Chapter 7
Tracking Animals in a Dynamic Environment: Remote Sensing Image Time Series

Mathieu Basille, Ferdinando Urbano, Pierre Racine, Valerio Capecchi and Francesca Cagnacci

Abstract This chapter looks into the spatiotemporal dimension of both animal tracking data sets and the dynamic environmental data that can be associated with them. Typically, these geographic layers derive from remote sensing measurements, commonly those collected by sensors deployed on earth-orbiting satellites, which can be updated on a monthly, weekly or even daily basis. The modelling potential for integrating these two levels of ecological complexity (animal movement and environmental variability) is huge and comes from the possibility to investigate processes as they build up, i.e. in a full dynamic framework. This chapter's exercise will describe how to integrate dynamic environmental data in the spatial database and join to animal locations one of the most used indices for ecological productivity and phenology, the normalised difference vegetation index (NDVI) derived from MODIS. The exercise is based on the database built so far in Chaps. 2, 3, 4, 5 and 6.

M. Basille (✉)
Fort Lauderdale Research and Education Center, University of Florida,
3205 College Avenue, Fort Lauderdale, FL 33314, USA
e-mail: basille@ase-research.org

F. Urbano
Università Iuav di Venezia, Santa Croce 191 Tolentini, 30135 Venice, Italy
e-mail: ferdi.urbano@gmail.com

P. Racine
Centre for Forest Research, University Laval, Pavillon Abitibi-Price,
2405 de la Terrasse, Bureau 1122, Quebec City, QC G1V 0A6, Canada
e-mail: pierre.racine@sbf.ulaval.ca

V. Capecchi
Istituto di Biometeorologia, Consiglio Nazionale delle Ricerche,
Via Madonna del piano 10 50019 Sesto Fiorentino, Firenze, Italy
e-mail: capecchi@lamma.rete.toscana.it

F. Cagnacci
Biodiversity and Molecular Ecology Department, Research and Innovation Centre,
Fondazione Edmund Mach, via E. Mach 1, 38010 S.Michele all'Adige, TN, Italy
e-mail: francesca.cagnacci@fmach.it

Keywords NDVI · Raster time series · Spatial database · Spatiotemporal intersection

Introduction

The advancement in movement ecology from a data perspective can reach its full potential only by combining the technology of animal tracking with the technology of other environmental sensing programmes (Cagnacci et al. 2010). Ecology is fundamentally spatial, and animal ecology is obviously no exception (Turchin 1998). Any scientific question in animal ecology cannot overlook its spatial dimension, and in particular the dynamic interaction between individual animals or populations, and the environment in which the ecological processes occur. Movement provides the mechanistic link to explain this complex ecosystem interaction, as the movement path is dynamically determined by external factors, through their effect on the individual's state and the life history characteristics of an animal (Nathan et al. 2008). Therefore, most modelling approaches for animal movement include environmental factors as explanatory variables. As illustrated in earlier portions of this book, this technically implies the intersection of animal locations with environmental layers, in order to extract the information about the environment that is embedded in spatial coordinates. It appears very clear at this stage, though, that animal locations are not only spatial, but are also fully defined by spatial and temporal coordinates (as given by the acquisition time).

Logically, the same temporal definition also applies to environmental layers. Some characteristics of the landscape, such as land cover or road networks, can be considered static over a large period of time (of the order of several years), and these static environmental layers are commonly intersected with animal locations to infer habitat use and selection by animals (e.g. Resource Selection Functions, RSF, Manly et al. 2002). However, many characteristics relevant to wildlife, such as vegetation biomass or road traffic, are indeed subject to temporal variability (of the order of hours to weeks) in the landscape and would be better represented by dynamic layers that correspond closely to the conditions actually encountered by an animal moving across the landscape (Moorcroft 2012). In this case, using static environmental layers directly limits the potential of wildlife tracking data, reduces the power of inference of statistical models and sometimes even introduces sources of bias (Basille et al. 2013).

Nowadays, satellite-based remote sensing can provide dynamic global coverage of medium-resolution images that can be used to compute a large number of environmental parameters very useful to wildlife studies. Through remote sensing, it is possible to acquire spatial time series which can then be linked to animal locations, fully exploiting the spatiotemporal nature of wildlife tracking data. Numerous satellites and other sensor networks can now provide information on resources on a monthly, weekly or even daily basis, which can be used as explanatory variables in statistical models (e.g. Pettorelli et al. 2006) or to parameterise Bayesian inferences or mechanistic models. One of the most commonly used satellite-derived environmental time series is the normalised difference vegetation index (NDVI) but other examples include data sets on ocean primary productivity, surface temperature or salinity, all available in equally fine spatial and temporal scales (McClain 2009), and, in North America, snow depth data at daily scales (SNODAS, at 1-km resolution), spatial temperature and precipitation at monthly scales (PRISM data model, at 1-km resolution), and meteorological data on wind and pressure (ESRL[1]). Snow cover, NDVI and sea surface temperature are some examples of indices that can be used as explanatory variables in statistical models (e.g. Pettorelli et al. 2006) or to parameterise Bayesian inferences or mechanistic models. Moreover, there are user-friendly spatial tools to acquire (LPDAAC NASA website 2008[2]) and process (e.g. Marine Geospatial Ecology Tools—MGET[3], a plugin for the proprietary software ESRI ArcGIS and the free Movebank tool Env-DATA System[4]) data from the moderate-resolution imaging spectroradiometer (MODIS), a major provider of NDVI.

