CSCT-D5C3 User's Manual

Technical report of the Python functions for the
CANDU-SCWR Coupling Toolset - DONJON5 CATHENA3

Generated by Doxygen 1.9.1

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 PyEquilibrium.py File Reference

Contains ReachEqui(), the function that executes DONJON to reach neutronic equilibrium.

Functions

- def PyEquilibrium.ReachEqui (PATH_EXEC, PATH_PROC)

  INPUTS
  1) PATH_EXEC *; str, absolute path where to execute DONJON*
  2) PATH_PROC *; str, absolute path where DONJON procedures and databases are stored*

2.1.1 Detailed Description

Contains ReachEqui(), the function that executes DONJON to reach neutronic only equilibrium.

Author

U. Le Tennier (March 2021)

ReachEqui() is a trivial function that executes DONJON. The calculation options must be directly written in the DONJON input file. At the end of execution, the user is able to retrieve the DONJON output required to start coupling calculation. The equilibrium calculation cannot be restarted from where it stopped. If the process is interrupted, it has to be restarted from the beginning. The DONJON files retrieved are .OUT files. To start coupling, the extension of those files must be changed to .INP.

2.1.2 Setup the environment

Before executing ReachEqui(), the user must create a *proc* directory. In this *proc* must be placed the multicompos and several DONJON procedures (.c2m). Depending on the loading plan and whether the impact of reloading on fissile inventories is taken into account, the user must provide the good set of multicompos and DONJON procedures.

A total of 9 multicompos must be provided in the *proc* directory. Finally, the DataTherm.INP and both DBREFL1.INP DBREFL2.INP files are required to give a first distribution to thermodynamical parameters and to generate the reflector macrolib.

The required directories architecture follows :

A) EXEC_DIRECTORY

a) proc

1) SCWR64Geo.c2m
2) SCWR64FII.c2m
3) SCWR64FIC.c2m
4) SCWR64Cr.c2m
5) SCWR64Mc1.c2m
6) SCWR64Re.c2m
7) SCWR64Mc2.c2m $*$
8) DataTherm.INP
9+) DBREXXxX.INP
10+) DBREFLX.INP

b) SCWR64N1Eq.x2m

c) donjon.exe

d) rdonjon5.bat

SCWR64Mc1.c2m is only required when reloading is taken into account. Otherwise, it is not necessary to put this file in *proc* directory.

Remarks

In the SCWR64N1Eq.x2m file, the user must define which options he wants to use. Option 1 is about the refueling, whether a 3-batches or 4-batches cycle is chosen. The user must then replace the "OPTION1" string in SCWR64N1Eq.x2m by either "3c" or "4c". The second option to choose is the number of cycles to simulate to reach equilibrium. It is advised to replace "OPTION2" with 18 if "3c" is used or 29 if "4c" is used. The third option is the time model used. Are available "REP1b" "REP2" "CANDU5" "CANDU3" and "CANDU2", a description of each is given is the time loop of each SCWR64N1Eq.x2m file. It is advised to use "CANDU5"

2.2 PyMain_lvl2m.py File Reference

Contains Coupling_lvl2() function that performs coupling between CATHENA3 and DONJON5 for the safety analysis of the CANDU-SCWR.

Functions

- def PyMain_lvl2m.Coupling_lvl2 (PATH_EXEC, PATH_PROC_DJ, PATH_PROC_CA, Step, CycleRef, TimeModel, LoadModel, IncConvC, IncConvD)

INPUTS

1) PATH_EXEC	*; str, absolute path where to execute all the operations*
2) PATH_PROC_DJ	*; str, absolute path where DONJON files are stored*
3) PATH_PROC_CA	*; str, absolute path where CATHENA files are stored*
4) Step	*; int, index of current calculation step*
5) CycleRef	*; int, number of cycles to simulate*
6) TimeModel	*; str, how to go through each cycle, available : "CANDU1" - "CANDU2" - "CANDU3" - "REP1" - "REP1b" - "REP2"*
7) LoadModel	*; str, which load model to be considered, 3 or 4 batches, available : "3c" - "4c" this information must match with the multicompos*
8) IncConvC (kg/s)	*; float, convergence criteria for the CATHENA simulations*
9) IncConvD (kW)	*; float, convergence criteria for the coupled simulations*

