Read Me Richards Module PCR-GlobWB2.0

Bright Minds Project by Joren Janzing
g.w.janzing@uu.nl
Version: 03-07-2020

Content

General

In this document, the modules that I made for the Bright Minds Project and that could be added to the PCRGlobWB2.0 hydrological model are explained. The work was done under supervision of Niko Wanders at the Physical Geography Department of Utrecht University. I will focus on the main structure and provide an overview of the assumptions that were made.

New modules which were added and which will be discussed in this document include:

- **Unsaturated Zone: Richards Equation**
  Solves the Richards Equation for the transport of water in the unsaturated zone.
- **Unsaturated Zone: Heat Function**
  Uses advection and diffusion to compute the soil temperature evolution in the unsaturated zone.
- **Surface: Energy Balance**
  Computes the surface energy balance and uses it to determine the skin temperature, which is used as an input for the heat function.

Files which I have adapted include `setup_30min_seperatefile.ini`, `landCover.py`, `landSurface.py`, `reporting.py`, `variable_list.py` and `meteo.py`. Note that all parts that are added are bounded by `#%% ADDED BY JOREN: START` and `#%% ADDED BY JOREN: STOP`.

The main file which has been added is `richards.py`, which includes all relevant functions.

Updates of Pre-Existing Files

setup_30min_seperatefile.ini

In this file the input for the model is provided. This part focusses on the changes with respect to the normal *.ini*-file.

Under `[landSurfaceOptions]`, there are new entries:

- `includeRichards = True`
  Boolean: True if you want to include the Richards model (so far only possible if number of soil layers is 2). Model should still be able to run normally if this is not included or set to `False`.
- `layerFactorRichards = 2`
  Integer: number of layers into which the upper and lower soil layer of PCR-GlobWB should be divided.
- `layerBoundariesRichards = [0, 0.10, 0.30, 0.40, 1.0]`
  Array [m]; Start with 0 (surface), only one layer between 0.13-0.30m and one between 0.65-1.5m. These ranges are the ranges of thicknesses of the 2 layer unsaturated soil in PCR-GlobWB and make sure that it is easy to switch between the two layer main model and my flexible layer model. **Not used if `layerFactorRichards` is not 0!**
- `numberOfCoresRichards = 8`
  Integer: Number of cores you want to run the model on. The Richards model is quite slow and in order to have faster computation times, more cores are recommended. However, 1 core is also possible.
- `timestepRichards = 6`
  Integer [hours]: By default, the surface energy balance is computed every 3 hours. The Richards model can be computed in 3,6,12 or 24 hours, in order to increase or decrease the temporal resolution if desired.

Furthermore, under `[meteoOptions]` new paths for netCDF-files to read are added:

- `cloudcoverNC = ../cru_ts3.21.cld.dat.nc`
  netCDF file containing cloudcover fractions over the basin.
- `radiationNC = ../short_wave_radiation_monthly_climatology_30arcmin.nc`
  netCDF file containing incoming short wave radiation over the basin.
- `vaporNC = ../cru_ts3.21.vap.dat.nc`
  netCDF file containing near surface vapor pressure over the basin.
- `sunhoursTable = ../sunhoursfrac.tbl`
  Table containing the sunhours fractions.

Note that the model should still be able to run if these are not provided, as long as `includeRichards = False` or not mentioned in the file.

meteo.py

In this file the meteorological input for the model is provided. This part focusses on the changes with respect to the normal `meteo.py`.

- In the `__init__` function, lines are added to read relevant meteo iniItems from the *ini*-file.
- The `readExtensiveMeteo()` function is included. This reads the relevant extra meteorological input for the correct timestep.

landCover.py

In this file the landsurface input for the model is provided. This part focusses on the changes with respect to the normal `landCover.py`.

- At the top, the `richards.py` file is imported as `richards`.

- In the `__init__` function, lines are added to read relevant land surface iniItems from the *ini*-file.\

- In the `getICsLC`-functions, `initialiseRichards()` is called to initialise the model.
- `updateSoilStates()`:
  - The function definition is changes: `(self, *args)`. This is done to include the meteo object and to include the current time step.
  - If number of soil layers is 2 and the `includeRichard = True`, the `runRichards()` is called.

landSurface.py

This part focusses on the changes with respect to the normal `landSurface.py`.