The main shortcoming of such remote sensing layers is the relatively low spatial resolution (e.g. 250 m for MODIS, e.g. Cracknell 1997; 1 km for SPOT vegetation, e.g. Maisongrande et al. 2004), which does not fit the current average bias of wildlife tracking GPS locations (less than 20 m, see Frair et al. 2010 for a review), thus potentially leading to a spatial mismatch between the animal-based information and the environmental layers (note that the resolution can still be perfectly fine, depending on the overall spatial variability and the species and biological process under study). Yet, this is much more desirable than using static layers when the temporal variability is an essential component of the ecological inference (Basille et al. 2013). Higher resolution images and new types of information (e.g. forest structure) are presently provided by new types of sensors, such as those from LIDAR, radar or hyper-spectral remote sensing technology. However, the use of these images for intersection with wildlife tracking data sets is still limited by the high cost of source data, including direct costs such as flights of aircrafts, that restricts the collection of comprehensive high-resolution time series. In the case of animals that move over large distances (regions, nations, continents), these limitations are even greater. Few software packages provide the functionalities to handle time series of images easily (Eerens et al. 2014), while at the same time offering a complete set of the tools required by movement ecology in general and tracking data in particular.

In this chapter, we discuss the integration in the spatial database of one of the most used indices for ecological productivity and phenology, i.e. NDVI, derived from MODIS images. The intersection of NDVI images with GPS locations requires a

[1] http://www.esrl.noaa.gov/.

[2] https://lpdaac.usgs.gov/.

[3] http://mgel.env.duke.edu/mget.

[4] http://www.movebank.org/node/6607.

system that is able to handle large amounts of data and explicitly manage both spatial and temporal dimensions, which makes PostGIS an ideal candidate for the task.

MODIS NDVI Data Series

The MODIS instrument operates on NASA's Terra and Aqua spacecraft. The instrument views the entire earth's surface every 1–2 days and captures data in 36 spectral bands ranging in wavelengths from 0.4 to 14.4 μm and at varying spatial resolutions (250 m, 500 m and 1 km). The global MODIS vegetation indices (MODIS 13 products, MODIS 1999) are designed to provide consistent spatial and temporal comparisons of vegetation conditions. Red and near-infrared reflectances, centred at 645 and 858 nm, respectively, are used to determine vegetation indices, including the well-known NDVI, at daily, 8 d, 16 d and monthly scales. This index is calculated by contrasting intense chlorophyll pigment absorption in the red against the high reflectance of leaf mesophyll in the near infrared. It is a proxy of plant photosynthetic activity and has been found to be highly related to the green leaf area index (LAI) and to the fraction of photosynthetically active radiation absorbed by vegetation (FAPAR; see for instance Bannari et al. 1995).

Past studies have demonstrated the potential of using NDVI data to study vegetation dynamics (Townshend and Justice 1986; Verhoef et al. 1996). More recently, several applications have been developed using MODIS NDVI data such as land cover change detection (Lunetta et al. 2006), monitoring forest phenophases (Yu and Zhuang 2006), modelling wheat yield (Moriondo et al. 2007) and other applications in forest and agricultural sciences. However, the utility of the MODIS NDVI data products is limited by the availability of high-quality data (e.g. cloud free), and several processing steps are required before using the data: acquisition via Web facilities, reprojection from the native sinusoidal projection to a standard latitude–longitude format, eventually the mosaicking of two or more tiles into a single tile. A number of processing techniques to 'smooth' the data and obtain a cleaned (no clouds) time series of NDVI imagery have also been implemented. These kinds of processes are usually based on a set of ancillary information on the data quality of each pixel that is provided together with MODIS NDVI.

In the framework of the present project, a simple R[5] procedure has been adapted in order to download, reproject and mosaic the NDVI data. The procedure is flexible and can be modified to work with other MODIS data (snow, land surface temperature, etc.). It is dependent on the MODIS Reprojection Tool (MRT), a set of tools developed by the NASA to manipulate MODIS files. MRT enables users to read data files in the native HDF-EOS format, specify a geographic subset or specific science data sets as input to processing, perform geographic

[5] http://www.r-project.org/.

transformation to a different coordinate system/cartographic projection and write the output to a GDAL-compatible file format.

A preliminary visual examination of the available NDVI images indicates that they may contain a variable number of pixels with erroneous values. Several techniques have been developed to remove these pseudo-hikes and drops in the time series (probably due to clouds) and substitute the missing data with a reliable value. In the present case, we applied a very simple but efficient procedure, first developed and described by Escadafal et al. (2001), for the 10-day NOAA-AVHRR NDVI images. The procedure was also applied in other similar cases in the European context (Maselli et al. 2006) with the 10-day NOAA-AVHRR and SPOT-VGT NDVI images. Here, we adapted the procedure to work with the 16-day MODIS images. The procedure simply consists of a preliminary filtering in order to remove isolated pixels with anomalous NDVI values and replace them with local (5-point) averages. The final result of such a procedure is shown in Fig. 7.1, which displays NDVI values extracted before and after the smoothing for a location in the municipality of Terlago (46.10°N, 11.05°E, northern Italy). Note that, for convenience purposes, NDVI values have been multiplied by 10,000 to be stored as integers with a maximum value of 10,000 (you will have to keep this in mind when presenting NDVI values).

Dealing with Raster Time Series

Raster time series are quite common from medium- and low-resolution data sets generated by satellites that record information at the same location on earth at regular time intervals. In this case, each pixel has both a spatial and a temporal reference. In this exercise, you integrate an NDVI data set of 92 MODIS images covering the period 2005–2008 (spatial resolution of 1 km and temporal resolution of 16 days). In this example, you will use the *env_data* schema to store raster time series, in order to keep it transparent to the user: all environmental data (static or dynamic) are in this schema. However, over larger amounts of data, it might be useful to store raster time series in a different schema to support an easier and more efficient backup procedure.

When you import a raster image using *raster2pgsql*, a new record is added in the target table for each raster, including one for each tile if the raster was tiled (see Chap. 6). At this point, each record does not consider time yet and is thus simply a spatial object. To transform each record into a spatiotemporal object, you must add a field with the timestamp of the data acquisition, or, better, the time range covered by the data if it is related to a period. The time associated with each raster (and each tile) can usually be derived from the name of the file, where this information is typically embedded. In the case of MODIS composite over 16 days, this is the first day of the 16-day period associated with the image in the form '*MODIS_NDVI_yyyy_mm_dd.tif*' where *yyyy* is the year, *mm* the month, and *dd* the day.