OUTPUTS

a) Exec_report	*; text file, execution report that is copied in the execution directory when simulation is over*
b) Param_DistX	*; text files containing distributions of thermalhydraulic parameters, only returned for normal end of execution (files are created in PATH_EXEC). Note that X in Param_DistX can take values :*
X = A	*; Density of upward coolant (kg/m3, CaloUp Dens, DCu) ;*
X = B	*; Temperature of upward coolant (K, CaloUp Temp, TCu) ;*
X = C	*; Density of downward coolant (kg/m3, CaloDw Dens, DCd) ;*
X = D	*; Temperature of downward coolant (K, CaloDw Temp, TCd) ;*
X = E	*; Temperature of fuel (K, Fuel Temp, TF) ;*
X = F	*; Inner fuel ring centerline temperature (K) ;*
X = G	*; Outer fuel ring centerline temperature (K) ;*
X = H	*; Inner fuel ring cladding surface temperature (K) ;*
X = I	*; Outer fuel ring cladding surface temperature (K) ;*
X = M	*; Mass flow (kg/s).*

2.2.1 Detailed Description

Contains Coupling_lvl2() function that performs coupling between CATHENA3 and DONJON5 for the safety analysis of the CANDU-SCWR.

Author

U. Le Tennier (March 2020), (rev. 02/2021 for CNL)

Coupling_lvl2() uses file managing functions available in PyProcs_lvl1m. The algorithm is presented in IGE-379.pdf. In PyMain_lvl2m.py is provided the function Coupling_lvl2() which performs the coupling. The global architecture of the environment required is specified hereafter. If the impact of reloading on fissiles inventories has to be taken into account, the user must use the function named Couplinglvl2Th() and the corresponding DONJON inputs. The only difference between Coupling_lvl2() and Coupling_lvl2Th() is that the last one is able to handle the MICROLIB.INP object. Before executing Coupling_lvl2(), the user must choose or create a directory where the execution will take place (so called EXEC_DIRECTORY). In this directory, several elements will be placed, they are listed hereafter. Once done, the code will upload PyProcs_lvl1m that contains several functions used by the main. Fortran structures

handling functions will be uploaded from PyProcs_lvl1m

The directories architecture follows :

A) EXEC_DIRECTORY

a) CA

1) TEMPLATE_CATH

b) DJ

1) TEMPLATE_SCWR64N1Flu.x2m
2) TEMPLATE_SCWR64N1Relo.x2m
3) TEMPLATE_SCWR64N1Upda.x2m
4) SCWR64N1Crit.c2m
5) SCWR64N1MacU.c2m
6) FMAP.INP (∗)
7) HISTORY.INP
8) GEOCORE.INP (∗) (!)
9) MATEX.INP (∗) (!)
10) TRACK.INP (∗) (!)
11) MACRORFL.INP (∗) (!)
12) FMAP0.INP (∗) (!)
13) HISTORY0.INP (!)
14) MICROFL.INP (∗)
1X) DBREXXxX.INP (!)

c) PY

1) PyMain_lvl2m.py
2) PyProcs_lvl1m.py
3) ASCIIGetv4.py
4) ASCIILenv4.py
5) ASCIILibv4.py
6) ASCIIOpnv4.py
7) ASCIISixv4.py

d) DISTRIBUTION

e) donjon.exe
f) cat3_6_1_1-b01.exe
g) rdonjon5.bat

In order to upload the required procedures, the code automatically defines PATH_EXEC, PATH_PROC_DJ and PATH_PROC_CA which are the paths to EXEC_DIRECTORY, **DJ** and **CA** respectively. If the directories architecture is respected, the three paths are consistent with a Coupling_lvl2() call. Otherwise, if the call to Coupling_lvl2() follows the equilibrium calculation, the STEP will be 1. If the LOADMODEL is "3c", it is advised to use the value of 6 for CYCLEREF. If the LOADMODEL is "4c", it is advised to use the value of 8 for CYCLEREF. Only CANDU5 is usually used as TIMEMODEL. Finally, common values for INCCONVC and INCCONVD are 0.3 kg/s and 5 kW.

Remarks

CA directory contains the CATHENA main input template (prefilled) generated by WriteScriptM.CompleteWriting.

DJ directory contains the DONJON pre-filled inputs. The .x2m are the DONJON main input templates (pre-filled), TEMPLATE_SCWR64N1Flu.x2m is used for the flux calculation, TEMPLATE_SCWR64N1Relo.x2m for reloading and TEMPLATE_SCWR64N1Upda.x2m to update the fuelmap. SCWR64N1Crit.c2m is a procedure that finds out the boron quantity required to reach criticality. It calls SCWR64N1MacU.c2m to interpolate the

databases DBREFXXxX.INP.

Other files in **DJ** directory are DONJON data structures. (∗) files are retrieved after neutronic equilibrium calculations, performed with PyEquilibrium.ReachEqui. (!) files are only read, never updated. FMAP.INP is the fuelmap, HISTORY.INP contains important informations to carry the coupling and also several indicators such as the k_eff or the critical quantity of boron (ppmB). FMAP0.INP and HISTORY0.INP are vanilla copies of respectively FMAP.INP and HISTORY.INP. They are used if the user wants to restart coupled calculation from neutronics equilibrium. Finally DBREXXxX.INP are the different multicompo objects (databases). Depending on the way the assemblies at the edge of the core are treated. If corner and side assemblies are used, nine multicompos are required. Otherwise, only three multicompos are required. The corresponding SCWR64N1MacU.c2m should be used.

