**Table of Contents**

[Defining Root Architecture and Root Hydraulic Parameters 3](#_Toc513811486)

[Create architecture using RootTyp model 3](#_Toc513811487)

[Calculate root hydraulic parameters from RootTyp 3](#_Toc513811488)

[Create architecture using RootBox model 3](#_Toc513811489)

[Calculate root hydraulic parameters from RootBox 4](#_Toc513811490)

[Generation of Input Files for PFLOTRAN-Root Model 5](#_Toc513811491)

[Installation/Configuration – University of Michigan 6](#_Toc513811492)

[Environment Variables and Modules 6](#_Toc513811493)

[PETSc configuration 6](#_Toc513811494)

[pflotran-dev-root-system build 7](#_Toc513811495)

[Installation/Configuration – Blue Waters 8](#_Toc513811496)

[Modules 8](#_Toc513811497)

[Environment Variables 8](#_Toc513811498)

[PETSc configuration 9](#_Toc513811499)

[Current settings (4/2018) 9](#_Toc513811500)

[Job Submission 9](#_Toc513811501)

[Regression Tests 11](#_Toc513811502)

[Submitting a Job to Flux 12](#_Toc513811503)

[PBS Job Script Format 12](#_Toc513811504)

[PBS Job Submission 13](#_Toc513811505)

[Debugging the Code 15](#_Toc513811506)

[Using GDB 15](#_Toc513811507)

[Using PFLOTRAN-Root Debug Mode 15](#_Toc513811508)

[Comparing versions of the code 16](#_Toc513811509)

[Blue Waters DDT 16](#_Toc513811510)

[Adjusting Solver Tolerances 18](#_Toc513811511)

[Solver Theory 18](#_Toc513811512)

[Issue of False Convergence 18](#_Toc513811513)

[Default Tolerance Values 19](#_Toc513811514)

[PFLOTRAN Inputs 20](#_Toc513811515)

[PFLOTRAN Outputs 21](#_Toc513811516)

[Post-processing Scripts 22](#_Toc513811517)

[File Utilities 22](#_Toc513811518)

[readPFData - Read in PFLOTRAN hdf5 files 22](#_Toc513811519)

[readRootID - Read in root parameter hdf5 files 22](#_Toc513811520)

[readRootPET - Read in potential transpiration dat files 23](#_Toc513811521)

[cellIDtoXYZ – Cell mapping tool 24](#_Toc513811522)

[checkMassBalance 25](#_Toc513811523)

[Root Water Uptake 26](#_Toc513811524)

[domainET - Domain Transpiration 26](#_Toc513811525)

[domainHR - Domain Hydraulic Redistribution 27](#_Toc513811526)

[treeET – Individual Transpiration 28](#_Toc513811527)

[plotTreeET – Individual Transpiration Plot 29](#_Toc513811528)

[temporalUptakeProfile 30](#_Toc513811529)

[Centroids 30](#_Toc513811530)

[centroidUptake 30](#_Toc513811531)

[wtdCtdUptake 31](#_Toc513811532)

[radialUptakeCentroid 32](#_Toc513811533)

[Soil Moisture 33](#_Toc513811534)

[rootCellSM 33](#_Toc513811535)

# Defining Root Architecture and Root Hydraulic Parameters

## Create architecture using RootTyp model

REF: Pagès, L., Vercambre, G., & Drouet, J. (2004). Root Typ: a generic model to depict and analyse the root system architecture. Plant and Soil, 103–119.

RootTyp [source code, RootTyp1.cpp] was developed by Loic Pagés [ref]. It requires the following input files: sol.txt (soil parameters) and param.txt (root parameterizations). See Appendix for example input files.

Compile the source code:

g++ -o RootTyp.exe RootTyp1.cpp

To run, make sure param.txt and sol.txt are in executable’s directory and run command:

./RootTyp.exe

Output is found in noeud.txt

Results can be visualized using the read\_noeud\_and\_lr\_cleaned.m script in MATLAB.

## Calculate root hydraulic parameters from RootTyp

REF: Couvreur, V., Vanderborght, J., & Javaux, M. (2012). A simple three-dimensional macroscopic root water uptake model based on the hydraulic architecture approach. *Hydrology and Earth System Sciences*, *16*(8), 2957–2971.