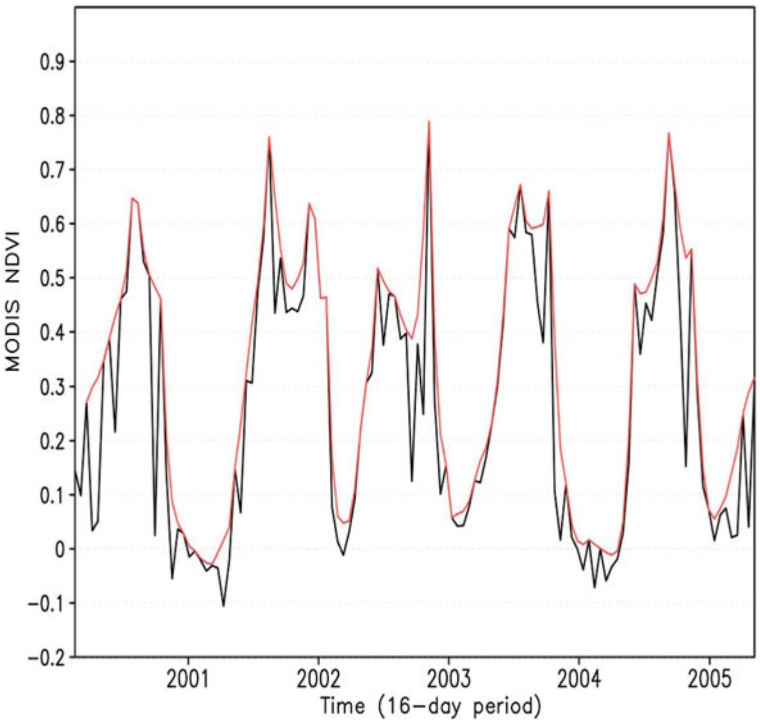


Fig. 7.1 Temporal profiles of raw (*black line*) and cleaned (*red line*) MODIS NDVI data over a 5-year period (2000–2004) for a location in the municipality of Terlago (46.10°N, 11.05°E, northern Italy)

Time Ranges in PostgreSQL

A new data type was introduced in PostgreSQL 9.2 to facilitate storage and manipulation of ranges, i.e. an interval of values with a beginning and an end. The range type[6], stored in a single column, eliminates the need to store the beginning and end values in two separate columns and allows for much simpler comparisons as there is no longer a need to check for both bounds. Ranges can be discrete (e.g. every integer from 0 to 10 included) or continuous (e.g. all real numbers between 0 and 10 included, or any point in time between 1 January and 30 January 2013).

The simplest example would be a series of integers, say from 1 to 10:

```
SELECT int4range(1, 10);
```

[6] http://www.postgresql.org/docs/9.2/static/rangetypes.html.

The result is

```
 int4range
-----------
 [1,10)
```

A range is defined solely by its lower and upper bounds, and, by default, assumes that the lower bound is included, but the upper bound is excluded. In other words, the former query defined the series 1, 2, 3, 4, 5, 6, 7, 8, 9 and did not include 10. As can be seen from the output, the convention used by PostgreSQL is a square bracket for an inclusion and a parenthesis for an exclusion. Each range type has a constructor function with the same name as the range type (e.g. *int4range*, *numrange,* or *daterange*), and accepts as a third argument the specification of the bound (e.g. *'[)'*, which is the default setting). Hence, a range of dates (from 20 March 2013 to 22 September 2013 inclusive) would be constructed like this:

```
SELECT daterange('2013-03-20', '2013-09-22', '[]');
```

The result is

```
       daterange
-------------------------
 [2013-03-20,2013-09-23)
```

Note that in this case, PostgreSQL transformed the result with the default '*[)*' notation, which excludes the upper bound, using the next day. The same result can be achieved by the use of an explicit formulation 'casted' to the range type of interest:

```
SELECT '[2013-03-20, 2013-09-22]'::daterange;
```

This query results in the exact same output, i.e. the period containing all days from 20 March 2013 (included) to 22 September 2013 (included), which defines the period between the two equinoxes when days are longer than nights in the northern hemisphere in 2013. To avoid any confusion, it might be a good practice to use explicit formulas, which forces the declaration of the bounds, and are hence less error prone. You could be even more precise and specify the exact times between the two equinoxes using the *tsrange* type:

```
SELECT '[2013-03-20 11:01:55, 2013-09-22 20:44:08)'::tsrange;
```

The result is

```
                   tsrange
-----------------------------------------------
 ["2013-03-20 11:01:55","2013-09-22 20:44:08")
```

The main interest of using a range is to easily check whether a value (or another range) is included in it. Note that only elements from the same type can be compared (e.g. dates with dates, timestamps with timestamps). You thus need to use explicit casting when necessary. Let us check for instance whether you are currently between the two equinoxes of 2013, using the operator '@>' (containment):

```
SELECT '[2013-03-20, 2013-09-22]'::daterange @> now()::date AS in_range;
```

If you run this query after 22 September 2013, the output of the last query is likely to be false. Other useful operators include equality (=), union (+), intersection (*) or overlap (&&)[7]:

```
SELECT
  '[2013-03-20, 2013-09-22]'::daterange = daterange('2013-03-20',
  '2013-09-22', '[]') AS equal_range;
SELECT
  '[2013-03-20, 2013-09-22]'::daterange + '[2013-06-01, 2014-01-01)'
  ::daterange AS union_range;
SELECT
  '[2013-03-20, 2013-09-22]'::daterange * '[2013-06-01, 2014-01-01)'
  ::daterange AS intersection_range;
SELECT
  '[2013-03-20, 2013-09-22]'::daterange && '[2013-06-01, 2014-01-01)'
  ::daterange AS overlap_range;
```

While the first two return '*t*', the union range is '*[2013-03-20,2014-01-01)*' and the intersection is '*[2013-06-01,2013-09-23)*'.