PY directory contains the present file, the PyProcs_lvl1m and ASCII procedures. PyProcs_lvl1m provides functions used by Coupling_lvl2. ASCII procedures enable to navigate efficiently in DONJON data structures.

DISTRIBUTION directory is initially empty. Coupling_lvl2 fills and updates it with the last cycle solution. Then it uses the stored information to begin an iteration with the solution found during the previous cycle. If Coupling_lvl2 is interrupted, information in DISTRIBUTION can be used to recover every parameter distribution found for the last cycle. For instance, if the interruption occurs at step 6 of the cycle 4, DISTRIBUTION contains every information from step 6 of cycle 3 to step 5 of cycle 4.

The files donjon.exe and cat3_6_1_1-b01.exe are the DONJON and CATHENA executables. rdonjon5.bat is the batch file that performs the execution of donjon.exe If a different name is used for one of the e) f) or g) item, Coupling_lvl2 must must be updated (look and find for the items in the code and replace their name by the accurate one).

2.2.2 Error handling

Four internal booleans are used : DON1_exec, DON2_exec, DON3_exec and CATH_exec. If the execution of CATHENA or DONJON goes wrong, the corresponding boolean are set to False. Then, the execution of the main program terminates and the errors are notified in the Exec_report. The correspondance between each boolean and each script executed follows :

DON1_EXEC is related to SCWR64N1Flu execution
DON2_EXEC is related to SCWR64N1Upda execution
DON3_EXEC is related to SCWR64N1Relo execution
CATH_EXEC is related to CATHENA execution

Most of the time, errors come from SCWR64N1Flu because it is the first and most complicated call to DONJON. A quick look at the bottom of the SCWR64N1Flu.results gives insights about out to fix the problem. If there is one typical mistake to highlight, it is the misuse of SCWR64N1MacU. If the SCWR64N1MacU placed in **DJ** is not consistent with the number of mixtures required, DONJON execution could crash. Particularly, if a call to more than 9 mixtures in MacU is made and the other inputs account for 9 mixtures, the code will crash.

If an error is triggered elsewhere in the code, the boolean **errors** turns True. **errors** is updated after a call to certain PyProcs procedures. Then execution of the main stops after writing the Exec_report in which insights are given about where the error happened.

Sometimes, sudden errors happen but are not related to the code. For instance, the code fails to find TEMPLATE↩ _CATH in **CA** directory. Such a case will only trigger python error messages. This error can happen after the code was running for several days. No explanation can be given to this kind of bug. It is only required to restart the code where it crashed.

2.2.3 How to restart

If the coupling is interrupted, to restart the user needs to :
1) Clear the execution directory (remove what is not listed in the directories architecture)
2) Open HISTORY.INP file in **DJ** directory
3) Look in the "Step" repertory what is the last number written
4) Call Coupling_lvl2 with the Step retrieved at **3**)

2.3 PyProcs_lvl1m.py File Reference

Contains all the function used by the main coupling function Coupling_lvl2().

Functions

- def PyProcs_lvl1m.readN1Flu (PATH_PROC_DJ, FILE_HISTORY)

  Read the history to retrieve useful information

  INPUTS
  1) PATH_PROC_DJ *; str, absolute path to DJ directory*
  2) FILE_HISTORY *; str, name of the history file, should be "HISTORY.inp"*

  OUTPUTS
  a) Reload *; binary, = 1 if reloading must be performed*
  b) CycleIndex *; int, index of the current cycle or cycle to prepare*
  c) Step *; int, index of the calculation step*
  d) Out *; binary, = 1 all the calculation are done in DONJON*
  e) errors *; boolean, = True if the wrong lines are considered*

- def PyProcs_lvl1m.writeN1Flu (PATH_EXEC, FILE_FLU, Step, CycleRef, TimeModel, LoadModel)

  Generate SCWR64N1Flu.x2m from its template to prepare the calculation at required Step

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) FILE_FLU *; str, name of the Flu.x2m file, should be 'SCWR64N1Flu.x2m'*
  3) Step *; int, index of current calculation step*
  4) CycleRef *; int, total number of cycles to simulate*
  5) TimeModel *; str, how to go through each cycle, available : "CANDU1" - "CANDU2" - "CANDU3" - "REP1" - "REP1b" - "REP2"*
  6) LoadModel *; str, which load model to be considered, 3 or 4 batches, available : "3c" - "4c" this information must match with the multicompos, available : "3c" - "4c"*

  OUTPUTS
  a) errors *; boolean = True if the writing zone was not found*
  b) wentgood *; boolean = True if everything was written correctly*