- In the `__init__` function, lines are added for reporting new variables as netCDF files.

variable_list.py

- Extra variables are created for reporting. These new variables are:
  - `soilTempUpp`: soil temperature of the upper cell in the main model.
  - `soilTempLow`: soil temperature of the lower cell in the main model.
  - `tempDeficit_6AM`: difference between skin and air temperature at 6AM.
  - `tempDeficit_6PM`: difference between skin and air temperature at 6PM.
  - `netSW`: net short wave radiation.
  - `netRad`: net incoming radiation (both short and long wave).
  - `longWaveRad`: outgoing long wave radiation.
  - `latentHF`: latent heat flux.
  - `sensibleHF`: sensible heat flux.
  - `groundHF`: ground heat flux.

reporting.py

- In `basic_post_processing()`, new variables are added for postprocessing.

The New `richards.py` File

General

`richard.py` is mainly built in numpy and not in pcraster as the rest of the code. For the most important variables, the reader is referred to the variable list at the end of this document. `richard.py` consists of multiple seperate functions. The main ones are:

- `initialiseRichards()`
  Defines all relevant variables which are used in the rest of the model.
  It starts by converting all relevant variables from pcraster to numpy and flatten them, while remembering the original order.\
  - In the section *MODEL DIMENSIONS*, the spatial and temporal resolution is defined. In general, units are [m] and [day] if not specified. Note that index [0] means the lower most layer and index [-1] is the top layer.
  - In the section *SOIL CONDITIONS*, the relevant soil conditions are defined. It is tried to retrieve most from the PCRGlobWB model, but for variables not defined, conditions are assumed. Van Genuchten parameters are assumed in the function `GuelphLoamDrying()`. Saturated and residual soil moisture content is taken from the model.

    - In the section *VEGETATION TYPES*, vegetation types can be defined. For now, only the albedo is relevant.
    - In the section *CONSTANTS*, the relevant constants are defined. The saturated and dry thermal conductivity are now the same for every cell, but this should be updated in a later version.
    - In the section *INITIAL VALUES AND BOUNDARY CONDITIONS*, the initial conditions for the soil temperature, richards equation and energy balance are defined.

- `runRichards()`
  Runs the actual model.

    - In the section *CONVERSION*, the relevant input is converted to numpy.
    - *INITIALISATION*: Data is flattened, empty arrays are defined. Some cells in the model have nan values, since they might fall outside the basin. The model does not work if these are incorporated. Therefore, `nanindexUpp` is defined to eliminated these. However, the main model still uses these cells and therefore `nanindex` is used to insert nanvalues in these cells in the *POSTPROCESSING* section.
    - *RUN THE MODEL*: The actual model is run and it exists of two loops: one for the energy balance (predefined to run in steps of 3 hours) and one for the Richards model in larger timesteps. More details on how the energy balance, richards model and temperature equations work in their specific sections.
    - *POSTPROCESSING*: The water balance is closed by computing the outgoing percolation. Note that it is first negative, since that is how the model is defined, but later made positive for the PCR-GlobWB model. Futhermore, NaN-values are inserted, the arrays are shaped into the proper dimensions and sizes. In the end, relevant variables are coupled to the landCover object and send to the main model.

The smaller functions will be explained in the sections on the Richards model, heat function and energy balance.

Wishlist:

- Make the model usable for the 3 layer configuration.
- Make the multiple layers completely flexible (so that the boundaries of the cells do not have to be included in the *ini*-file).

Unsaturated Zone: Richards Model

This model solves the Richards equation. It was based on a model on GitHub which can be found here.

In `runRichards()`, the main model is the `RichardsModel_dz2()`, which is solved with `scipy.odeint()`. In the case that multiple cores are provided in the *ini*-file, this is done by calling `solve()` per cell by using functions from the `multiprocessing` library. If only one core is provided, `scipy.odeint()` is called directly for all cells at the same time (this is very slow). `scipy.odeint()` needs high allowed maximum number of steps in order to stabilize the model (now set on `mxstep=3500`).

`RichardsModel_dz2()` uses the matrix potential `psi0` at time `t[0]` (in the function called `psi`) to compute the fluxes `q`. Note that the z-axis is positive upwards (**all downward fluxes are negative**). There convergence is then used to provide the time derivative of `psi` to compute the following timestep. As boundary conditions, potentials or fluxes at the top (`psiTop` and `qTop`) or at the bottom (`psiBot` and `qBot`). The model now uses `qTop` as infiltration and has free drainage at the bottom. The model uses the Van Genuchten parameters to compute the hydraulic conductivity (`K` from `KFun1()`) and dθ/dψ (`C` from `CFun()`). Hydraulic conductivity values, and therefore fluxes are averaged between two adjacent cells to compute the correct flux between them. Furthermore, no flow is allowed if the soil temperature is below 0 degrees Kelvin, so then the hydraulic conductivity is set to 0 for that layer to assume freezing.

