Readme file for PyWofost version 7.1-3.0

Allard de Wit

7 November 2009

Contents

1 New features in PyWofost 3.0

- *Internals related to the FORTRAN code have been completely revised.* A python wrapper has been created that wraps each call to the underlying WOFOST

FORTRAN routines and hides much of the ugly details from the general PyWofost processing loop. This has much improved the readability of the code.

- *Automatic testing of PyWofost output.* Automatic testing has been implemented using the python unittesting framework. The revision of the PyWofost internals would not have been easy without an automated procedure that checks whether the result of the revised PyWofost simulations were still consistent with the old version. Reference data for unittesting resides in the database table `pywofost_unittest_benchmarks`.

- *Information about WOFOST variables has now been formalised.* A new PyWofost module `w60_variable_description.py` which formalises variables names and provides various functions that allow to retrieve information about variables such as long and abbreviated variable names (e.g. leaf_area_index vs. lai), the units of a particular variable and a check if a given variable name is actually a valid PyWofost variable. See table 1 for an overview of all PyWofost variables.

- *Generalised interface for getting/setting PyWofost variables.* A PyWofost class now has a method `get_variable(variable_name)` which allows to retrieve the value from a PyWofost instance of any variable that is known to PyWofost. Similarly, a method `set_variable(variable_name, value)` has been implemented that allows to change the value of a particular variable in PyWofost. However, because the internal logic for setting the value of a PyWofost variable is far more complex then for getting a value, the `set_variable()` method is supported only for the variables 'leaf_area_index' and 'profile_moisture_content'. This can be extended in future versions.

- *Simplified PyWofost output customisation.* The improved way of dealing with variable names in combination with the strength of SQLAlchemy has made it possible to greatly simplify the way PyWofost output can be customised. In previous versions, the columns in the database table `pywofost_output` were fixed. In PyWofost 3.0 you can simply add or remove output variables by adding or removing columns from this table. This works both for long and abbreviated variable names. Thus, adding a column `dry_matter_increase` will contain the values of dry matter increase after running PyWofost.

- *Plotting module for time-series plots added.* A plotting module has been added that can create time-series plots of (ensemble) PyWofost simulation results and supports various graphics formats. The module has a simple command line interface for specifying variables, setting graphics format, output filename, etc.

2 Introduction

This package contains the 3.0 version of the WOFOST 7.1 crop growth model linked to the python interpreter, called PyWofost 7.1-3.0. The development of this implementation of the WOFOST model was done in the framework of my PhD research. During my PhD work I ran into the limits of the currently existing WOFOST implementations: neither the WOFOST 7.1 version in FORTRAN nor the version of WOFOST that was implementen in the Crop Growth Monitoring System (CGMS) was suitable for my needs. The FORTRAN version was a standalone version which was good for detailed analyses, but lacked the ability to communicate with databases which is necessary for storing results from batch processing. The C++ version was inflexible and targeted at operational implementation of the system.

Not being very happy with this situation, I decided that a new, more flexible, version of WOFOST was needed. This new version of WOFOST should make use of modern programming concepts like object oriented design, however a full rewrite of the model was not attractive because of the time involved. Therefore I decided to take the WOFOST kernel routines out of the WOFOST 7.1 model and link them into the python interpreter. Using this approach, the old routines could be implemented in a modern programming framework. Moreover the rich set of extension modules that are part of python (or can be added) allowed many improvements, such as the handling of dates and a largely database independent implementation of the model I/O. Finally, the use of a scripting language like python means that PyWofost is very flexible and can be adapted for specific situations quit easily.

Nevertheless, this flexibility comes at a price: there is currently no user interface for PyWofost and for running PyWofost you will need to write the appropriate 'wrapper' scripts in order to carry out the simulations that you want. An example for how PyWofost can be run is actually integrated in this package. Also, remember that PyWofost is a tool designed for scientific purposes and was therefore not designed to be an 'end user' application.

