LUTO II (neoLUTO)

Fjalar J de Haan*

Planet A, Centre for Integrative Ecology, Deakin University, Australia

Abstract

This document provides an overview of the LUTO II modelling software package.

Contents

*Corresponding author. Email: f.dehaan@deakin.edu.au, fjalar@fjalar.org

1 Introduction

This document provides an overview of the software package of LUTO II. It describes its architecture, the mathematical model behind the cost-minimising dynamics, the data required and how to build them from the raw datasets. It also gives instructions on how to set up and run the model.

The LUTO II model simulates land-use change. The model is spatially explicit and the modelled territory is discretised into 1×1 km^2 (approximately) grid cells. These grid cells are represented in the model as 1D arrays, with the array entry representing the land use. A *land use* is, as the name suggests, the way land is used, which typically means what kind of crop is grown or what type of livestock grazes on it. Associated with the land use are production costs and yields, both annual, which allow the model to decide which land use is optimal for a cell. There is a list of land uses considered and these are represented by integer values (e.g. 'Apples' = 1, 'Dairy' = 7). Thus, at the heart of it, land-use change is modelled by a changing integer array. This integer array is called the land-use map or *lumap* for short.

In addition to the land use, the model also takes into account *land management*. Apples, for example, can be cultivated with (or without) irrigation, using organic methods (or not) and so on as well as combinations of such land managements. Depending on the land management, there will typically be different production costs and different yields associated with the land use. Irrigation is more expensive but may be expected to increase yields. There is a list of land managements and these are, like land uses, represented by integer values. Conventional, dry-land agriculture is the default and represented by zero, while irrigation is represented by one. The model currently only considers conventional dry land and irrigation as land managements. Thus there is also a land-management map, or *lmmap* for short.

The *lumap* and *lmmap* are the dependent variables of the model. Everything else is calculated on the basis of these two 1D arrays. The mechanisms driving the dynamics are economic. In the original CSIRO LUTO, the economic rationale was profit maximisation, i.e. farmers were supposed to cultivate whatever would profit them most (subject to some risk-based thresholds). In LUTO II the economic rationale is more systemic. The idea is that the agricultural system tries to meet demands (inputs to the model) the best it can, subject to certain constraints, while trying to minimise the total cost of production. Thus, using a yearly time step, the model is fed new demands and the model's solver produces a new land-use map.

Minimising cost while meeting demand means the model is trading off the ex-

penditure of production against the yield of the crop or livestock. In addition to this, there are *transition costs* associated with switching from one land use to another. This means that if a grid cell changes from growing apples to raising cattle, there is not only the new production cost to be paid (which may be lower) but also a transition cost. These transition costs subsume various costs of switching (including infrastructural investments and changed irrigation costs). Transition costs introduce 'memory' into the model, avoiding that the land-use map changes unrealistically much at each time step as it attempts to meet demand at lowest cost.

LUTO II, like its progenitor, is an optimisation model. The economic rationale is formalised as a linear programme, which is then solved using external, commercial, black-box, closed-source solver software (GUROBI for LUTO II and CPLEX for LUTO I). The mathematical model of LUTO II is an actual linear programme but an alternative solver prototype using binary decision variables is also part of the package.

The model is solved under various constraints. One constraint ensures that all of every cell is in use. That is, there is always *some* land use on a cell and all of it is used. In principle, a cell can multiple 'fractional' land uses, though in practice, if the territory consists of many cells this seems not to occur. Another constraint is that the deviation of production from the demanded quantities should be minimal. This is a so-called soft constraint. These two constraints are an essential part of the model formulation. The remaining constraints are concerned with environmental targets and they can be switched on or off. Of these, water use is the first considered and the only one currently implemented. The water constraint demands that current water use relative to the water yield (how much runs of) in a river region should not exceed the water use in 2010 relative to pre-European yields (which are estimated using 1985 data and assuming deeply-rooted vegetation). Water is implemented as a hard constraint.

2 Architecture of the model

The LUTO II software is contained in a Python 3 package. The package structure (see Figure 1) is meant to reflect the different stages of the modelling process. The various sub packages live under a main package called `luto`. Thus there is a sub package concerned with the loading of data (`luto.data`), one with cost calculations (`luto.economics`) and so on. Each sub package typically contains several modules, each with several functions.

A small number of design principles has generally been adhered to, though not into the extreme. The most important were:

- Avoiding global variables and avoiding statefullness. The only exception to this being the `luto.solvers.simulation` module which has the express purpose of keeping the state of a simulation.
- Avoiding object orientation where possible. The notable exception is again `luto.solvers.simulation` which uses a `class Data`.
- Functions should be *pure*. This means that a function should return something, it should always return the same thing if given the same arguments and it should do nothing else ('no side effects'). Deviations from this principle are typically explicit (e.g. setters and functions to write to file).
- Dependencies should be few. Generally, the external dependencies are limited to Numpy and Pandas (almost everything uses these) and things like HDF5 libraries. Dependencies on other LUTO modules are used where unavoidable, e.g. in a module where many things come together. Again, in `luto.solvers.simulation` there are of course plenty of internal imports.