- def PyProcs_lvl1m.writeN1Relo (PATH_EXEC, FILE_RELO, Step, LoadModel)

  Generate SCWR64N1Relo.x2m from its template to prepare the new cycle, cycle index retrieved from HISTORY at Step

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) FILE_RELO *; str, name of the Relo.x2m file, should be 'SCWR64N1Relo.x2m'*
  3) Step *; int, index of current calculation step*
  4) LoadModel *; str, which load model to be considered, 3 or 4 batches, available : "3c" - "4c" this information must match with the multicompos, available : "3c" - "4c"*

OUTPUTS
a) errors *; boolean = True if the writing zone was not found*

- def PyProcs_lvl1m.writeN1Upda (PATH_EXEC, FILE_UPDA, ThermoP, INPUTX1, INPUTX2, INPUTX3, INPUTX4, INPUTX5)

  Generate SCWR64N1Upda.x2m from its template to update the local parameter INPUT1 retrieved from INPUT2

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) FILE_UPDA *; str, name of the Upda.x2m file, should be 'SCWR64N1Upda.x2m'*
  3) ThermoP *; str, which parameter to consider available : "CaloUp Dens" - "CaloUp Temp" - "CaloDw Dens" - "CaloDw Temp" - "Fuel Temp"*
  X) INPUTXX *; str (5), 5 distributions of the 5 ThermoP possibilities*

  OUTPUTS
  a) errors *; boolean = True if the writing zone was not found*

  IMPORTANT : THIS FUNCTION WILL COPY IN A DONJON INPUT FILE A SET OF LOCAL PARAMETERS, EACH DONJON LINE CAN CONTAIN A MAX OF 72 CHARACTERS. HERE, 960 LINES OF 7 VALUES ARE WRITTEN, IF EACH OF THE SEVEN VALUES HAS A LENGTH OF 9 CHARAC (ex: 1234567.9) THEN THE LINE WILL CONTAIN 70 CHARACTERS If CATHENA gives values with more than 8 digits, reshape internal variables Text and Li

- def PyProcs_lvl1m.readN1Fmap (PATH_EXEC, FILE_FMAP, FILE_HISTORY)

  Retrieve power distribution from a FuelMap

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) FILE_FMAP *; str, name of the fuelmap file, should be "FMAP.out"*
  3) FILE_HISTORY *; str, name of the history file, should be "HISTORY.out"*

  OUTPUTS
  a) param_dist *; array (10∗1344∗5), power distribution, FuelMap format*
  b) errors *; boolean = True if the writing zone was not found*

- def PyProcs_lvl1m.writeN1CATH (PATH_EXEC, FILE_CATH, Pwr_Chn, Pwr_Coeff, MFlow_Dist, TimeCath)

  Produce CATHENA input file from its template

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) FILE_CATH *; str, name of the fuelmap file, should be "FMAP.out"*
  3) Pwr_Chn *; array (84∗1), total channel power of 4 equivalent channels, created by treatN1paramDC()*
  4) Pwr_Coeff *; array (84∗20), channel axial power distribution, created by treatN1paramDC()*
  5) MFlow_Dist *; float array (84∗1), flow mass rate distribution*
  6) TimeCath *; float, execution time for CATHENA, by default 1500*

  OUTPUTS
  a) errors *; boolean = True if the writing zone was not found*

- def PyProcs_lvl1m.readN1CATH (PATH_EXEC, ThermoP, Param_DistCa)

  Retrieve thermal hydraulic distributions, updates Param_DistCa

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) ThermoP *; str, which parameter to consider available : "CaloUp Dens" - "CaloUp Temp" - "CaloDw Dens" - "CaloDw Temp" - "Fuel Temp"*
  3) Param_DistCa *; array (10∗336∗5), 10 distributions of thermohydraulic parameters*

  OUTPUTS
  a) param_distca *; array (10∗336∗5), parameter distribution, CATHENA format*
  b) errors *; boolean = True if the writing zone was not found*

- def PyProcs_lvl1m.treatN1paramDC (Param_Dist)

  Treat power distribution to make it consistent with CATHENA input

  INPUTS

1) Param_Dist	*; array (10∗1344∗5), parameter distribution, FuelMap format, retrieved from readN1Fmap()*

  OUTPUTS

a) pwr_chn	*; array (84∗1), total channel power of 4 equivalent channels*
b) pwr_coeff	*; array (84∗20), channel axial power distribution*

- def PyProcs_lvl1m.treatN1paramCD (Param_Dist, Param_DistCa)

  Treat thermal hydraulic distributions to make it consistent with DONJON input and create str for final coupling outputs

  INPUTS

1) Param_Dist	*; array (10∗1344∗5), parameter distribution, FuelMap format, retrieved from readN1Fmap()*
2) Param_DistCa	*; array (10∗336∗5), parameter distribution, CATHENA format*