3 Hardware and software requirements

PyWofost is not a heavy model and it should be possible to run PyWofost on any modern computing architecture, probably without ever reaching the limits of your machine. PyWofost is known to run on Linux (Mandrake 9.2, Fedora 6, Ubuntu 8.04, Redhat EL4), Windows XP and Mac OS X (10.4.11), but it should not be too difficult to run PyWofost on any architecture as long as these packages are available:

- Python $\geq$ 2.4.6 (2.5 recommended, 3.0 is not supported yet.)
- SQLAlchemy $\geq$ 0.4.7 (not tested on 0.5 or 0.6 releases)

- Database connectors for the database of your choice (not necessary for SQLite)
- NumPy $\geq$ 1.1
- A FORTRAN compiler supported by NumPy/F2PY: GNU gfortran (GCC4.0) on Linux, GNU gfortran 4.2.0 (GCC4.0) on MacOSX and MinGW 5.1.4 (g77) on Windows XP were used by the author.

4 Content of the pywofost package and installation details

PyWofost is currently not provide as a python package so the usual `python setup install` will not work. Instead, some manual work is involved. First of all unzip the package in the location where you want to install PyWofost. The following directories will be created:

- *wofost:* Contains the core python routine `pywofost.py`, the script for running unittests '`pywofost_unittest.py`' and some example scripts:
  `test_pywofost.py` for starting single PyWofost runs;
  `test_pywofostensemble.py` for making an ensemble run;
  `test_pywofost_enkf.py` for making a run with the ensemble Kalman filter.
- *database:* The SQLite PyWofost demo database and the create scripts for setting up the database on other database systems;
- *database_comm:* Python routines used by PyWofost for deriving input from the database;
- *doc:* This readme file;
- *enkf:* Python routines for running PyWofost in ensemble and EnKF mode;
- *libttutil:* the TTUTIL FORTRAN library used by the WOFOST FORTRAN routines;
- *libw60:* ancillary FORTRAN routines used by the WOFOST FORTRAN routines;
- *plotting:* The tool for making plots of PyWofost output.
- *pyfortran:* the WOFOST 7.1 kernel routines in FORTRAN 77 that are linked into python, the wrapper routines and the module that describes the variables in PyWofost;

- *w60_util:* The timer routine that PyWofost uses as well as some ancillary python routines (date conversion, etc.).
- *taskmanager:* The task manager which is used for distributed computing.

5 Compiling PyWofost yourself

While python itself is essentially platform independent, PyWofost reuses part of the FORTRAN code of Wofost 7.1 which reside in a shared library. PyWofost ships with shared libraries for Windows XP, Linux and MacOS X, but depending on your specific version of python, numpy, FORTRAN and C-compilers these shared libraries may fail to load and you may need to recompile them for your platform.

The necessary scripts for compiling are provided in the PyWofost distribution. It is assumed here that the GNU compiler suite has been installed which is usually the case on linux. On MacOSX you can use MacPorts to install the GNU compilers while on Windows MinGW can be downloaded and installed. Carry out the following steps:

- Change directory to libttutil and execute `make_ttutil_<platform>` where <platform> is either 'linux', 'osx' or 'win32'. A file 'libttutil.a' should have been created.
- Change directory to libw60 and execute `make_w60lib_<platform>`, a file 'libw60.a' should have been created.
- Change directory to pyfortran and execute `make_py_<platform>` a file 'py_w60lib_<platform>.so' or a file 'py_w60lib_win32.pyd' should have been created. Now start python and import py_w60lib_<your platform> into python using the normal importing procedures. This should give no error messages and you should be able to print the generated module documentation string through py_w60lib_<platform>.__doc__

Note:

- Be not surprised when these scripts fail, this is probably because names of compilers differ on different systems or paths are not set properly. Try to find out the name of compilers and paths on your system and edit the scripts accordingly.
- Interface generation relies on the `f2py` utility included with numpy. In order to check that `f2py` works correctly you can use the command `f2py -c --help-fcompiler` which will list all FORTRAN compilers that `f2py` was able to recognize and find on your system.