Files needed:  
macro\_hydraulic\_parms.m --script  
Noeud2IM.m --script  
read\_condroot.m --script

noeud.txt --RootTyp output  
CondRoot\_1.in --script input

Open MATLAB, and go to directory where those files are, and type in:

>>[Krs,Kcomp,SUF,coordinates]=macro\_hydraulic\_parms('CondRoot\_0.in',0,'noeud\_0.txt')

## Create architecture using RootBox model

REF: Leitner, D., Klepsch, S., Bodner, G., & Schnepf, A. (2010). A dynamic root system growth model based on L-Systems. *Plant and Soil*, *332*(1-2), 177–192. doi:10.1007/s11104-010-0284-7

Dunbabin, V. M., Postma, J. a., Schnepf, A., Pagès, L., Javaux, M., Wu, L., … Diggle, A. J. (2013). Modelling root–soil interactions using three–dimensional models of root growth, architecture and function. *Plant and Soil*, *372*(1-2), 93–124. doi:10.1007/s11104-013-1769-y

The RootBox model is a series of MATLAB scripts which can be modified to suit users’ needs. It uses a series of L-system strings which are essentially a series of prescribed instructions which build the root architectures.

The input file is saved as a MATLAB script that can be run using the Windows MATLAB interface or the Unix MATLAB interface that can be found on the FLUX environment (see appendix for an example). For Windows, open MATLAB and type in:

>>rootbox\_input\_file\_name

In the FLUX environment, the equivalent command is:

$matlab –r rootbox\_input\_file\_name

## Calculate root hydraulic parameters from RootBox

Note: at this time, to calculate macroscopic root parameters, the root system matrix ‘str’ is required. If the root system is small, these parameters can be calculated directly in MATLAB using the following command:

>>[Krs,Kcomp,SUF,coordinates]=macro\_hydraulic\_parms('CondRoot\_0.in',1,str)

For the FLUX environment, a job should be submitted to the PBS job scheduler using the gen\_root\_flux.m script. See the appendix for an example.

qsub PBS\_calc\_root\_hydraulic\_params

TODO: Update this section to include more information about the calculation of the hydraulic parameters.

# Generation of Input Files for PFLOTRAN-Root Model

Inputs needed:  
Tree Information\* – the tree locations (X and Y) within the domain and DBH in cm  
PET\* – Hourly potential evaporation for the length of the simulation time  
SUF – generated by marco\_hydraulic\_parms script (needed for each root system type)  
SUF coordinates – x, y, z coordinates of the root node generated by macro\_hydraulic\_parms

\* *inputs needed for synthetic root structure*

# Getting Started

To get started, download the PFLOTRAN-Root model and petsc.

## PFLOTRAN

## PETSc

UPDATE FOR VERSIONS LATER THAN AUGUST 2014:

The most current versions PFLOTRAN-Root have been parallelized and merged with the current PFLOTRAN-dev versions. As such, the following branch of PETSc should be “checked out” from the git repository before configuration.

git clone https://bitbucket.org/petsc/petsc petsc  
cd petsc  
git checkout 821a7925fede8aa3b3b482fc9ccb2d087e2f80fa

./configure --with-mpi-dir=<dir> --with-hdf5-dir=<dir> --with-blas-lapack-dir=<dir>

mpi-dir = /usr/local/Cellar/open-mpi/4.0.2

# Installation/Configuration – University of Michigan

## Environment Variables and Modules

Include the following in the bashrc file:

export PETSC\_ARCH=petsc3\_arch-intel1

#use this version of petsc to run latest version of pflotran-dev

#and also for versions of pflotran-dev-root-system newer than June 2014

export PETSC\_DIR=/home/lizagee/pflotran\_petsc/petsc

#use this version of petsc to run pflotran-dev-root-system

#export PETSC\_DIR=/home/lizagee/pflotran\_petsc/petsc-dev09162013

Include the following in the default file in the ./home/user/privatemodules directory:

module load intel-comp/12.1

module load mkl/11.1

module load cmake

module load torque

module load hdf5/1.8.9/intel/12.1

module load openmpi/1.6.0/intel/12.1

module load hypre/2.8.0b/intel/12.1

#uncomment the following to compile pflotran-dev  
#module swap intel-comp intel-comp/14.0.2

## PETSc configuration

pflotran-dev-root-system currently requires an older version on PETSc which I have housed in the ./home/lizagee/pflotran\_petsc/petsc-dev09162013 directory. Use the following configuration to compile (do not copy/paste):

./config/configure.py --with-mpi-dir=$MPI\_HOME \

--with-debugging=1 \

--with-shared-libraries=0 \

--download-mumps=1 \

--download-parmetis=1 \

--download-metis=1 \

--download-scalapack=1 \

--download-blacs=1 \

--download-ml=1 \

--download-hdf5=1 \

--with-blas-lapack-lib="$MKL\_ROOT/lib/intel64/libmkl\_intel\_lp64.a $MKL\_ROOT/lib/intel64/libmkl\_core.a $MKL\_ROOT/lib/intel64/libmkl\_sequential.a"

NOTE: If during configuration, the message “ATTENTION! pax archive volume change required” appears, you must enter “.” to continue with the configuration. If you do not do so within a timely fashion, the process will time out.