In the end the new matrix potential `psi` is returned to `runRichards()` for time `t[-1]` . With `thetaFun()` , this can be converted to water content.

`fluxModel2()` is basically the same model as `RichardsModel_dz2()` , but only computes and returns the fluxes at the initial model. Note that we therefore only have the fluxes at the beginning and end of the run. Therefore, we do not exactly know how much water has left the soil, and this can only be assessed at the end by assuming the water balance to be closed per definition and computing how much has left the model.

Main Assumptions:

- Water Balance is closed by definition. Therefore, all water that is not in any other variable, is put in percolation.
- Psi can be larger than 0. However, if this happens and the is returned to a water content, the rest of the water is converted to either interflow (bottom layer) or saturation excess (top layer) after `RichardsModel_dz2` is run. Due to this the timestep has some influence on the amount of interflow and saturation excess from the model.
- No flow if soil is frozen.
- There is no interaction between the unsaturated zones of neighbouring cells.

Wishlist:

- If the surface is flooded, `psiTop` could be changed automatically to increase infiltration (instead of only having rain).
- Model is quite slow. It should be faster in a future version.

Unsaturated Zone: Heat Function

The main function for this model is `heatFun3()` . It solves for the soil temperature and also uses `scipy.odeint()` to compute the new state. However, since it stabilises much faster than `RichardsModel_dz2` , there is no need for multiprocessing and the maximum allowed number of steps can be low (now set to 1000, but can be lower).

Based on the fraction of water in the soil the thermal conductivity (`ThermalConduct()`) and heat capacity (`heatcapacity1()`)is computed for the soil layers. Note that it is assumed that the soil part consists of dry sand for the heat capacity. Furthermore, if it freezes the heat capacity and thermal conductivity of ice is used for the water fraction, which is fixed as of now.

Main Assumptions:

- Top layer has the skin temperature. It does not change its temperature.
- Bottom layer temperature is fixed in the calculation. However, at the end, it receives the same dTdt as the layer just above it, in order to change through time.

- Advective fluxes are computed from the state at the end of the previous run of `RichardsModel_dz2`. However, the real fluxes deviate probably a small amount from this.

Wishlist:

- Temperature should be influenced by the energy released from freezing or melting.
- Bottom layer temperature should change by itself.
- Small correction could be done to the moisture content if it freezes, since ice has a different density. This could have some influence on the conductivity.

Surface: Energy Balance

The surface energy balance is closed by means of the Newton-Raphson method. For this, `scipy.optimize.newton()` is used and the functions `radiation_f2()` (computes how much energy is unacounted for in the energy balance) and the `radiation_deriv2()` (computes the derivative with respect to the skin temperatures). This is very fast, so no multiprocessing is needed (now `mxiter=150`).

The actual fluxes are computed in the function `ComputeFluxes2()`. The latent heat flux is taken from the evapotranspiration computed by the original PCR GlobWB model itself and the incoming short wave radiation from the input netCDF files. Longwave radiation (`longWave_out2()`: based on cloudfraction and relative humidity), the sensible heatflux (`computeH()`) and the ground heatflux are computed (`GFlux()`).

Fluxes are positive away from the surface. Only the incoming short wave radiation is positive towards the surface.

Assumptions

- There is no melting or snow.
- Since latent heat is taken from the PCR GlobWB model, there is no dependence on the other variables of the energy balance.
- `GFlux()` uses the full depth of the first layer to compare the skin temperature against (based on equations in the literature and fitted to the Cabauw data). It further assumes that the heatflux at that depth is neglible over the times step (dG/dz=(G(top)-0)/dz)
- The aerodynamical resistance `ra` in `computeH()` taken as constant.
- A fixed daily cycle is assumed: `Tair_list` assumes deviations from the mean temperature during the day and `SW_fractions` devides the incoming solar radiation over the cells.
- Incoming shortwave radiation is not affected by cloud cover.

Wishlist:

- `ComputeFluxes2()` is now called double. It is very fast, so it does not take time but this could be made more efficient in a future model.
- Energy used for melting should be included in the energy balance.
- Snow layer albedo should be coupled in the energy balance. Now, always the albedo of grass is used.
- The energy balance could be decoupled from the Richards equation, so it can also be used in the normal model.
- Make the aerodynamical resistance `ra` a function of soil cover and wind.

Variable list

List of the most important variables.