The user interface of the model is the `luto.solvers.simulation` module. Importing this module will implicitly load the `data` module. It will use the parameters set in the `luto.settings` module. At this point, those parameters are limited to directory locations and some settings relating to the water constraints. If all the requisite data is available where it should be (`input/` by default), this is sufficient to run the model. One can run 'interactively', that is, from the prompt of a Python interpreter or call the functions of the `luto.solvers.simulation` module from a run script. Preparing the input data and setting up the model to run is discussed in Sections 4 and 5, respectively.

2.1 Loading data

The main idea behind the `luto.data` module is to have a namespace for all input data and parameters. For example, importing the module, by issuing

```
import luto.data as data
```

will make the list of land uses available as `data.LANDUSES`. Data is made available in three ways, (1) by reading the data directly from a file, typically

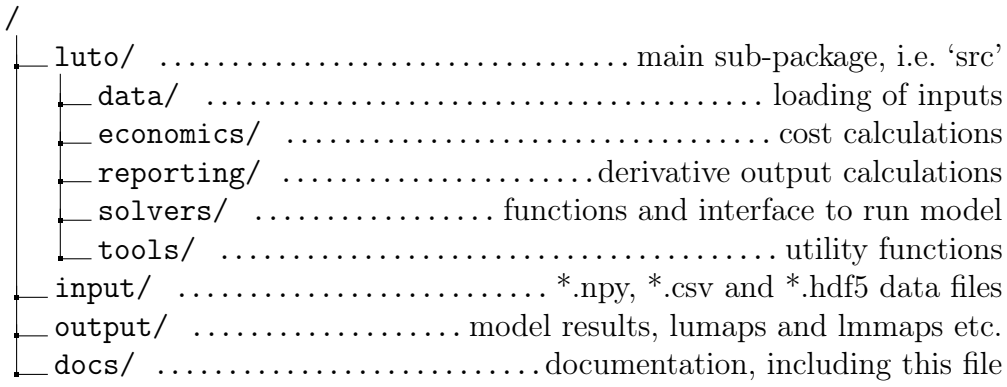

Figure 1: Directory structure of LUTO II Python package.

a *.npy, *.csv or *.hdf5 file, by (2) post-processing data read from file to some extent, or (3) by defining variables in the code directly (i.e. 'magic numbers'). This third option is obviously to be avoided and the only magic number is the starting year, 2010, which is added to the year index (zero-based) whenever an actual calendar year is required.

The files required by the `luto.data` are:

- ...

The data in the files is used to *infer* various parameters of the modelling problem. In other words, in principle, the code knows nothing about the particulars of the modelling problem under consideration. Things like the spatial extent (number of grid cells), the lists of land uses and land managements and so on, the model obtains from inspecting the data files. This has the obvious advantage that the model is not tied to a certain number of cells or a particular set of land uses.

There are complications to this picture. The most notable being the following. The list of *land uses* (whatever lives on the land) is not the same as, and not in a one-one relation with, the list of *products* those land uses yield. For example, 'Sheep - natural land' (a land use) yields 'SHEEP - NATURAL LAND MEAT' as well as 'SHEEP - NATURAL LAND WOOL'. To complicate matters further, the list of *commodities* that are demanded (the time series of which are a key input to the model) are not the same nor in a one-one relationshep with either the list of land uses or products. For example, the demand for wool is indifferent to whether the sheep grazed on natural or modified land. So, in the list of products there is just 'sheep wool'.

To avoid confusion and force useful errors, the three lists (`data.LANDUSES`,

`data.PRODUCTS`, `data.COMMODITIES`) use different cases (sentence, all upper and all lower case, respectively). Nevertheless, the lists are produced by inferring the list of land-uses from the agricultural data and a host of string manipulations. This is clearly error prone in light of future extensions. Moreover, the *conversions* from e.g. land-use to product representation, which are necessary to set up the linear programme in the solver, employ conversion matrices. These matrices are defined in the `luto.data` module following the above logic.

The situation with the land-uses, products and commodities make the `luto.data` module cluttered, hard to read and a potential source of future bugs. The schema can likely be simplified to just two lists and some processing could be factored out while other things can be turned into file reads. At the moment however, the module works and once loaded the complications are at any rate not so visible.

Since the functions for e.g. cost calculations, receive a `data` object as an argument

Since the model avoids global dependencies, the `luto.data` loaded module needs to be passed as an argument to all functions that need it.

2.2 Calculating costs

2.2.1 Production costs

2.2.2 Transition costs

2.3 Solving and running

3 Optimisation mathematics

4 Data requirements and preparation

5 Running the model