Once the configure process is complete, follow the make prompts provided by the program.

If for some reason the process is unsuccessful, it is essential to remove the architecture directory before attempting to rebuild. The example command for this would be:

rm –rf petsc\_arch-intel1

## pflotran-dev-root-system build

I also had viable results with the intel-comp/12.1 (I’m not sure that the updated intel-comp/14.0.2 will work).

cd pflotran-dev-root-system/src/pflotran  
make root\_system=1 pflotran

This version will not build if you leave out the root\_system=1 tag. (Remember this for later: make pflotran 2>&1 | tee make\_log)

UPDATE FOR VERSIONS LATER THAN JUNE 2014:

cd pflotran-dev-root-system/src/pflotran  
make pflotran

|  |  |
| --- | --- |
| C:\Users\lizagee\Downloads\idea-light-bulb-clip-art-black-and-white-MTLEnkBTa.jpeg | To disable the warnings during the compilation. Open the makefile and insert ‘MYFLAGS += -w’ below ‘MYFLAGS = -I.’. |

# Installation/Configuration – Blue Waters

Currently, the Blue Waters configuration must be done using GNU compilers. When using Intel compilers, PFLOTRAN will fail on input file read-in. Developer Glenn Hammond is currently investigating the issue, but for now, the following configuration should be used:

## Modules

The following module list can be exactly duplicated using the following commands in the login shell. It will not replicate exactly within the job environment, which I believe is alright, but I have not tested it yet.

module swap PrgEnv-cray PrgEnv-gnu

module unload cray-libsci

module load cmake

module load forge

Currently loaded modules:

1) modules/3.2.10.4  
2) eswrap/1.3.3-1.020200.1278.0  
3) cray-mpich/7.3.3  
4) torque/6.0.1  
5) moab/9.0.2.TAS2-1477409647\_6706986-sles11  
6) java/jdk1.8.0\_51  
7) globus/5.2.5  
8) gsissh/6.2p2  
9) xalt/0.7.6  
10) scripts  
11) user-paths  
12) darshan/2.3.0.1  
13) cmake/3.1.3  
14) forge/6.0.3.1  
15) gcc/4.9.3  
16) craype-interlagos  
17) craype-network-gemini  
18) craype/2.5.4  
19) udreg/2.3.2-1.0502.10518.2.17.gem  
20) ugni/6.0-1.0502.10863.8.28.gem  
21) pmi/5.0.10-1.0000.11050.179.3.gem  
22) dmapp/7.0.1-1.0502.11080.8.74.gem  
23) gni-headers/4.0-1.0502.10859.7.8.gem  
24) xpmem/0.1-2.0502.64982.5.3.gem  
25) dvs/2.5\_0.9.0-1.0502.2188.1.113.gem  
26) alps/5.2.4-2.0502.9774.31.12.gem  
27) rca/1.0.0-2.0502.60530.1.63.gem  
28) atp/2.0.1  
29) PrgEnv-gnu/5.2.82

## Environment Variables

export CRAYPE\_LINK\_TYPE=static

export CRAY\_ADD\_RPATH=yes

export PETSC\_DIR= {your PETSc folder}

export PETSC\_ARCH=BW-gnu-debug

## PETSc configuration

The recorded PETSc changeset for PF-Root will not compile using GNU compilers on Blue Waters. Instead use the following changeset and configuration.

git clone https://bitbucket.org/petsc/petsc petsc

git checkout c41c7662de68b036bda6be236f939e8b55959cb0

./configure --configModules=PETSc.Configure --optionsModule=PETSc.compilerOptions --with-cc=cc --with-cxx=CC --with-fc=ftn --with-debugging=1 --with-c2html=0 --download-parmetis=1 --download-mumps=1 --download-metis=1 --download-scalapack=1 --download-blacs=1 --download-ml=1 --download-hdf5=1 --download-fblaslapack=1 --with-shared-libraries=0 --with-mpiexec=aprun --PETSC\_ARCH=BW-gnu-debug

|  |  |
| --- | --- |
| C:\Users\lizagee\Downloads\idea-light-bulb-clip-art-black-and-white-MTLEnkBTa.jpeg | **Notes from Glenn Hammond**   * PFLOTRAN should always be compiled as a static executable (as opposed to a dynamically linked one) * PFLOTRAN uses the compiler flags set for the given PETSC\_ARCH. * Default PETSc is non-optimized. Probably should build without debugging, as well. Production runs should utilize a PETSc configuration that adds the flags:--COPTFLAGS=’-O3’ --FOPTFLAGS=’-O3’ |

## Current settings (4/2018)

export CRAYPE\_LINK\_TYPE=dynamic

export CRAY\_ADD\_RPATH=yes

export PETSC\_DIR= {your PETSc folder}

export PETSC\_ARCH=BW-gnu-debug

module swap PrgEnv-cray PrgEnv-gnu

module unload cray-libsci

module load cmake

module load forge

module unload darshan

module list

Notes: darshan is an I/O profiler that is used by the Blue Waters admins to determine what I am using Blue Waters to do. Jon Calhoun has mentioned that is a major culprit in codes breaking, so he recommended removing it and then rebuilding.