  OUTPUTS

a) param_dist	*; array (10∗1344∗5), parameter distribution, FuelMap format*
b) DCu_coeff	*; str, upward densities temperature values (5 values per line)*
c) TCu_coeff	*; str, upward coolant temperature values (5 values per line)*
d) DCd_coeff	*; str, downward densities temperature values(5 values per line)*
e) TCd_coeff	*; str, downward coolant temperature values (5 values per line)*
f) TF_coeff	*; str, fuel temperature values (5 values per line)*
g) TCI_coeff	*; str, inner ring fuel centerline temperature values (5 values per line)*
h) TCO_coeff	*; str, outer ring fuel centerline temperature values (5 values per line)*
i) TSi_coeff	*; str, inner ring fuel cladding temperature values (5 values per line)*
j) TSO_coeff	*; str, outer ring fuel cladding temperature values (5 values per line)*

- def PyProcs_lvl1m.checkN1convcath (PATH_EXEC, FILE_CONV, FILE_MFLO, IncConvC)

  Assess if the CATHENA simulation converged and create str for final coupling outputs (mass flow only)

  INPUTS

1) PATH_EXEC	*; str, absolute path to execution directory*
2) FILE_CONV	*; str, name of the fuelmap file, should be "CONV.RES"*
3) FILE_MFLO	*; str, name of the mass flow file, should be "MFLOW.RES"*
4) IncConvC	*; float, convergence criteria (kg/s) for the CATHENA simulations*

  OUTPUTS

a) convc	*; boolean , = True if the CATHENA calculation converged*
b) MaxCat	*; float , = Maximum (absolute) discrepancy in a channel*
c) MassFLW_dist	*; array (84∗1), mass flow distribution*
d) MassFLW_coeff	*; str , mass flow values (5 values per line)*
e) errors	*; boolean = True if the reading zones were not found*

- def PyProcs_lvl1m.checkN1convdist (Param_Dist, Param_Dist_new, IncConvD)

  Compare two consecutive distributions to assess if the simulation converged or not

  INPUTS

1) Param_Dist	*; array (10∗1344∗5), old parameter distribution, FuelMap format*
2) Param_Dist_new	*; array (10∗1344∗5), new parameter distribution, FuelMap format, retrieved from readN1Fmap()*
3) IncConvC	*; float, convergence criteria (kW) for simulation*

  OUTPUTS

a) convc	*; boolean, = True if the distributions converged*
b) maxP	*; float , Max mean squared difference between powers of 2 channels*
c) maxTCu	*; float , Max absolute difference between 2 TCu*
d) maxTCd	*; float , Max absolute difference between 2 TCd*
e) maxTF	*; float , Max absolute difference between 2 TF*

- def PyProcs_lvl1m.helpconv (Param_Dist, Param_Dist_new)

  Calculate the mean value of 2 sets of distributions

  INPUTS
  1) Param_Dist *; array (10∗1344∗5), old parameter distribution, FuelMap format*
  2) Param_Dist_new *; array (10∗1344∗5), new parameter distribution, FuelMap format, retrieved from readN1Fmap()*

  OUTPUTS
  a) param_dist *; array (10∗1344∗5), parameter distribution, FuelMap format*
  b) DCu_coeff *; str, upward densities temperature values (5 values per line)*
  c) TCu_coeff *; str, upward coolant temperature values (5 values per line)*
  d) DCd_coeff *; str, downward densities temperature values(5 values per line)*
  e) TCd_coeff *; str, downward coolant temperature values (5 values per line)*
  f) TF_coeff *; str, fuel temperature values (5 values per line)*
  g) TCI_coeff *; str, inner ring fuel centerline temperature values (5 values per line)*
  h) TCO_coeff *; str, outer ring fuel centerline temperature values (5 values per line)*
  i) TSi_coeff *; str, inner ring fuel cladding temperature values (5 values per line)*
  j) TSO_coeff *; str, outer ring fuel cladding temperature values (5 values per line)*

- def PyProcs_lvl1m.stockN1dist (PATH_EXEC, ThermoP, Step, LoadModel, XXX_coeff)

  Store the converged solution (thermal-hydraulics parameters distributions) into DISTRIBUTION directory, each file in DISTRIBUTION contains one parameter distribution at only one step, file names as "Dist_PARAM_Step"

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) ThermoP *; str, which parameter to consider available : "CaloUp Dens" - "CaloUp Temp" - "CaloDw Dens" - "CaloDw Temp" - "Fuel Temp" -"Fuel Temp" - "Fuel CITemp" - "Fuel COTemp" - "ISheathTemp" - "OSheath↩ Temp" - "MassFlow "*
  3) Step *; int, index of current calculation step*
  4) LoadModel *; str, which load model to be considered, 3 or 4 batches, available : "3c" - "4c" this information must match with the multicompos, available : "3c" - "4c"*
  5) XXX_coeff *; str, ThermoP parameter values (5 values per line), must match with ThermoP*

