**TECHNICAL REPORT**
**IGE-381**

# The V5-PyKit user guide

A. Hébert

Institut de génie nucléaire
Département de génie mécanique
École Polytechnique de Montréal
May 21, 2025

# Contents

# 1 The PyKit python bindings

The PyKit API is a collection of simple python3 utility classes dedicated to the modification or creation of APEX[1] and MPO[2] *multiparameter* files, as created by the APOLLO2[3] or APOLLO3[4] lattice codes. These two type of files are HDF5-implemented.[5] The PyKit API is build over the three PyGan classes, namely `lifo`, `lcm`, and `cle2000`.[6] The PyGan classes are wrapping the Version5 distribution of DRAGON5 and DONJON5, including the Version5 HDF5 bindings.[7,8]

The compilation and link edition of the PyGan API require the definition of a UNIX environment variables on a UNIX system. Add the information about the `HDF5_INC`, `HDF_API` and `FORTRANPATH` environment variables in the `.profile` or `.bashrc script`. These lines are OS-dependent.

**on OSX operating system:**

```
# Support for HDF5
export HDF5_INC="/opt/homebrew/Cellar/hdf5/1.14.6/include" # HDF5 include directory
export HDF5_API="/opt/homebrew/Cellar/hdf5/1.14.6/lib"     # HDF5 C API
# Support for Python3 API
export FORTRANPATH=/opt/homebrew/Cellar/gcc/14.2.0_1/lib/gcc/14/ # libgfortran.a
export PYTHONPATH=/opt/homebrew/lib/python3.13/site-packages/
```

**on recherche network at Polytechnique Montreal:**

```
# Support for Python3 and HDF5
if [ $MachineExtension = "-aix" ]
then
  export HDF5_INC="/usr/include" # HDF5 include directory
  export HDF5_API="$HDF5_INC/../lib" # HDF5 C API
  export FORTRANPATH="/usr/lib/gcc/x86_64-redhat-linux/4.8.5/" # libgfortran.so
elif [ $MachineExtension = "-ubuntu" ]
then
  export HDF5_INC="/usr/include/hdf5/serial/" # HDF5 include directory
  export HDF5_API="/usr/lib/aarch64-linux-gnu/hdf5/serial" # HDF5 C API
  export FORTRANPATH="/usr/lib/gcc/aarch64-linux-gnu/9/" # libgfortran.so
  export NVTOOLS="/opt/nvidia/hpc_sdk/Linux_aarch64/24.3/compilers/lib" # libnvf.so
  export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$GD_API"
  export PATH="$PATH:$NVTOOLS"
else
  export HDF5_INC="/usr/local/hdf5/include" # HDF5 include directory
  export HDF5_API="$HDF5_INC/../lib" # HDF5 C API
  export FORTRANPATH="/usr/lib/gcc/x86_64-redhat-linux/4.8.5/" # libgfortran.so
  export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$GD_API"
fi
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$HDF5_API"
```

**on RedHat 8 operating system:**

```
# Support for HDF5
export HDF5_INC="/usr/include" # HDF5 include directory
export HDF5_API="$HDF5_INC/../lib64" # HDF5 C API
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$HDF5_API"
# Support for Python3 API
export FORTRANPATH="/usr/lib/gcc/x86_64-redhat-linux/8/" # libgfortran.so
```

**on Ubuntu operating system:**

```
# Support for HDF5
export HDF5_INC="/usr/include/hdf5/serial/" # HDF5 include directory
export HDF5_API="/usr/lib/x86_64-linux-gnu/hdf5/serial" # HDF5 C API
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$HDF5_API"
# Support for Python3 API
export FORTRANPATH="/usr/lib/gcc/x86_64-linux/9/" # libgfortran.so
```

**on Scibian 10 operating system:**

```
# Support for HDF5
export HDF5_INC="/usr/include/hdf5/serial" # HDF5 include directory
export HDF5_API="/usr/lib/x86_64-linux-gnu/hdf5/serial" # HDF5 C API
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$HDF5_API"
# Support for Python3 API
export FORTRANPATH="/usr/lib/gcc/x86_64-linux-gnu/8/" # libgfortran.so
```

Before using PyKit, you need to compile and install the Version5 distribution. We recommend to install Version5 on the same directory as PyKit. After setting the PyGan environment variables, you may proceed with the following Unix shell instructions: commands are

```
tar xvfzp Version5.0.8_ev2640.tgz
cd Version5_ev2640/PyGan/
make hdf5=1
```