## Job Submission

To submit an interactive job on BW:

qsub -I -l nodes=1:ppn=32:xe -l walltime=4:00:00

To run the PFLOTRAN executable

aprun -n <N> ~/pflotran\_petsc/pflotran-dev-root-system/src/pflotran/pflotran >runfile.txt

Where <N> is the number of desired processors.

# Regression Tests

Regression tests verify the installation of PFLOTRAN by running a series of small simulations and comparing the output with certain “gold standard” files. The regression tests are performed with Python versions >2.7.x, so the Python module must be loaded into the FLUX environment.

module load python

Once the module is loaded, the regression tests can be run with the following commands:

python regression\_tests.py \  
--executable ../src/pflotran/pflotran \  
--config-file root/root.cfg

# Submitting a Job to Flux

## PBS Job Script Format

REF: CAEN Advanced Computing, *Using PBS on nyx/flux*, http://cac.engin.umich.edu/resources/systems/nyx/pbs

Jobs are submitted to the Flux environment with a PBS script. The following is an example script for the running of the pflotran model:

#PBS -A <flux account name>

#PBS -l qos=flux

#PBS -q flux

#PBS -l walltime=100:00:00

#PBS -l nodes=2

#PBS -l mem=20gb

#PBS -N EX

#PBS -M <your email address>

#PBS -m abe

#PBS -V

# for Bourne shell

ulimit -c unlimited

cd <dir of simulation inputs>

# Run the executable in the background

echo "I ran on:"

cat $PBS\_NODEFILE

mpirun -np 1 <pflotran executable path> -pflotranin <input file path> > runfile\_UMBS.txt

The PBS script parameters are as follows:

#PBS -N your-mpi-job

Name of the job in the queue is "your-mpi-job". This can be anything as long as it is less that 13 characters long; you should make it descriptive so you know which of your jobs are running and queued.

#PBS -A cac

The name of the account you want to use. The account will determine what resources you have available to you and the destination of any charges that may be applied (in the case of flux). While the example uses 'cac', only use it if that is what you intended, otherwise you might be limiting yourself or your job may not run at all.

To see what accounts you have available to you, type and look at the list for ALIST:

mdiag -u <your username> -v

#PBS -l qos=cac

The name of the Quality of Service (qos) for your job. In most cases, this will be "flux" if you are using Flux, or "cac" if you are using Nyx. To find out which QoS options you have, type and look at the list for QLIST:

mdiag -u <your username> -v

#PBS -l nodes=7:ppn=2,walltime=1:00:00

Reserve 14 processors for 1 hour. Please note that this example requests 7 groups of two processors, with each group of two processors in the same physical node. If you do not need a particular geometry, you can replace "nodes=7:ppn=2" with "procs=14" instead, which will allow your job to start sooner because it won't have to wait for a specific geometry to be available.

#PBS -l pmem=500mb

Reserve 500 megabytes for each process requested. In this case, 7 nodes \* 2 processors per node is 14 processors \* 500mb, for a total of around 7 gigabytes of total memory.

# PBS -S /path/to/shell

Script is /bin/sh (see below)

#PBS -q cac

Submit to the queue named cac.

#PBS -M YourEmailAddressGoesHere

Email me at this address.

#PBS -m abe

Email me when the job **a**borts, **b**egins, and **e**nds.

#PBS -j oe

Join your stdout and stderr output into one file, to be placed in your home directory.

#PBS -V

Takes your current environment (like PATH and other variables) and sends them along with your job to the compute node. This is especially use with Modules

## PBS Job Submission

To submit a PBS job to the queue, use the following command:

qsub <PBS\_Script\_Name>

Upon submission, a job number will be given. To check the status of the current job, use:

qstat <PBS\_Job\_Number>

To check the status of all jobs in the account queue and current node usage, use:

showq –w acct=<account\_name>

# Debugging the Code

## Using GDB

In the flux environment, to start the GDB debugger, just enter

$gdb

To specify the program to debug

(gdb) file <path to executable>

To specify arguments for the program to receive, such as input files, use the set args command:

(gbd) set args “arguments for program to receive”

You can check which arguments are currently being used with the show args command.

Once all arguments are set, run the program using the run command.