Finally, note that a range can be infinite on one side and thus does not have a lower or a upper bound. This is achieved by using the NULL bound in the constructor or an empty value in the explicit formulation, for example

```
SELECT
  '[2013-01-01,)'::daterange;
```

or

```
SELECT
  '[2013-01-01,)'::daterange @> now()::date AS after_2013;
```

[7] See the full list of operators and functions here: http://www.postgresql.org/docs/9.2/static/functions-range.html.

Import the Raster Time Series

With this data type, you can now associate each image or tile with the correct time reference, that is, the 16-day period associated with each raster. This will make the spatiotemporal intersection with GPS positions possible by allowing direct comparisons with GPS timestamps.

To start, create an empty table to store the NDVI images, including a field for the temporal reference (of type *daterange*) and its index:

```
CREATE TABLE env_data.ndvi_modis(
  rid serial NOT NULL,
  rast raster,
  filename text,
  acquisition_range daterange,
  CONSTRAINT ndvi_modis_pkey
    PRIMARY KEY (rid));

CREATE INDEX ndvi_modis_wkb_rast_idx
  ON env_data.ndvi_modis
  USING GIST (ST_ConvexHull(rast));

COMMENT ON TABLE env_data.ndvi_modis
IS 'Table that stores values of smoothed MODIS NDVI (16-day periods).';
```

Now, the trick is to use two arguments of the *raster2pgsql* command (see also Chap. 6): *-F* to include the raster file name as a column of the table (which will then be used by the trigger function) and *-a* to append the data to an existing table, instead of creating a new one. Another aspect is the absence of a flagged 'no data' value in the MODIS NDVI files. In these rasters, -3000 represents an empty pixel, but this information is not stored in the raster: you will have to declare it explicitly using the '*-N*' argument. The NDVI data used here consist of 92 tif images from January 2005 to December 2008, but you can import all of them in a single operation using the wildcard character '*' in the input filename. You can thus run the following command in the Command Prompt[8] (warning: you might need to adjust the rasters' path according to your own set-up):

```
C:\Program Files\PostgreSQL\9.2\bin\raster2pgsql.exe -a -C -F -M -s 4326 -t
20x20 -N -3000 C:\tracking_db\data\env_data\raster\raster_ts\*.tif
env_data.ndvi_modis | psql -p 5432 -d gps_tracking_db -U postgres
```

You can confirm that the raster was properly loaded with all its attributes by looking at the *raster_columns* view, which stores raster metadata (here, you only retrieve the table's schema, name, SRID and NoData value, but it is a good practice to examine all information stored in this view):

[8] Note that this is not an a SQL code and cannot be run in an SQL interface.

```
SELECT
  r_table_schema AS schema,
  r_table_name AS table,
  srid,
  nodata_values AS nodata
FROM raster_columns
WHERE r_table_name = 'ndvi_modis';
```

```
 schema   |    table     | srid | nodata
----------+------------+------+--------
 env_data | ndvi_modis | 4326 | {-3000}
```

Each raster file embeds the acquisition period in its filename. For instance, 'MODIS_NDVI_2005_01_01.tif' is associated with the period from 1 January 2005 (included) to 17 January 2005 (excluded). As you can see, the period is encoded on 10 characters following the common prefix 'MODIS_NDVI_'. This allows you to use the *substring* function to extract the year, the month and the starting day from the filename (which was automatically stored in the *filename* field during the import). For instance, you can extract the starting date from the first raster imported (which should be *'MODIS_NDVI_2005_01_01.tif'*):

```
SELECT filename,
  (substring(filename FROM 12 FOR 4) || '-' ||
   substring(filename FROM 17 FOR 2) || '-' ||
   substring(filename FROM 20 FOR 2))::date AS start
FROM env_data.ndvi_modis LIMIT 1;
```

```
         filename          |   start
---------------------------+------------
 MODIS_NDVI_2005_01_01.tif | 2005-01-01
```

The same approach can be used to define the ending date of the period, by simply adding 16 days to the previous date (remember that this day will be excluded from the range). Note that adding 16 days takes into account the additional day at the end of February in leap years:

```
SELECT filename,
  (substring(filename FROM 12 FOR 4) || '-' ||
   substring(filename FROM 17 FOR 2) || '-' ||
   substring(filename FROM 20 FOR 2))::date + 16 AS end
FROM env_data.ndvi_modis LIMIT 1;
```

```
         filename          |    end
---------------------------+------------
 MODIS_NDVI_2005_01_01.tif | 2005-01-17
```

In the case of more complex filenames with a variable number of characters, you could still retrieve the encoded date using the *substring* function, by extracting the relevant characters relative to some other characters found first using the *position* function. Let us now update the table by converting the filenames into the date ranges according to the convention used in file naming (note that there is an additional constraint that selects 1 January when the start date + 16 days exceeds the beginning of the year):

```
UPDATE env_data.ndvi_modis
SET acquisition_range = daterange(
  (substring(filename FROM 12 FOR 4) || '-' ||
   substring(filename FROM 17 FOR 2) || '-' ||
   substring(filename FROM 20 FOR 2))::date,
  LEAST((substring(filename FROM 12 FOR 4) || '-' ||
         substring(filename FROM 17 FOR 2) || '-' ||
         substring(filename FROM 20 FOR 2))::date + 16,
        (substring(filename FROM 12 FOR 4)::integer + 1
        || '-' || '01' || '-' || '01')::date));
```

As for any type of column, if the table contains a large number of rows (e.g. >10,000), querying based on the *acquisition_range* will be faster if you first index it (you can do it even if the table is not that big, as the PostgreSQL planner will determine whether the query will be faster by using the index or not):

```
CREATE INDEX ndvi_modis_acquisition_range_idx
ON env_data.ndvi_modis (acquisition_range);
```