- def PyProcs_lvl1m.getN1dist (PATH_EXEC, ThermoP, Step, LoadModel)

  Retrieve the converged solution (thermal-hydraulics parameters distributions) from DISTRIBUTION directory

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) ThermoP *; str, which parameter to consider available : "CaloUp Dens" - "CaloUp Temp" - "CaloDw Dens" - "CaloDw Temp" - "Fuel Temp" -"Fuel Temp" - "Fuel CITemp" - "Fuel COTemp" - "ISheathTemp" - "OSheath↩ Temp" - "MassFlow "*
  3) Step *; int, index of current calculation step*
  4) LoadModel *; str, which load model to be considered, 3 or 4 batches, available : "3c" - "4c" this information must match with the multicompos, available : "3c" - "4c"*

  OUTPUTS
  a) xxx_coeff *; str, ThermoP parameter values (5 values per line), must match with ThermoP*

- def PyProcs_lvl1m.getN1mflw (PATH_EXEC, INPUT1, INPUT2)

  Retrieve the converged solution (mass flow distribution) from DISTRIBUTION directory

  INPUTS
  1) PATH_EXEC *; str, absolute path to execution directory*
  2) Step *; int, index of current calculation step*
  3) LoadModel *; str, which load model to be considered, 3 or 4 batches, available : "3c" - "4c" this information must match with the multicompos, available : "3c" - "4c"*

  OUTPUTS
  a) MFlow_dist *; array (84,1), mass flow distribution*

Variables

- **PyProcs_lvl1m.PATH_PROC_PY** = os.path.dirname(__file__)

2.3.1 Detailed Description

PyProcs_lvl1m.py contains all the function used by the main coupling function Coupling_lvl2().

Author

U. Le Tennier (March 2020), (rev. 02/2021 for CNL)

Various functions are provided. The description of how they are chained is provided in IGE-379.pdf. Some functions write specific information such as power or different parameter distributions in CATHENA or DONJON inputs. Others read the information in both HISTORY and FUELMAP (DONJON structures) or in CATHENA outputs. There is a total of 11 parameters measured. 7 of them are useful for coupling calculation : Power, upward coolant density (DCu) and temperature (TCu), downward coolant density (DCd) and temperature (TCd), fuel temperature (TF) and finally mass flow. The other parameters are the inner/outer rods cladding surface temperature and inner/outer rods centerline temperature.

Because CATHENA models a quarter core and DONJON a full core, different arrays with different formats are used. Some functions enable to switch from one format to another. Most of the time, arrays of 3 dimensions are used to handle parameters. The first dimension states which parameter is considered (10 possibilities). The two others are related to the position of the assembly in the core. Because DONJON files can only contain characters before column 72, it was chosen to limit the number of values on a line to 5. To be consistent with this limitation, most of the array have the following format (10xAx5). Their second dimension can have 336 or 1344 possibilities, respectively for a quarter or a full core. Besides, in some functions, the digits are managed to avoid any crash due to the 72 characters limitation.

Mass flow distribution is handled differently because each channel has only one mass flow (because of steady-state simulations). 84x1 arrays are used. Besides, CATHENA requires power per channel in addition with relative power sharing between every node of the channel, which involves 84x20 arrays. Finally, functions to store and get parameter distributions in DISTRIBUTION directory, to check convergence of both CATHENA et coupled calculation and to help convergence are provided.

To help convergence, a new distribution is created from the two previous calculations. By design, each channel has fixed inlet/outlet coolant temperatures and densities. The solution provided by CATHENA faces those important restrictions, no exotic situation can happen. From iteration to iteration, distributions frequently oscilliate around a mean value. Therefore, when convergence helped is triggered, the new distributions are the mean distributions from the two previous calculations.

To easily recover information from DONJON structures, ASCII functions are used. They need to be imported from PATH_PROC_PY. If the global architecture suggested in PyMain_lvl2m.py is properly implemented, ASCII functions are located in the same directory as the present file. The code automatically finds the required path to make the import. If not, the exact path to the ASCII functions must be provided to PATH_PROC_PY variable in the section IMPORTS of the present file.

Remarks

Overall, functions inputs are written in capital letters, output have no capital letters and local variables have some capital letters but not only. Boolean are used from python to python, if a binary information has to be retrieved or given to CATHENA or DONJON, it will be as an integer : 0 or 1

2.4 TreatMapMFlw.py File Reference

Contains the data treatment functions to handle mass flow distributions retrieved from coupling.