(gdb) run.

|  |
| --- |
| **EXAMPLE**  [lizagee@flux-login2]$ gdb  (gdb) file /home/lizagee/pflotran\_petsc/pflotran-dev-root-system\_clone/src/pflotran/pflotran  Reading symbols from /home/lizagee/pflotran\_petsc/pflotran-dev-root-system\_clone/src/pflotran/pflotran...done.  (gdb) set args -pflotranin ./pflotran\_UMBS\_2roots\_ref.in  (gdb) show args  Argument list to give program being debugged when it is started is "-pflotranin ./pflotran\_UMBS\_2roots\_ref.in".  (gdb) run  **OUTPUT**  Program received signal SIGSEGV, Segmentation fault.  0x0000000000855088 in INIT\_MODULE::assignrootproptoregions (realization=...) at init.F90:4101  4101 cur\_patch => cur\_level%patch\_list%first |

## Using PFLOTRAN-Root Debug Mode

The root system model has a built in debug code which prints out soil water pressures, sink terms, and other relevant information when unfavorable conditions are reached (e.g., soil water pressure below the wilting point). This results in large runfiles which can slow down the simulation run, so debug mode is not intended for final simulations.

To use debug mode, the option must be activated at the time of build (see: insert link to installation).

cd pflotran-dev-root-system/src/pflotran  
make root\_debug=1 pflotran

To split runfiles into manageable sizes for reading in standard text editors, use the following bash command:

[lizagee@flux-login2 runfile]$ split --help

Usage: split [OPTION]... [INPUT [PREFIX]]

Output fixed-size pieces of INPUT to PREFIXaa, PREFIXab, ...; default

size is 1000 lines, and default PREFIX is `x'. With no INPUT, or when INPUT

is -, read standard input.

Mandatory arguments to long options are mandatory for short options too.

-a, --suffix-length=N use suffixes of length N (default 2)

-b, --bytes=SIZE put SIZE bytes per output file

-C, --line-bytes=SIZE put at most SIZE bytes of lines per output file

-d, --numeric-suffixes use numeric suffixes instead of alphabetic

-l, --lines=NUMBER put NUMBER lines per output file

--verbose print a diagnostic just before each

output file is opened

--help display this help and exit

--version output version information and exit

SIZE may be (or may be an integer optionally followed by) one of following:

KB 1000, K 1024, MB 1000\*1000, M 1024\*1024, and so on for G, T, P, E, Z, Y.

## Comparing versions of the code

To determine the version of the PFLOTRAN code you are using, use the hg parent command.

|  |
| --- |
| [lizagee@flux-login2 pflotran-dev-root-system\_2015]$ hg parent  changeset: 8392:c8f914e32da9  tag: tip  parent: 8034:f18a3af6c1d8  parent: 8391:4a4324cb3abe  user: Gautam Bisht <gbisht@lbl.gov>  date: Thu Oct 09 19:40:59 2014 -0700  summary: merged with pflotran-dev |

The information regarding the PETSc hash is stored in **/pflotran\_dir/tools/buildbot/petsc/petsc-git-version.txt**.

TODO: include information about the **diff** command and also helpful **hg** and **git** functions.

## Blue Waters DDT

Resource: https://bluewaters.ncsa.illinois.edu/ddt

Blue Waters provides access to the debugging program called DDT from (also known as Allinea Forge) which can be used via GUI to look at parallel processes and walk through the code as it is running. To access it, log into Blue Waters using X11 forwarding and issue the following commands:

|  |
| --- |
| > ssh –Y agee@h2ologin.ncsa.illinois.edu  > module load ddt-memdebug  > ddt |

You must then specify the program executable and environment variables you want in play.

# Adjusting Solver Tolerances

## Solver Theory

Solvers use standard local error control techniques to monitor the error at each time step. During each time step, the solvers compute the state values (in the case of Richards mode, soil water pressure) at the end of the step and also determine the local error, the estimated error of these state values. They then compare the local error to the *acceptable error*, which is a function of the relative tolerance (*rtol*) and absolute tolerance (*atol*). If the nonlinear solve does not converge, the time step is cut and the nonlinear iterations repeated. If after a maximum number of time step reductions the nonlinear equations still fail to converge, PFLOTRAN gives up and stops execution.

**Relative tolerance (rtol)** measures the error relative to the size of each state. The relative tolerance represents a percentage of the state’s value. For example, 1E-3, would mean the computed state will be accurate to within 0.1%.

**Absolute tolerance (atol)** is a threshold error value. This tolerance represents the acceptable error as the value of the measured state approaches zero.

