the declination angle.

It is not easy to infer much from plots of Euler angles especially roll and heading. A better way to visualize these data is to animate them. The matlab tagtool kit has a plotting tool that will show a whale (reflecting the marine bias of its authors!) that is animated by pitch, roll and heading data. (R version coming soon.)

2.8 Saving the corrected data

When you are comfortable that the data pass all of the quality checks, save a new NetCDF file with the revised data. First decimate the depth data to the same rate as the other sensors using the decimation factor, df, you calculated earlier:

```
Pd = decdc(ts3$P,df)
```

Then save the corrected and decimated data as follows:

```
save_nc('sw03_253a_5Hz_animal_frame',X= list(Adr,Mdr,Pd))
```

There is no need to also save pitch, roll and heading because these can be easily re-computed from Adr and Mdr. The archive file should only contain source data or data that has been corrected. Note that because all of the correction steps that you used to get Adr and Mdr are stored in their structures, this information will also be saved automatically in your archive file. This means that your processing steps are traceable.