### 1.1 The `rpython` script

The V5-PyKit classes are importing the three PyGan classes `lifo`, `lcm`, and `cle2000`. Access to these classes is performed in the `rpython` script located in the `Version5_pykit` directory where the PyGan path is defined. This script is provided to help executing the V5-PyKit utility actions, but *is not mandatory* for using these utility actions. For example, dataset `pincell_mpo_concat.py` is executed as

```
./rpython pincell_mpo_concat.py
```

For `rpython` to work, additional information should be provided:

`pincell_mpo_concat.access`**:** Bourne script recovering the input MPO files

`pincell_mpo_concat.save`**:** Bourne script moving the output MPO file towards a convenient location

`pincell_mpo_concat_proc`**:** Sub-directory containing input files required by `pincell_mpo_concat.py`.

The `rpython` script sets the `PYTHONPATH` environment variable using a definition of the form

```
export PYTHONPATH=‘pwd‘/../Version5_ev2640/PyGan/lib/Linux_x86_64/python
```

You should check that `PYTHONPATH` is pointing towards the correct directory in your setting.

The `rpython` script also set `PYTHONMALLOC` as

```
export PYTHONMALLOC=debug
```

*Note:* Not using the `rpython` script and writing “`python3 pincell_mpo_concat.py`” may not work if input files and class definitions are not made available before the call. The `rpython` script is a convenient way to set input information.

### 1.2 The `SPH_MPO` script

The `SPH_MPO` utility perform a SPH equivalence on *each* calculation point of an MPO file. A new set of $G$-group equivalence factors is appended to the `LOCALVALUE` dataset in the output MPO file, as depicted in Fig. 1. The remaining of the MPO file is not modified.

The `SPH_MPO` utility, accessible from Python3, is imported by the command

```
from SPH_MPO import SPH_MPO
```

It has one constructor: `SPH_MPO()`, used to create an *object instance* `o` and one method to execute the utility function.

#### *1.2.1 Attribute Variables*

A SPH_MPO object `o` contains eight attribute variables:

`o.geom_text` name of the `'''...'''`-delimited text object containing the definition of the macro-geometry, as specified in Sect. 3.3 of Ref 7.

`o.name` name of the input MPO file

`o.htype` string set to “`RT`” (Raviart-Thomas diffusion or SPN macro calculation) or “`SN`” (discrete ordinates macro calculation).

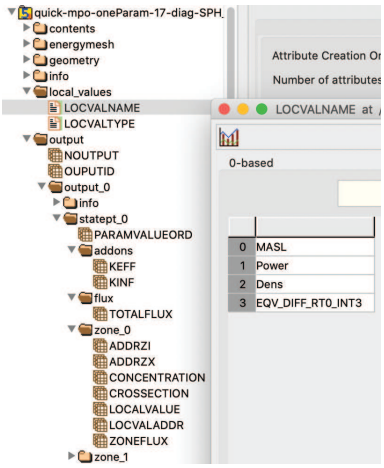


Figure 1: MPO enrichment with SPH factors.

`o.ipoly` polynomial order in space. 0: RT0 or DD0; 1: RT1 or DD1, etc.

`o.icol` type of Raviart-Thomas integration in the macro-calculation. 1: analytical; 2: Gauss-Lobatto; 3: Gauss-Legendre.

`o.isn` order of angular quadrature with SN method. 2: S2; 4: S4; 6: S6; 8: S8, etc.

`o.ispn` order of angular quadrature with SPN method. 0: diffusion theory; 1: SP1, 3: SP3, 5: SP5, etc.

`o.iscat` scattering anisotropy order. 1: isotropic in LAB; 2: P1; 3: P2; 4: P2, etc.