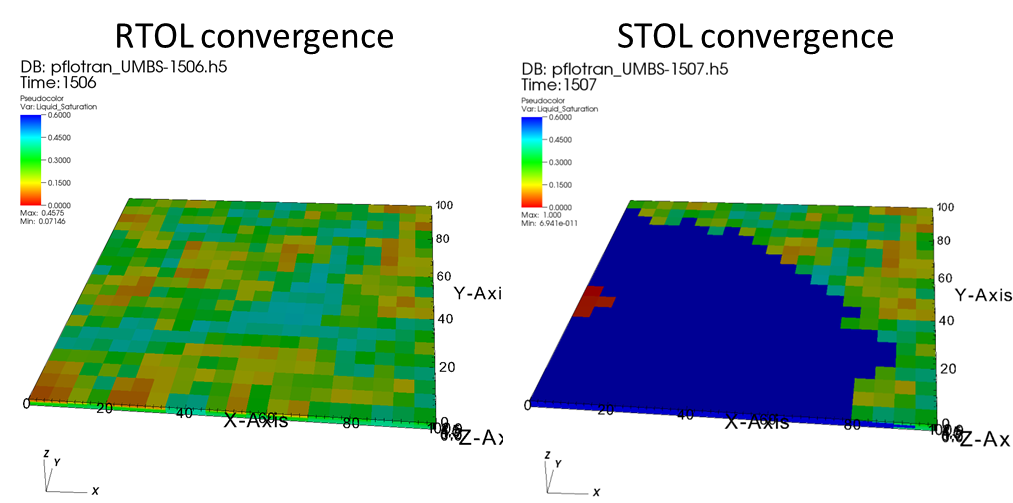
**Relative update tolerance (stol)** is the relative decrease in size of the 2-norm of the solution. (Tang et al., 2016) addressed numerical challenges in coupling PFLOTRAN with CLM-CN and cited the issues with loose STOL tolerances leading to false convergence. The default STOL (10-8) was tightened to 10-12, with minimal increases in computation time.

The **infinity tolerance (itol)** looks at the infinity norm of the residual (e.g., the maximum residual),

The **infinity update tolerance (itol\_update)** is a special tolerance formulated by Glenn Hammond for PFLOTRAN. With it, you can specify that the nonlinear solve is converged once the max pressure change in a cell is less than the tolerance (e.g., 1.0 in Pascal).

## Issue of False Convergence

A potential issue with the RWU simulations is an issue of false convergence. False convergence is when the nonlinear solve converges on a solution that meets specified thresholds, but is not realistic. Simulations may continue for several time steps after the false convergence, but generally fail due to strong gradients and reduced time steps. An example is shown below.



## Default Tolerance Values

|  |  |  |
| --- | --- | --- |
| **Tolerance Type** | **Default Value** | **Convergence Code** |
| ATOL | 1E-50 | 2 |
| RTOL | 1E-8 | 3 |
| STOL | 1E-8 | 4 |
| DTOL | -2E0 |  |
| MAXNORM | 1E+20 |  |
| ITOL | 1E-50 | 10 |
| ITOL\_UPDATE | 1E-50 | 11 |

# PFLOTRAN Inputs

Filename: Rootsystem.h5

📂 Regions

📂 Rootsystem1 – domain grid cells that root system 1 occupies

❑ Cell Ids

📂 Soil – domain grid cells that NO root systems occupy (soil only)

❑ Cell Ids

# PFLOTRAN Outputs

PFLOTRAN outputs state variables of liquid pressure, relative saturation, domain actual transpiration, and actual transpiration for specified root systems. Additional settings can be implemented to output other aspects of flow and transport. [TODO: Add additional output information later]

Filename: pflotran\_input\_name-HH.h5

📂 Coordinates

❑ X [m] *64-bit floating point N x 1*

❑ Y [m] *64-bit floating point M x 1*

❑ Z [m] *64-bit floating point K x 1*

📂 Provenance

📂 PETSc

 detail\_petsc\_config

 detail\_petsc\_parent

 detail\_petsc\_status

📂 PFLOTRAN

 detail\_pflotran\_diff

 detail\_pflotran\_fflags

 detail\_pflotran\_parent

 detail\_pflotran\_status

 pflotran\_compile\_date\_time

 pflotran\_compile\_hostname

 pflotran\_compile\_user

 pflotran\_input\_file

📂 Time: 1.00000E+00 h

❑ Actual\_Transpiration [kg\_m^2\_s] – *64-bit floating point N x M x K*

❑ Liquid\_Pressure [Pa] – *64-bit floating point N x M x K*

❑ Liquid\_Saturation – *64-bit floating point N x M x K*

❑ Material\_ID – *32-bit integer N x M x K*

❑ Root\_ID\_RootSystem1 – *32-bit integer N x M x K*

❑ T\_RootSystem1 [kg\_m^2\_s] – *64-bit floating point N x M x K*

# Post-processing Scripts

## File Utilities

### readPFData - Read in PFLOTRAN hdf5 files

[actual\_T\_root, domain\_T, P, S] = readPFData(filehead, hr, rep\_root\_system\_ID)