6 Setting up the PyWofost database

For testing PyWofost you can use SQLite which is a standard module in python 2.5 and available as an add-on package for python 2.4. In this case you only need to copy the SQLite database file `pywofost_demo.db` to a file `pywofost.db` in the folder `wofost/` and you're done.

However, for serious modelling with ensembles, SQLite storage capabilities will quickly be insufficient as SQLite databases are limited to 2Gb of storage. PyWofost contains some tools for migrating the PyWofost database to another database system, all databases supported by python and SQLAlchemy can be used but testing has mainly been done with MySQL. For migrating the PyWofost demo DB execute the following steps:

1. Create a database on your DBMS of choice and assign a user with sufficient rights for your database.

2. Change directory to the `database` directory edit the connection string in the `create_engine()` command in the file `create_pywofost_demo_db.py` in order to have the apropriate connection string for your database. Some examples are included, see the SQLAlchemy documentation for more information.

3. Execute the python script `create_pywofost_demo_db.py` in the python interpreter and the script will start building the demo database. Note that this can take a while because some of the database tables contain a substantial amount of records.

4. When the database is ready, move to the `wofost/` directory and edit the wrapper scripts `test_pywofost.py`, `test_pywofostensemble.py` and `test_pywofost_enkf.py`. Make sure that the same connection string is used in the `create_engine()` command.

Note:

- All table names and column names have been defined in lowercase, depending on your system, databases sometimes make a difference between upper and lowercase for tables and column names.

7 Running PyWofost

PyWofost can be run by by executing the python scripts `test_pywofost.py`, `test_pywofostensemble.py` and `test_pywofost_enkf.py` in the `wofost` directory.

The first scripts runs the model for 6 crops for the year 2000 at a location in southern Spain. Model simulation results will be written to the table `pywofost_output` and to several output files, each filename has the format 'GRID_YEAR_CROP_MODE.out'. Depending on what is convenient, the results from files or database can be imported to other programs for analysis.

The second and third scripts run over the same crops but starts an ensemble of 50 PyWofost models, additionally the third script uses the built-in ensemble Kalman filter to assimilate observations of rootzone soil moisture. These two scripts use the taskmanager and read the tasks from the table `tasklist` in the database. Output is also written to a table `pywofost_output` as well as one large file `pywofost_output.csv`.

8 Relationship with the Crop Growth Monitoring System (CGMS)

The database structure used by PyWofost is partially a copy of the CGMS[1] database structure. The following tables are exact copies of their CGMS equivalents: `crop`, `crop_parameter_value`, `grid`, `grid_weather` and `variety_parameter_value`. The table `crop_calendar` was modified in the sense that grid_no, crop_no and year now uniquely identify a PyWofost run. Moreover the start_day and end_day are defined as date, making it more easy to define when PyWofost runs cross a year boundary (e.g. the campaign for winter-wheat harvested in 2000 starts in October 1999 in Western-Europe).

The tables `site`, `soil_physical_group`, `soil_type` and `rooting_depth` have been adapted, in fact simplified. The main simplification that has been carried out is that PyWofost allows only one soil type per grid. Also, some column names have been adapted to make them more self explanatory (e.g INITIAL_WATER_AVAILABILITY instead of WAV).

9 Acknowledgements

The development of the WOFOST model was already started in the late eighties in Wageningen and many people have contributed to its development over the years. Just to name a few: Kees van Diepen, Joost Wolf, Herman van Keulen, Daniel van Kraalingen, Hendrik Boogaard and Tamme van der Wal.

[1] http://mars.jrc.it/mars/About-us/AGRI4CAST/Crop-yield-forecast/The-Crop-Growth-Monitoring-System-CGMS

For a publication describing WOFOST see:

- Diepen, C.A. van, Wolf, J. and Keulen, H. van (1989). WOFOST: a simulation model of crop production. *Soil Use and Management* **5**: 16–24.