Now, each tile (and therefore each pixel) has a spatial and a temporal component and thus can be queried according to both criteria. For instance, these are the 10 first tiles corresponding to 1 March 2008, using the '@>' operator ('contains'). Note that this is a leap year so that the corresponding period ends on 5 March:

```
SELECT rid, filename, acquisition_range
FROM env_data.ndvi_modis
WHERE acquisition_range @> '2008-03-01'::date
LIMIT 10;
```

```
 rid  |         filename          |    acquisition_range
------+---------------------------+-------------------------
 3889 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3890 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3891 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3892 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3893 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3894 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3895 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3896 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3897 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
 3898 | MODIS_NDVI_2008_02_18.tif | [2008-02-18,2008-03-05)
```

Based on this, you can now create a trigger and its associated function to automatically create the appropriate date range during the NDVI data import. Note that the *ndvi_acquisition_range_update* function will be activated before an NDVI tile is loaded, so that the transaction is aborted if, for any reason, the *acquisition_range* cannot be computed, and only valid rows are inserted into the *ndvi_modis* table:

```
CREATE OR REPLACE FUNCTION tools.ndvi_acquisition_range_update()
RETURNS trigger AS
$BODY$
BEGIN
  NEW.acquisition_range = daterange(
    (substring(NEW.filename FROM 12 FOR 4) || '-' ||
     substring(NEW.filename FROM 17 FOR 2) || '-' ||
     substring(NEW.filename FROM 20 FOR 2))::date,
    LEAST((substring(NEW.filename FROM 12 FOR 4) || '-' ||
           substring(NEW.filename FROM 17 FOR 2) || '-' ||
           substring(NEW.filename FROM 20 FOR 2))::date + 16,
          (substring(NEW.filename FROM 12 FOR 4)::integer + 1
          || '-' || '01' || '-' || '01')::date));
RETURN NEW;
END;
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
COMMENT ON FUNCTION tools.ndvi_acquisition_range_update()
IS 'This function is raised whenever a new record is inserted into the MODIS
NDVI time series table in order to define the date range. The
acquisition_range value is derived from the original filename (that has the
structure MODIS_NDVI_YYYY_MM_DD.tif)';

CREATE TRIGGER update_ndvi_acquisition_range
BEFORE INSERT ON env_data.ndvi_modis
  FOR EACH ROW EXECUTE PROCEDURE tools.ndvi_acquisition_range_update();
```

Every time you add new NDVI rasters, the *acquisition_range* will then be updated appropriately. At this stage, your database contains all environmental data proposed for the database in this book and should look like Fig. 7.2 (using the DB Manager in QGIS).

Intersection of Locations and NDVI Rasters

To intersect a GPS position with this kind of data set, both temporal and spatial criteria must be defined. In the next example, you retrieve the MODIS NDVI value at point (11, 46) using the *ST_Value* PostGIS SQL function and for the whole year 2005 with the '&&' operator ('overlap'):

```
SELECT
  rid,
  acquisition_range,
  ST_Value(rast, ST_SetSRID(ST_MakePoint(11, 46), 4326)) / 10000 AS ndvi
FROM env_data.ndvi_modis
WHERE ST_Intersects(ST_SetSRID(ST_MakePoint(11, 46), 4326), rast)
  AND acquisition_range && '[2005-01-01,2005-12-31]'::daterange
ORDER BY acquisition_range;
```

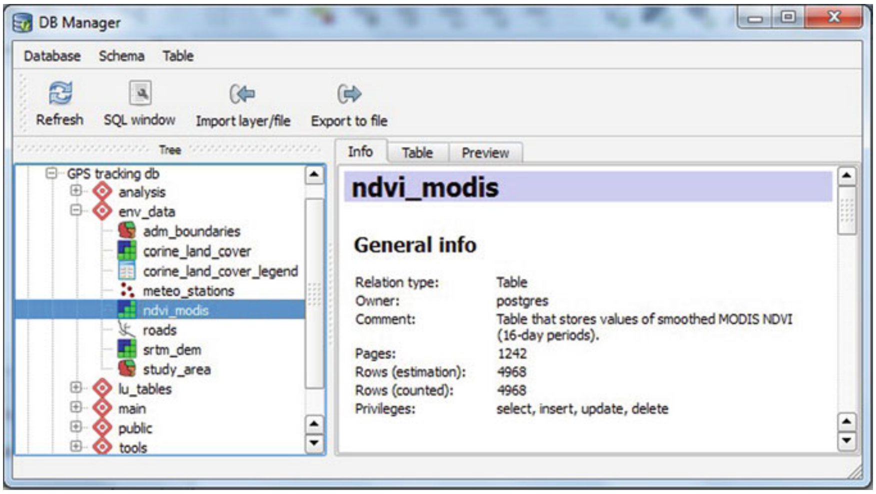
The result gives you the complete NDVI profile at this location for the year 2005:

```
 rid  |    acquisition_range     |  ndvi
------+-------------------------+--------
   31 | [2005-01-01,2005-01-17) | 0.7047
   85 | [2005-01-17,2005-02-02) | 0.6397
  139 | [2005-02-02,2005-02-18) | 0.5974
  193 | [2005-02-18,2005-03-06) | 0.5645
  247 | [2005-03-06,2005-03-22) | 0.5745
  301 | [2005-03-22,2005-04-07) | 0.6076
  355 | [2005-04-07,2005-04-23) |  0.649
  409 | [2005-04-23,2005-05-09) | 0.8086
  463 | [2005-05-09,2005-05-25) | 0.8511
  517 | [2005-05-25,2005-06-10) | 0.8935
  571 | [2005-06-10,2005-06-26) | 0.8935
  625 | [2005-06-26,2005-07-12) | 0.8951
  679 | [2005-07-12,2005-07-28) | 0.8979
  733 | [2005-07-28,2005-08-13) | 0.9006
  787 | [2005-08-13,2005-08-29) |  0.907
  841 | [2005-08-29,2005-09-14) | 0.8682
  895 | [2005-09-14,2005-09-30) | 0.8441
  949 | [2005-09-30,2005-10-16) | 0.7556
 1003 | [2005-10-16,2005-11-01) | 0.6895
 1057 | [2005-11-01,2005-11-17) | 0.6979
 1111 | [2005-11-17,2005-12-03) | 0.7291
 1165 | [2005-12-03,2005-12-19) | 0.7778
 1219 | [2005-12-19,2006-01-01) | 0.9654
```

In Fig. 7.3, the NDVI variation for the year is displayed in graphical format (screenshot taken from QGIS).