**Input:**

|  |  |
| --- | --- |
| filehead | The entire filename that comes before the hour. Ex: C:\SIMULATION\pflotran\_UMBS- |

**Output:**

|  |  |
| --- | --- |
| actual\_T\_root | Cell array which contains the hourly uptake for individual root systems. Cell array has dimensions {n, m} where n is the length of the rep\_root\_system\_ID vector and m is the length of the hour vector |
| domain\_T | Cell array {m x 1} which contains a vector for each hour of total uptake. |
| P | Cell array {m x 1} containing hourly soil water pressure [Pa]. |
| S | Cell array {m x 1} containing hourly relative saturation [ - ]. |

### readRootID - Read in root parameter hdf5 files

[ind\_roots, SSD] = readRootID(filename)

**Input:**

|  |  |
| --- | --- |
| filename | The entire filename for the root parameters. EX: C:\SIMULATION\root\_param\_1.h5 |

**Output:**

|  |  |
| --- | --- |
| ind\_roots | Vector containing root cell indices. |
| SSD | Vector containing the standard sink density for the root system. |

### readRootPET - Read in potential transpiration dat files

[PET] = readRootPET(filename)

**Input:**

|  |  |
| --- | --- |
| filename | The entire filename for the potential transpiration time series. EX: C:\SIMULATION\Trans\_RootSystem1.dat |

**Output:**

|  |  |
| --- | --- |
| PET | Vector containing potential transpiration [kg/s] time series. |

### cellIDtoXYZ – Cell mapping tool

[x,y,z]=cellIDtoXYZ(cellid)

**Description:** Given a PFLOTRAN grid cell ID, this determines the midpoint x, y, z coordinate of the cell. Assumes z = 0 is the bottom of the domain of the domain, with values increasing towards the ground surface.

**Input:**

|  |  |
| --- | --- |
| cellid | PFLOTRAN cell ID |

**Output:**

|  |  |
| --- | --- |
| x | X-coordinate of cell (midpoint) |
| y | Y-coordinate of cell (midpoint) |
| z | Z-coordinate of cell (midpoint) |

### checkMassBalance

checkMassBalance(filename,numRoots)

**Description:** Uses the PFLOTRAN mass balance data file (must specify to print in the pflotran.in file). File has the following formation: Time [h], dt\_flow [h], Global Water Mass [kg], bottom water mass [kg], bottom water mass [kg/h], RootSystem1 water mass [kg], RootSystem1 water mass [kg/h], …, RootSystemN water mass [kg], RootSystemN water mass [kg/h].

**Input:**

|  |  |
| --- | --- |
| filename | File location of the mass balance data file. |
| numRoots | Number of roots planted in the domain |

**Output:**

Generates a figure of the hourly difference between dm (initial global mass + drainage + precipitation + uptake) and the current global mass with the expectation that dm should be near zero.

## Root Water Uptake

### domainET - Domain Transpiration

[Tcum\_mm, T\_mm]=domainET(domain\_T, plot\_flag)

**Description:** Computes the domain scale cumulative transpiration (Tcum\_mm) and hourly time series, both in [mm]. The original data output is in [kg s-1]. There is a discrepancy in the HDF5 output files that lists the output units as [kg m-2 s-1]. Note: the current plot is specific to the UMBS 2010 simulation domain.

**Input:**

|  |  |
| --- | --- |
| domain\_T | The cell array containing the Actual\_Transpiration variable for each hour. |
| plot\_flag | 1 = plot cumulative transpiration, 0 = no plot |

**Output:**

|  |  |
| --- | --- |
| Tcum\_mm | Cumulative transpiration in [mm] |
| T\_mm | Transpiration time series in [mm/hr] |

### domainHR - Domain Hydraulic Redistribution

[HR,T\_mm\_woHR] = domainHR(domain\_T, plot\_flag)

**Description:** Computes the domain scale cumulative transpiration (Tcum\_mm) and hourly time series, both in [mm]. The original data output is in [kg s-1]. There is a discrepancy in the HDF5 output files that lists the output units as [kg m-2 s-1]. Note: the current plot is specific to the UMBS 2010 simulation domain.

