# Changelog 10.03.2015, Jules

**Problem I**: setupGUI can’t load shapefile, because it’s of type PolygonZ.

* Solution: Convert shapefile from 3D to 2D, for example wirh GRASS GIS. Just import the shapefile and then export it again with the option “3D output (…)” unchecked.

**Problem II**: getRaw and processRaw terminate unexpectedly.

* Solution: ndays=0. Set it to 1

**Problem III**: Wrong output of getSubIndex leads to an error while running processRaw. (Only with the old version, where no parfor-loop is implemented.)

* Solution: The buffer matrix A must be cleared after processing a subcatchment (one lap of the outer loop): clear A. Otherwise values of the preceeding subchatchment are retained in the next smaller subcatchment.

*Code changes: getSubIndex.mat, line 97*

**Problem IV**: Long runtime of function adj2E(…).

* Description: for the adjustment of temperature and precipitation, the nearest neighbor of each cell above the threshold value must be found. The algorithm itself goes through every cell of the matrix, what causes long cpu-time for larger raster data.
* Solution Part 1: Because the cells above the threshold and their nearest neighbor never change (DEM is constant, georeference of rasterdata is constant), calculating the nearest neighbor must only be done once. The result will be saved in the file iCiE\_datatype.mat (e.g. iCiE\_GDAS.mat) in the sub folder ./data/processed/sub/. Every subsequent run of processRaw will then load this file instead of calculating it again.

*Code changes: split adj2E into findNeighbor.m and adjmeteodata.m*

*processRaw.m becomes processRaw\_fast.m, line 94-106 ; 284-297*

* Solution Part 2: Speed up the function adj2E by parallelization. Some changes in the code make it possible to use the parfor-loop instead of the basic loop in matlab, so on multi-core processors runtime can be decreased. The adaptions are: 1. every iteration of the loop must be independent -> replace c=c+1 with a function using the loop variables. 2. Inner loop matrices must be sliced. It means, that every iteration of the loop can work on a complete vector of the matrices (nxm matrix–> n vectors 1xm and n iterations.). At the end, that sliced matrix can be reorganized into a vector, as the original code used to output it.

*Code changes: findNeighbor, line 27-37*

**Problem V**: long runtime of the function getSubIndex.

Solution: The runtime can be decreased by using a parallelization. So the values of several subchatchments can be calculated at the same time. Not many adaptions must be implemented here, and it will incidentally also solve Problem III. Because a parfor loop needs sliced variables, the buffer matrix A must be discarded. But A isn’t necessary anyway and the code now saves the output values directly into the sliced cell array “areas”.

*Code changes: getSubIndex.m becomes getSubIndex\_parfor.m, line 53; 86-90*

*processRaw\_fast: replace all getSubIndex with getSubIndex\_parfor*

**Runtime comparison without/with solutions to problem IV&V on a dual core cpu:**

|  |  |  |
| --- | --- | --- |
|  | Without (processRaw.m) | with (processRaw\_fast.m) |
| 1st run | 4930 s | 2040 s |
| 2nd run | 730 s | 50 s |

# update up to 14.10.2015, Bea

**Problem 6**: matlab timers are inherently unstable.

* Solution: Switch to launch daemon. Master.m is replaced by matlab\_batcher.py.

**Problem 7**: The proper working of the different routines has to be tested regularly during code development.

* Solution: Programming of unit tests for getRaw, processRaw, and runModel.

**Problem 8**: Different versions of the rainfall runoff model exist.

* Solution: Consistent implementation of git hub repository, production of regular releases for major updates and running of stable release on the mac mini (server). The model is now called RRMDA

**Problem 9**: rrm/src/setup copies templates of all scripts to the application folder. My modifications are all stored in /app/Themi/src. Other changes have been done in /app/Themi/src instead of in /src as well.

* Clean up setupGUI.m and /app/Themi/src. Look ad diff and merge the files. Use following command to check differences between files:   
  $ diff -E -b -B -y --strip-trailing-cr src/setup/controlData.m app/Themi/src/controlData\_Themi.m > controlData.diff
  + Copy getRaw\_Themi to src/setup.
  + Copy processRaw\_Themi to src/setup
  + Copy runModel\_Themi to src/setup
  + Copy sendtoDB\_Themi to src/setup
  + Copy controlData\_Themi to src/setup
* Adapt src/setup/setupGui.m to copy my scripts as well. Have them modular! Following edits in setupGUI.m:
  + Remove writing of master.m which has been replaced with a batcher in python in combination with a plist.
  + Add copyfile line for MatlabMail.m, controlData.m, matlab\_oda\_batcher\_offline.py
* Add documentation to the readme.docx

**Problem 10**: Overall goal: Make RRM more modular. Allow different data assimilation techniques by coupling the model to openDA. More details on the coupling of RRMDA and openDA can be read in the pptx in the git hub repository model\_RRMDA\_Themi.

* Solution part 1: Write wrappers in a separate openDA module. -> model\_RRMDA\_Themi.
* Solution part 2: Adapt matlab\_batcher.py
* Solution part 3: Write openDA configuration files.