You can now retrieve NDVI values at coordinates from real animals:

```
SELECT
  animals_id AS ani_id,
  ST_X(geom) AS x,
  ST_Y(geom) AS y,
  acquisition_time,
  ST_Value(rast, geom) / 10000 AS ndvi
FROM main.gps_data_animals, env_data.ndvi_modis
WHERE ST_Intersects(geom, rast)
  AND acquisition_range @> acquisition_time::date
ORDER BY ani_id, acquisition_time
LIMIT 10;
```



GPS tracking db
- analysis
- env_data
  - adm_boundaries
  - corine_land_cover
  - corine_land_cover_legend
  - meteo_stations
  - ndvi_modis
  - roads
  - srtm_dem
  - study_area
- lu_tables
- main
- public
- tools

ndvi_modis

General info

Relation type:	Table
Owner:	postgres
Comment:	Table that stores values of smoothed MODIS NDVI (16-day periods).
Pages:	1242
Rows (estimation):	4968
Rows (counted):	4968
Privileges:	select, insert, update, delete

Fig. 7.2 Summary of the different environmental layers using the DB Manager in QGIS, showing the type of data in the database (*lines*, *polygons*, *raster* or *simple tables*)

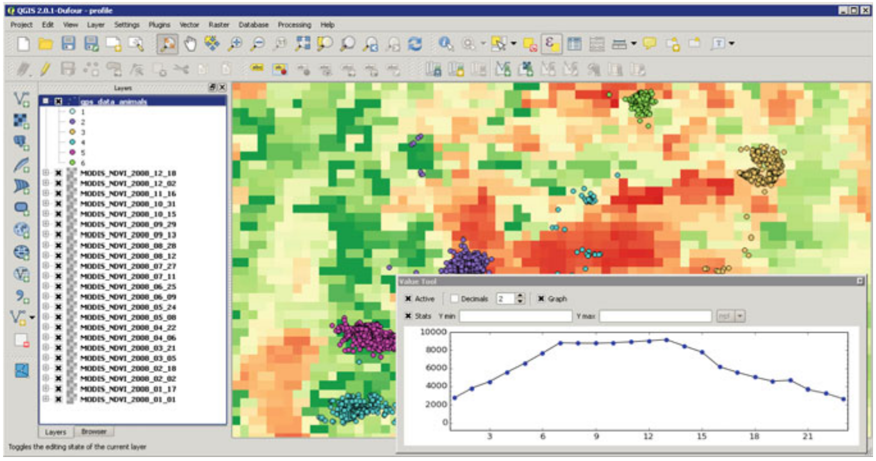


Fig. 7.3 Example of complete NDVI profile for the year 2005 in a pixel in the study area. Remember that NDVI values have been multiplied by 10,000

```
 ani_id |    x     |    y     |     acquisition_time      |  ndvi
--------+--------+--------+------------------------+--------
      1 | 11.044 | 46.011 | 2005-10-18 16:00:54-04 | 0.5426
      1 | 11.045 | 46.012 | 2005-10-18 20:01:23-04 | 0.5426
      1 | 11.045 | 46.008 | 2005-10-19 00:02:22-04 | 0.6566
      1 | 11.046 | 46.006 | 2005-10-19 04:03:08-04 | 0.5839
      1 | 11.043 | 46.010 | 2005-10-20 16:00:53-04 | 0.5528
      1 | 11.042 | 46.011 | 2005-10-20 20:00:48-04 | 0.5528
      1 | 11.041 | 46.010 | 2005-10-21 00:00:53-04 | 0.5429
      1 | 11.044 | 46.007 | 2005-10-21 04:01:42-04 | 0.6566
      1 | 11.046 | 46.007 | 2005-10-21 12:01:16-04 | 0.6566
      1 | 11.038 | 46.009 | 2005-10-21 16:01:23-04 | 0.5252
```

Now, as an example of the capabilities of PostGIS, let us try to retrieve NDVI values in animal home ranges during a whole season. In habitat selection studies, the habitat used by an individual is generally compared to the habitat that is considered available to the individual. In this example, we assume that the convex polygon encompassing all locations of the winter 2005–2006 defines the area available during this season. In contrast, the exact locations determine what was used by the animals. You will then, for each animal monitored during the winter 2005–2006, compute the average NDVI value at all locations and the average NDVI value in the area covered during the same season. The following query uses the *WITH*[9] syntax, which allows you to break down a seemingly complex query: in this case, you first compute the convex polygons during winter (in *mcp*), then extract the NDVI values in these polygons (in *ndvi_winter_mcp*), and then extract NDVI values at the GPS locations (in *ndvi_winter_locs*). Finally, in the last part of the query, you just display the relevant information:

```
WITH
  mcp AS (
    SELECT
      animals_id,
      min(acquisition_time) AS start_time,
      max(acquisition_time) AS end_time,
      ST_ConvexHull(ST_Collect(geom)) AS geom
    FROM main.gps_data_animals
    WHERE acquisition_time >= '2005-12-21'::date
      AND acquisition_time < '2006-03-21'::date
    GROUP BY animals_id),
  ndvi_winter_mcp AS (
    SELECT
      animals_id,
      start_time,
      end_time,
      ST_SummaryStats(ST_Clip(ST_Union(rast), geom)) AS ss,
```