**Input:**

|  |  |
| --- | --- |
| domain\_T | The cell array containing the Actual\_Transpiration variable for each hour. |
| plot\_flag | 1 = plot cumulative transpiration, 0 = no plot |

**Output:**

|  |  |
| --- | --- |
| HR | Total source from root systems in [mm/hr], displayed as positive number |
| T\_mm\_woHR | Hourly transpiration |

### treeET – Individual Transpiration

[T\_ind, HR\_ind, Tcum\_ind]=treeET(rootsys\_T, plot\_flag)

**Description:** Computes the individual scale cumulative transpiration (Tcum\_ind) and hourly time series (T\_ind), both in [mm]. The original data output is in [kg s-1]. There is a discrepancy in the HDF5 output files that lists the output units as [kg m-2 s-1]. Note: the current plot is specific to the UMBS 2010 simulation domain.

**Input:**

|  |  |
| --- | --- |
| rootsys\_T | The cell array containing the root system transpiration variable for each hour. |
| plot\_flag | 1 = plot cumulative transpiration, 0 = no plot |

**Output:**

|  |  |
| --- | --- |
| HR\_ind | Total source from root systems in [mm/hr], displayed as positive number |
| Tcum\_ind | Cumulative individual transpiration |
| T\_ind | Hourly individual transpiration |

### plotTreeET – Individual Transpiration Plot

plotTreeET(T\_ind)

**Description:** Plots the hourly time series (T\_ind), in [mm/hr]. The original data output is in [kg s-1].

**Input:**

|  |  |
| --- | --- |
| T\_ind | Hourly individual transpiration |

### temporalUptakeProfile

[T\_img] = temporalUptakeProfile(rootsys\_T,ii)

**Description:** Plots timeseries of rootsystem uptake with depth.

**Input:**

|  |  |
| --- | --- |
| rootsys\_T | Cell array containing the individual root uptake for all simulation hours |
| ii | Vector index of root system to plot |

**Output:**

|  |  |
| --- | --- |
| T\_img | Array containing individual root uptake with depth with time [k, hr] |

## Centroids

### centroidUptake

[centroid] = centroidUptake(T\_root)

**Description:** Computes the centroid of uptake [m] for the current timestep for a vector taken from the actual\_T\_root cell array. Follows the steps: (1) at each depth, sum the uptake over xy; (2) calculate the “area” of transpiration; (3) find the moment = area\*midpt; (4) calculate the centroid c = sum(moment)/sum(area).

**Input:**

|  |  |
| --- | --- |
| T\_root | Vector containing the uptake of root j at hour i. |

**Output:**

|  |  |
| --- | --- |
| centroid | Centroid of uptake [m] taken from the datum (z = 0 at the bottom of the domain). |

### wtdCtdUptake

[wctd] = weightedCtdUptake(hrStart,rootsys\_T)

**Description:** Computes the centroid of uptake [m] for the current timestep for a vector taken from the actual\_T\_root cell array. Follows the steps: (1) at each depth, sum the uptake over xy; (2) calculate the “area” of transpiration; (3) find the moment = area\*midpt; (4) calculate the centroid c = sum(moment)/sum(area).

**Input:**

|  |  |
| --- | --- |
| hrStart | Simulation hour the sunlight hours begin (e.g., noting the beginning of the transpirative day) |
| rootsys\_T | The cell array containing the root system transpiration variable for each hour. |

**Output:**

|  |  |
| --- | --- |
| wtcd | Weighted centroid of uptake [m] taken from the datum (z = 0 at the bottom of the domain). |

### radialUptakeCentroid

[th,r] = radialUptakeCentroid(T\_root,center\_ij)

**Description:** Computes the lateral centroid of uptake in radial coordinates.

**Input:**

|  |  |
| --- | --- |
| T\_root | Vector containing the uptake of root j at hour i. |
| center\_ij | The cell index values for the center of the stem given as a vector of form [i , j]. Can be found using xyzToCellID |

**Output:**

|  |  |
| --- | --- |
| th | Polar angle of the centroid [rad]. |
| r | Polar radius of the centroid [m]. |

## Soil Moisture

### rootCellSM

[th\_roots\_5, th\_roots\_95, th\_roots\_min, th\_roots\_ave, th] = rootCellSM(S,ind\_roots,th\_s,th\_r)

**Description:** Provides statistics for the root cell soil moisture.

**Input:**

|  |  |
| --- | --- |
| S | Entire relative saturation cell array. |
| ind\_roots | Root cell indices obtained from readRootID |
| th\_s | Soil water content at saturation [m3 m-3] |
| th\_r | Residual soil water content [m3 m-3] |

**Output:**

|  |  |
| --- | --- |
| th\_roots\_5 | Soil water content 5th percentile for root cells. |
| th\_roots\_95 | Soil water content 95th percentile for root cells. |
| th\_roots\_min | Minimum soil water content for root cells. |
| th\_roots\_ave | Mean soil water content for root cells |
| th | Cell array containing hourly root cell soil water content |