For more general information over the Wageningen crop models see:

- Bouman, B., Keulen, H. van, Laar, H. van and Rabbinge R. (1996). The 'School of de Wit' crop growth models: a pedigree and historical overview, *Agricultural Systems* **52**: 171–198.
- Van Ittersum, M.K., Leffelaar, P.A., Keulen, H. van, Kropff, M.J., Bastiaans, L. and Goudriaan, J. (2003). On approaches and applications of the Wageningen crop models, *European Journal of Agronomy* **18**: 201–234.

For a publication describing the application of PyWofost see:

- Wit, A.J.W. de and Diepen, C.A. van (2007). Crop model data assimilation with the ensemble kalman filter for improving regional crop yield forecasts. *Agricultural and Forest Meteorology* **146**: 38–56. **doi:**10.1016/j.agrformet.2007.05.004

10 License

The whole PyWofost package is distributed under the terms of the GNU Public license v3. See http://www.gnu.org/licenses/gpl.html. I do request that users of PyWofost send any improvements/bug fixes back to me so they can be included in the next version of PyWofost.

11 Contact

You can contact me for questions, bug reports or other problems at:
Allard de Wit
Centre for Geo-information
Alterra – Wageningen UR
P.O. Box 47
6700AA Wageningen - The Netherlands
E: allard.dewit@wur.nl
T: +31-317-481914

12 Variables in PyWofost

Table 1: Overview of PyWofost variables.

Short name	*Long name*	*Unit*
admi	aboveground_dry_matter_increase	kg ha-1 day-1
danth	day_of_anthesis	-
dmi	dry_matter_increase	kg ha-1 day-1
drlv	death_rate_leaves	kg ha-1 day-1
drrt	death_rate_roots	kg ha-1 day-1
drst	death_rate_stems	kg ha-1 day-1
dtsum	temperature_sum_increase	degree
dvr	development_rate	day-1
dvs	development_stage	-
dwlv	weight_dead_leaves	kg ha-1
dwrt	weight_dead_roots	kg ha-1
dwso	weight_dead_storage_organs	kg ha-1
dwst	weight_dead_stems	kg ha-1
gass	gross_assimilation_rate	kg CH2O day-1
gasst	total_gross_assimilation	kg CH2O
grlv	growth_rate_leaves	kg ha-1 day-1
grrt	growth_rate_roots	kg ha-1 day-1
grst	growth_rate_stems	kg ha-1 day-1
gwrt	net_growth_roots	kg ha-1 day-1
gwso	growth_rate_storage_organs	kg ha-1 day-1
gwst	net_growth_stems	kg ha-1 day-1
idwst	total_water_stress_days	-
lai	leaf_area_index	-
mres	maintenance_respiration	kg CH2O day-1
mrest	total_maintainance_respiration	kg CH2O
pgass	pot_gross_assimilation_rate	kg CH2O day-1
pmc	profile_moisture_content	-
ptra	potential_transpiration	cm
rd	rooting_depth	cm
rmc	rootzone_moisture_content	-
tadw	total_aboveground_living_biomass	kg ha-1
tagp	total_aboveground_biomass	kg ha-1
tra	transpiration	cm
trat	total_transpiration	cm
tsum	temperature_sum	degree day
twl	total_weight_leaves	kg ha-1
twr	total_weight_roots	kg ha-1
tws	total_weight_stems	kg ha-1
twso	total_weight_storage_organs	kg ha-1
w	rootzone_water_depth	cm
wlow	subsoil_water_depth	cm
wlv	weight_living_leaves	kg ha-1
wrt	weight_living_roots	kg ha-1
wso	weight_living_storage_organs	kg ha-1
wst	weight_living_stems	kg ha-1
wwlow	profile_water_depth	cm