Functions

- def TreatMapMFlw.CreateMatrixMFlw (FileId, CyNumb, CyStep, EvTy)

  Creates a 10x10x1 matrix and fill it with data in FileId, unfolds the geometry

  INPUTS
  1) FileId : string, absolute path to MassFlow_report
  2) CyNumb : int, number of targeted cycle
  3) CyStep : int, number of targeted step of cycle CyNumb
  4) EvTy : string, reloading model, available : "3c" and "4c"

  OUTPUTS
  a) matrix : array(10x10x1), mass flow distribution of a quarter core

- def TreatMapMFlw.CreateHMapMFlw (matrix)

  Creates a quarter core heatmap from a 10x10x1 matrix

  INPUTS
  1) matrix : array(10x10x1), mass flow distribution retrieved from CreateMatrixMFlw()

2.4.1 Detailed Description

Contain the data treatment functions to handle mass flow distributions retrieved from coupling.

Author

U. Le Tennier (March 2021)

Two functions are available : CreateMatrixMFlw() and CreateHMapMFlw(). CreateMatrixMFlw() retrieves the mass flow distribution from the coupling output and returns the matrix object that can be directly used as an input for CreateHMapMFlw(). CreateMatrixMFlw() uses as input the absolute path where the coupling output is located. It also uses the cycle number CyNumb and the cycle step CyStep to pick up the required information from the coupling outputs. The type of evolution EvTy (3-batches "3c" or 4-batches "4c") is finally asked.

Remarks

Certain combinations of CyStep and EvTy are impossible. For example, a "4c" has only 18 steps per cycle. An error can occur if an inconsistent call is made.

If the data required is not available but the combination (CyStep and EvTy) is possible, the codes does not return an error. It returns a cordial invitation to check if the data required exists in the coupling output. A possible reason is that the coupling code was interrupted while processing. There is still a way to retrieve the data of the last cycle simulated which lies in the DISTRIBUTION folder.

By default, at the end of CreateChart(), the line that enables to save directly the chart is written as a commentary. To use it, uncomment it and give the path where you want to save the chart.

CreateHMapMFlw() uses seaborn module to produce the heatmap. By default, the colormap used is coolwarm. Additionnal informations about seaborn and colormap options can be found on the following pages :
seaborn package https://seaborn.pydata.org/generated/seaborn.heatmap.html
additionnal colormaps https://matplotlib.org/3.1.0/tutorials/colors/colormaps.↩
html
create colormaps https://matplotlib.org/3.1.1/tutorials/colors/colormap-manipulation.↩
html
additionnal informations https://www.kdnuggets.com/2019/07/annotated-heatmaps-correlation-ma
html

2.5 TreatMaps.py File Reference

Contains the data treatment functions to handle parameter distributions retrieved from coupling (except mass flow).

Functions

- def TreatMaps.CreateMatrix (FileId, CyNumb, CyStep, EvTy)

  Creates a 20x20x20 matrix and fill it with data in FileId (coupling outputs), unfolds the geometry

  INPUTS
  1) FileId : string, absolute path to the parameter_report.txt
  2) CyNumb : int, number of targeted cycle
  3) CyStep : int, number of targeted step of cycle CyNumb
  4) EvTy : string, reloading model, available : "3c" and "4c"

  OUTPUTS
  a) matrix : array(20x20x20), distribution of parameter for a full core

- def TreatMaps.CreateMatrixPwr (FileId, CyNumb, CyStep, EvTy)

  Creates a 20x20x20 matrix and fill it with data in FileId (donjon outputs), unfolds the geometry

  INPUTS
  1) FileId : string, absolute path to FMAP.inp
  2) CyNumb : int, number of targeted cycle
  3) CyStep : int, number of targeted step of cycle CyNumb
  4) EvTy : string, reloading model, available : "3c" and "4c"

  OUTPUTS
  a) matrix : array(20x20x20), power distribution for a full core

- def CreateHmap() (matrix, z)

  Creates a quarter core heatmap from a 20x20x20 matrix

  INPUTS
  1) matrix : array(20x20x20), distribution retrieved from CreateMatrix() or CreateMatrixPwr()
  2) z : int, elevation to display : 0 bottom of the core, 19 top

2.5.1 Detailed Description

Contain the data treatment functions to handle parameter distributions retrieved from coupling (except mass flow).

Author

U. Le Tennier (March 2021)

Three functions are available : CreateMatrix(), CreateMatrixPwr() and CreateHmap(). CreateMatrix() retrieves the distribution from the coupling outputs (.txt) and returns the matrix object that can be directly used as an input for CreateHmap(). CreateMatrixPwr() works almost exactly the same but only retrieves power distribution in the fuelmap object. Therefore, it uses additional python functions (ASCIIOpnv4, ASCIIGetv4 and ASCIISixv4) that enable to navigate effectively in donjon outputs.
CreateMatrix() and CreateMatrixPwr() use as input the absolute path where are located the coupling outputs or the fuelmap. It also uses the cycle number CyNumb and the cycle step CyStep to pick up the required information from the coupling outputs. The type of evolution EvTy (3-batches "3c" or 4-batches "4c") is finally asked. CreateHmap() requires the matrix and the height z required. z belongs to [0;19] : 0 corresponds to the bottom of the core (where the flow reverses its path), 19 is at the top.