9 http://www.postgresql.org/docs/9.2/static/queries-with.html.

```
      acquisition_range
    FROM mcp, env_data.ndvi_modis
    WHERE ST_Intersects(geom, rast)
      AND lower(acquisition_range) >= '2005-12-21'::date
      AND lower(acquisition_range) < '2006-03-21'::date
    GROUP BY animals_id, start_time, end_time, geom,
      acquisition_range),
  ndvi_winter_locs AS (
    SELECT
      animals_id,
      avg(ST_Value(rast, geom)) / 10000 AS mean
    FROM main.gps_data_animals, env_data.ndvi_modis
    WHERE acquisition_time >= '2005-12-21'::date
      AND acquisition_time < '2006-03-21'::date
      AND ST_Intersects(geom, rast)
      AND acquisition_range @> acquisition_time::date
    GROUP BY animals_id)
SELECT
  m.animals_id AS id,
  m.start_time::date,
  m.end_time::date,
  avg((m.ss).mean) / 10000 AS mean_mcp, l.mean AS mean_loc
FROM ndvi_winter_mcp AS m, ndvi_winter_locs AS l
WHERE m.animals_id = l.animals_id
GROUP BY m.animals_id, m.start_time, m.end_time, l.mean
ORDER BY m.animals_id;
```

Note that this complex query takes less than one second! The results indicate that three out of four roe deer actually use greater NDVI values in winter than generally available to them:

```
 id | start_time |  end_time  | mean_mcp | mean_loc
----+------------+------------+----------+----------
  1 | 2005-12-21 | 2006-03-20 |    0.424 |    0.454
  2 | 2005-12-21 | 2006-03-20 |    0.276 |    0.333
  3 | 2005-12-21 | 2006-03-20 |    0.436 |    0.452
  4 | 2005-12-21 | 2006-03-20 |    0.538 |    0.510
```

Automating the Intersection

The last step is now to automate the intersection of the GPS locations and the NDVI data set. The approach is similar to the automatic intersection with other environmental layers (e.g. elevation or land cover) described in Chap. 6; however, the dynamic nature of the NDVI time series makes it slightly more complex. In the case of near real-time monitoring, you will generally acquire GPS data before the NDVI rasters are available. As a consequence, two automated procedures are necessary to update the table *main.gps_data_animals*: one after the import of new GPS locations and one after the import of new NDVI data.

First of all, you need a new column in the *gps_data_animals* table to store the NDVI values:

```
ALTER TABLE main.gps_data_animals
ADD COLUMN ndvi_modis integer;
```

Now, let us first update this column manually for those locations that actually correspond to an NDVI tile in the database:

```
UPDATE
  main.gps_data_animals
SET
  ndvi_modis = ST_Value(rast, geom)
FROM
  env_data.ndvi_modis
WHERE ST_Intersects(geom, rast)
  AND acquisition_range @>
  acquisition_time::date
  AND gps_validity_code = 1 AND
  ndvi_modis IS NULL;
```

You can verify that the fields are updated:

```
SELECT
  gps_data_animals_id AS id, acquisition_time,
  ndvi_modis / 10000.0 AS ndvi
FROM
  main.gps_data_animals
WHERE
  geom IS NOT NULL
ORDER BY
  acquisition_time
LIMIT 10;
```

The result is the following:

```
  id   |     acquisition_time     | ndvi
-------+------------------------+-------
 39212 | 2005-03-20 11:03:14-05 | 0.447
 39214 | 2005-03-20 19:03:06-05 | 0.505
 39215 | 2005-03-20 23:01:45-05 | 0.505
 39217 | 2005-03-21 07:02:19-05 | 0.431
 39218 | 2005-03-21 11:01:12-05 | 0.431
 39219 | 2005-03-21 15:01:49-05 | 0.480
 39220 | 2005-03-21 19:01:24-05 | 0.480
 39221 | 2005-03-21 23:02:51-05 | 0.480
 39222 | 2005-03-22 03:03:04-05 | 0.541
 39223 | 2005-03-22 07:01:42-05 | 0.541
```

Now, the last, more complicated step, is to use triggers to automate the process. Exactly as in Chap. 6, you need to extend the trigger function *new_gps_data_animals* that will be automatically triggered every time you add new GPS locations to the database:

```
CREATE OR REPLACE FUNCTION tools.new_gps_data_animals()
RETURNS trigger AS
$BODY$
DECLARE
  thegeom geometry;
  thedate date;
BEGIN
IF NEW.longitude IS NOT NULL AND NEW.latitude IS NOT NULL THEN
  thegeom = ST_SetSRID(ST_MakePoint(NEW.longitude, NEW.latitude), 4326);
  thedate = NEW.acquisition_time::date;
  NEW.geom = thegeom;
  NEW.pro_com =
    (SELECT pro_com::integer
     FROM env_data.adm_boundaries
     WHERE ST_Intersects(geom, thegeom));
  NEW.corine_land_cover_code =
    (SELECT ST_Value(rast,ST_Transform(thegeom, 3035))
     FROM env_data.corine_land_cover
     WHERE ST_Intersects(ST_Transform(thegeom, 3035), rast));
  NEW.altitude_srtm =
    (SELECT ST_Value(rast, thegeom)
     FROM env_data.srtm_dem
     WHERE ST_Intersects(thegeom, rast));
  NEW.station_id =
    (SELECT station_id::integer
     FROM env_data.meteo_stations
     ORDER BY ST_Distance_Spheroid(thegeom, geom, 'SPHEROID["WGS 84",
     6378137,298.257223563]')
     LIMIT 1);
  NEW.roads_dist =
    (SELECT ST_Distance(thegeom::geography, geom::geography)::integer
     FROM env_data.roads
     ORDER BY ST_distance(thegeom::geography, geom::geography)
     LIMIT 1);
  NEW.ndvi_modis =
    (SELECT ST_Value(rast, thegeom)
     FROM env_data.ndvi_modis
     WHERE ST_Intersects(thegeom, rast)
     AND acquisition_range @> thedate);
  END IF;
RETURN NEW;
END;$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
COMMENT ON FUNCTION tools.new_gps_data_animals()
IS 'When called by the trigger insert_gps_positions (raised whenever a new
position is uploaded into gps_data_animals) this function gets the longitude
and latitude values and sets the geometry field accordingly, computing a set
of derived environmental information calculated intersecting or relating the
position with the environmental ancillary layers.';
```