Variable Name	**Unit**	**Description**
initialiseRichards.py()		
self.basin_shape		Tuple: the dimension of the basin in PCR GlobWB2.0.
self.nCells		Integer:The number of cells in the original model.
dz	[m]	Array: containing the thickness of the layers in the model.
self.layerfractions	[m]	Array: to shift between the two layer model and the arbitrary layer Richards model.
self.layerFactorRichards		Integer: Number of layers in which the bottom layer is divided (the top layer as well if activated in *ini*-file).
n		Tuple: the number of layers (`len(dz)`) in the vertical and the number of cells (`self.nCells`).
dt	[hours]	Integer: number of hours between runs of the `RichardsModel_dz2()`
t	[days]	Array: times for which the `RichardsModel_dz2()` is evaluated in `scipy.odeint()`.
timeunit	[s]	Integer: number of seconds per timestep of the `RichardsModel_dz2()`.
p		Dictionary containing soil characteristics (Van Genuchten Parameters).
thetaS		Array: saturated moisture content.
thetaR		Array: residual moisture content.
theta_soil		Array: fraction of soil: 1-thetaS.
alpha		Array: surface albedo.
dT	[K]	Integer: temperature step; needed to compute the derivative in `radiation_deriv2()`.
self.soiltemperature	[K]	Array: soil temperature.
qBot	[m/day]	Array: fixed bottom drainage. For free drainage, set it to `None`
psiBot	[m]	Array: fixed bottom matrix potential. For free drainage, set it to `None`
psiTop	[m]	Array: fixed upper matrix potential. For fixed infiltration, set it to `None`
self.storTotRich	[m]	Array: water content (without residual water!)

		for all layers in the Richards Model.
dt_short	[hours]	Integer: number of hours between runs of the energy balance
t_short	[days]	Array: times for which the energy balance is evaluated in scipy.optimize.newton().
timeunit_short	[s]	Integer: number of seconds per timestep of the energy balance.
SW_fractions	[]	Array: fractions of incoming short wave radiation. Assumed daily cycle.
Tair_list	[K]	Array: deviations of air temperature from the mean. Assumed daily cycle.
runRichards.py()		
cloudCover		Array: contains the cloudcover as a fraction.
vaporPressure	[kPa]	Array: actual vapor pressure.
S0	[W/m^2]	Array: incoming short wave radiation.
Tair	[K]	Array: air temperature.
nanindexUpp		Array: indices where the storage array is NaN in the upper layer.
nanindexLow		Array: indices where the storage array is NaN in the lower layer.
nanindex		Array: indices where to insert nanvalues later (is different that nanindexUpp).
n1		Tuple: same as n but corrected for NaN values.
ra		Integer: aerodynamic resistance.
RH		Array: relative humidity.
plantevap	[m]	Array: total evaporation per layer.
ETpm	[m]	Array: total evaporation per cell.
theta0		Array: initial water content per day.
excess	[m]	Array: saturation excess and interflow together (from oversaturated layers).
oversaturated		Array: indices of oversaturated layers.
qTop	[m/day]	Array: infiltrating water.
Ts	[K]	Array: skin temperature.
Rn	[J/m^2]	Array: net radiation.
H	[J/m^2]	Array: sensible heat.

G	[J/m^2]	Array: ground heat.
Sn	[J/m^2]	Array: short wave radiation.
Ln	[J/m^2]	Array: long wave radiation.
LE	[J/m^2]	Array: latent heat.
S0_3H	[J/m^2]	Array: as S0 but with daily cycle.
Tair_3H	[K]	Array: as Tair but with daily cycle.
ETpm_3H	[m]	Array: as ETpm but with daily cycle.
T0	[K]	Array: intial soil temperature.
theta_run		Array: soil moisture for during the computation.
psi0	[m]	Array: matrix potential for during the computation.
theta_top_new	[m]	Array: soil moisture content for during the computation.
q0	[m/day]	Array: flux from psi0.
q	[m/day]	Array: flux from psi.
_mp		Multiprocessing
psi	[m]	Array: matrix potential for during the computation.
tempDeficit_6AM	[K]	Array: Difference between the skin temperature and the air temperature at 6AM.
tempDeficit_6PM	[K]	Array: Difference between the skin temperature and the air temperature at 6PM.
percTot	[m/day]	Array: total percolation per cell
percUpp	[m/day]	Array: percolation for upper cell (in PCR GlobWB)
percLow	[m/day]	Array: percolation for lower cell (in PCR GlobWB)
storTot_new	[m]	Array: the new self.storTotRich translated to the 2 cells of PCR GlobWB

Contact

If anything is unclear and questions remain, please contact me by email: g.w.janzing@uu.nl. Furthermore, if desired, multiple older versions of the code can be send as well (e.g. for Raam network data or Cabauw data in the Netherlands).