From the matrix built, different charts can be created, for instance the relative axial power distribution. Because it is relatively easy to create such a chart and impossible to cover all the possibilities, no dedicated function is provided. It is up to the user to create his own additional functions.

Remarks

Certain combinations of CyStep and EvTy are impossible. For example, a "4c" has only 18 steps per cycle. An error can occur if an inconsistent call is made.

If the data required is not available but the combination (CyStep and EvTy) is possible, the codes does not return an error. It sends a cordial invitation to check if the data required exists in the coupling output. Such an invitation cannot be sent by CreateMatrixPwr() because no data can be lost in the fuelmap object. There is still a way to retrieve the data of the last cycle simulated which lies in the DISTRIBUTION folder.

Because of the different values that can be found, the colorscale must be adjusted at the beginning of CreateHmap(). Otherwise, the heatmap risk to appear in only one color, without shades.

By default, at the end of CreateHmap(), the line that enables to save direclty the chart is wrote as a commentary. To use it, uncomment it and give the path where you want to save the chart.

CreateHmap() uses seaborn module to produce the heatmap. By default, the colormap used is coolwarm. Additionnal informations about seaborn and colormap options can be found on the following pages :
seaborn package https://seaborn.pydata.org/generated/seaborn.heatmap.html
additionnal colormaps https://matplotlib.org/3.1.0/tutorials/colors/colormaps.↩
html
create colormaps https://matplotlib.org/3.1.1/tutorials/colors/colormap-manipulation.↩
html
additionnal informations https://www.kdnuggets.com/2019/07/annotated-heatmaps-correlation-ma
html

2.6 WriteScriptM.py File Reference

Contains the function CompleteWriting() that produces n-channels CANDU-SCWR CATHENA input for coupling.

Functions

- def WriteScriptM.CompleteWriting (n)

  Aggregates strings and write the file 'CWRITING.INP' in the dir where WriteScriptM.py is located

  INPUTS
  1) n : int, number of channels to consider (only work with n = 84)

- def WriteScriptM.writeBase ()

  Write control group

- def WriteScriptM.writeCompo (n)

  Write components

- def WriteScriptM.writeConnec (n)

  Write connections

- def WriteScriptM.writeInit (n)

  Write initial conditions

- def WriteScriptM.writeTherMod (n)

  Write thermal models

- def WriteScriptM.writeSysCont (n)

  Write boundary conditions, system models (control devices) and outputs

2.6.1 Detailed Description

Contains the function CompleteWriting() that produces n-channels CANDU-SCWR CATHENA input for coupling.

Author

U. Le Tennier (March 2020), (rev. 02/2021 for CNL)

CompleteWriting() calls a total of 6 different functions to create a complete CANDU-SCWR input for CATHENA. Its output will be written in the CWRITING.INP file which is created in the directory where WriteScriptM.py is located.

Each function accounts for one or more definition groups. Functions writeBase(), writeInit() and writeTherMod() have special marks (respectively HERE00, HEREMFLi and HEREiIN/HEREiOUT) which are replaced by different values by Coupling_lvl2(). HERE00 will be replaced by the simulation time, HEREMFLi by the initial mass flow of the channel i and HEREiIN/HEREiOUT by the power distribution of the INNER or OUTER rods of the channel i.

Remarks

It is possible that os.path.dirname(**file**) does not work. A solution is to give the absolute path of the WriteScriptM.py file to PATH_WRITE variable. Besides, such path must be consistent with the computer used. For Windows, the \ mark separates the different levels. For a use with other OS, the mark must be changed

At the end of writeSystConst(), the valve component is declared. By default, the gain is of 1E-5 and the period is 30s. Some example of different setting are provided as comments. For more information on the valve component, please refer to IGE-379.pdf.

It is recommended to call CompleteWriting(84), for n different than 84, several adjustments either in the CATHENA input generating functions and in the coupling functions should be made to have a 336 channels core.

It is possible to remove the HERE marks by giving directly numerical values. HEREiIN/HEREiOUT in writeTherMod() must be replaced by several code lines. An example of those lines is part of the function but is a comments line. To use it, declare the lines that add the HERE mark as comments and uncomment the lines that declare the power distribution. Even with this modification, the coupling function will not crash but the coupling will not be a coupling anymore because the power will not go from DONJON to CATHENA.