However, note that the update process will be limited by the availability of NDVI data at the time of the GPS data import (NDVI data are generally available two weeks after the period considered, which might then be later than the GPS data import). In order to have a complete database, you thus also need to update the table when new NDVI data are added. You can do it by running the *UPDATE* query every time new images are imported or by creating another trigger function that will automatically update the GPS locations that correspond to the NDVI temporal range in *main.gps_data_animals*. Here is the trigger function:

```
CREATE OR REPLACE FUNCTION tools.ndvi_intersection_update()
RETURNS trigger AS
$BODY$
BEGIN
  UPDATE main.gps_data_animals
  SET ndvi_modis =
    (SELECT ST_Value(NEW.rast, geom)
      FROM env_data.ndvi_modis
      WHERE ST_Intersects(geom, NEW.rast)
      AND NEW.acquisition_range @> NEW.acquisition_time::date)
  WHERE ST_Intersects(geom, NEW.rast)
  AND NEW.acquisition_range @> acquisition_time::date
  AND ndvi.modis IS NULL;
END;
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
COMMENT ON FUNCTION tools.ndvi_intersection_update ()
IS 'When new NDVI data are added, the ndvi_modis field of
main.gps_data_animals is updated.';
```

The functions will be activated each time new data are loaded into *env_data.ndvi_modis*. The function *ndvi_intersection_update* will be activated after an NDVI tile is loaded, because we want the final version of the NDVI tiles before propagating the updates to other tables. Here is the trigger to achieve this:

```
CREATE TRIGGER update_ndvi_intersection
AFTER INSERT ON env_data.ndvi_modis
  FOR EACH ROW EXECUTE PROCEDURE tools.ndvi_intersection_update();
```

From now on, every time you collect NDVI data and feed them into the database using *raster2pgsql*, the magic will happen!

References

Bannari A, Morin D, Bonn F, Huete AR (1995) A review of vegetation indices. Remote Sens Rev 13:95–120

Basille M, Fortin D, Dussault C, Ouellet JP, Courtois R (2013) Ecologically based definition of seasons clarifies predator-prey interactions. Ecography 36:220–229

Cagnacci F, Boitani L, Powell RA, Boyce MS (2010) Animal ecology meets GPS-based radiotelemetry: a perfect storm of opportunities and challenges. Philos Trans R Soc B: Biol Sci 365:2157–2162

Cracknell AP (1997) The advanced very high resolution radiometer (AVHRR). Taylor & Francis, London

Eerens H, Haesen D, Rembold F, Urbano F, Tote C, Bydekerke L (2014) Image time series processing for agriculture monitoring. Environ Modell Softw 53:154–162

Escadafal R, Bohbot H, Mégier J (2001) Changes in arid mediterranean ecosystems on the long term through earth observation (CAMELEO). Final Report of EU contract IC18-CT97-0155, Edited by Space Applications Institute, JRC, Ispra, Italy

Frair JL, Fieberg J, Hebblewhite M, Cagnacci F, DeCesare NJ, Pedrotti L (2010) Resolving issues of imprecise and habitat biased locations in ecological analyses using GPS telemetry data. Philos Trans R Soc B: Biol Sci 365:2187–2200

Land Processes DAAC (2008) MODIS reprojection tool user's manual. USGS Earth Resources Observation and Science (EROS) Center

Lunetta RS, Knight JF, Ediriwickrema J, Lyon JG, Dorsey Worthy L (2006) Land-cover change detection using multi-temporal MODIS NDVI data. Remote Sens Environ 105:142–154

Maisongrande P, Duchemin B, Dedieu G (2004) VEGETATION/SPOT: an operational mission for the Earth monitoring; presentation of new standard products. Int J Remote Sens 25:9–14

Manly BFJ, McDonald LL, Thomas DL, McDonald TL, Erickson WP (2002) Resource selection by animals. Kluver Academic Publishers, Dordrecht

Maselli F, Barbati A, Chiesi M, Chirici G, Corona P (2006) Use of remotely sensed and ancillary data for estimating forest gross primary productivity in Italy. Remote Sens Environ 100:563–575

McClain CR (2009) A decade of satellite ocean color observations. Annu Rev Marine Sci 1:19–42

MODIS (1999) MODIS Vegetation Index (MOD 13): Algorithm Theoretical Basis Document Page 26 of 29 (version 3)

Moorcroft P (2012) Mechanistic approaches to understanding and predicting mammalian space use: recent advances, future directions. J Mammal 93:903–916

Moriondo M, Maselli F, Bindi M (2007) A simple model of regional wheat yield based on NDVI data. Eur J Agron 26:266–274

Nathan R, Getz WM, Revilla E, Holyoak M, Kadmon R, Saltz D, Smouse PE (2008) A movement ecology paradigm for unifying organismal movement research. Proc Natl Acad Sci 105:19052–19059

Pettorelli N, Gaillard JM, Mysterud A, Duncan P, Stenseth NC, Delorme D, Van Laere G, Toigo C, Klein F (2006) Using a proxy of plant productivity (NDVI) to find key periods for animal performance: the case of roe deer. Oikos 112:565–572

Townshend JRG, Justice CO (1986) Analysis of the dynamics of African vegetation using the normalized difference vegetation index. Int J Remote Sens 8:1189–1207

Turchin P (1998) Quantitative analysis of movement: measuring and modeling population redistribution in plants and animals. Sinauer Associates, Sunderland

Verhoef W, Menenti M, Azzali S (1996) A colour composite of NOAA-AVHRR–NDVI based on time series analysis (1981–1992). Int J Remote Sens 17:231–235

Yu XF, Zhuang DF (2006) Monitoring forest phenophases of Northeast China based on MODIS NDVI data. Resour Sci 28:111–117