### *1.2.2 SPH_MPO()*

```
o = SPH_MPO(geom_text, name, [htype], [ipoly], [icol], [isn], [ispn], [iscat])
```

| **input parameter:** | | |
|---|---|---|
| `geom_text` | *string* | name of the `'''...'''`-delimited text object containing the definition of the macro-geometry, as specified in Sect. 3.3 of Ref 7. |
| `name` | *string* | name of the input MPO file |
| `htype` | *string* | string set to "`RT`" (Raviart-Thomas diffusion or SPN macro calculation) or "`SN`" (discrete ordinates macro calculation). |
| `ipoly` | *int* | polynomial order in space. 0: RT0 or DD0; 1: RT1 or DD1, etc. |
| `icol` | *int* | type of Raviart-Thomas integration in the macro-calculation. 1: analytical; 2: Gauss-Lobatto; 3: Gauss-Legendre. Must be set only if `htype=RT`. |
| `isn` | *int* | order of angular quadrature with SN method. 2: S2; 4: S4; 6: S6; 8: S8, etc. Must be set only if `htype=SN`. |
| `ispn` | *int* | order of angular quadrature with SPN method. 0: diffusion theory; 1: SP1, 3: SP3, 5: SP5, etc. Must be set only if `htype=RT`. |
| `iscat` | *int* | scattering anisotropy order. 1: isotropic in LAB; 2: P1; 3: P2; 4: P2, etc. Must be set only if `htype=SN` or if `ispn`$>0$. |

### *1.2.3 o.exec()*

This method execute the SPH procedure.

```
o.exec()
```

*1.2.4 example*

```
#
# Equivalence_SPH_MPO: incorporation of SPH factors in an MPO file
#
from SPH_MPO import SPH_MPO

# recover the macro-geometry
geom_text='''
   CAR2D 10 10 (*ASSEMBLY 19 X 19*)
   X- DIAG X+ REFL
   Y- SYME Y+ DIAG
   MESHX 0.0 1.26502 2.53004 3.79506 5.06008 6.3251
         7.59012 8.85514 10.12016 11.38518 11.43398
   MIX  1  2  3  4  5  6  7  8  9 10
          11 12 13 14 15 16 17 18 19
             20 21 22 23 24 25 26 27
                28 29 30 31 32 33 34
                   35 36 37 38 39 40
                      41 42 43 44 45
                         46 47 48 49
                            50 51 52
                               53 54
                                  55
'''

MySPH = SPH_MPO(geom_text, 'quick-mpo-oneParam-17-diag.hdf', 'RT', ipoly=0, icol=3)
MySPH.exec()

print("test Equivalence_SPH_MPO_DIF_RT0 completed")
```

### 1.3 The SPH_APEX script

The SPH_APEX utility perform a SPH equivalence on *each* calculation point of an APEX file. A new set of equivalence factors is added to the MEDIA_SPH group in the output APEX file, as depicted in Fig. 2. Here, EQV_DIFF_RT2_INT1 is a *G*-group dataset containing SPH factors. The remaining of the APEX file is not modified.


Figure 2: APEX enrichment with SPH factors.

The SPH_APEX utility, accessible from Python3, is imported by the command

```
from SPH_APEX import SPH_APEX
```

It has one constructor: `SPH_APEX()`, used to create an *object instance* `o` and one method to execute the utility function.

### *1.3.1 Attribute Variables*

A SPH_APEX object `o` contains eight attribute variables:

`o.geom_text` name of the `'''...'''`-delimited text object containing the definition of the macro-geometry, as specified in Sect. 3.3 of Ref 7.

`o.name` name of the input APEX file

`o.htype` string set to "`RT`" (Raviart-Thomas diffusion or SPN macro calculation) or "`SN`" (discrete ordinates macro calculation).

`o.ipoly` polynomial order in space. 0: RT0 or DD0; 1: RT1 or DD1, etc.

`o.icol` type of Raviart-Thomas integration in the macro-calculation. 1: analytical; 2: Gauss-Lobatto; 3: Gauss-Legendre.

`o.isn` order of angular quadrature with SN method. 2: S2; 4: S4; 6: S6; 8: S8, etc.

`o.ispn` order of angular quadrature with SPN method. 0: diffusion theory; 1: SP1, 3: SP3, 5: SP5, etc.

`o.iscat` scattering anisotropy order. 1: isotropic in LAB; 2: P1; 3: P2; 4: P2, etc.

### *1.3.2 SPH_APEX()*

```
o = SPH_APEX(geom_text, name, [htype], [ipoly], [icol], [isn], [ispn], [iscat])
```

| **input parameter:** | | |
|---|---|---|
| `geom_text` | *string* | name of the `'''...'''`-delimited text object containing the definition of the macro-geometry, as specified in Sect. 3.3 of Ref 7. |
| `name` | *string* | name of the input APEX file |
| `htype` | *string* | string set to "`RT`" (Raviart-Thomas diffusion or SPN macro calculation) or "`SN`" (discrete ordinates macro calculation). |
| `ipoly` | *int* | polynomial order in space. 0: RT0 or DD0; 1: RT1 or DD1, etc. |
| `icol` | *int* | type of Raviart-Thomas integration in the macro-calculation. 1: analytical; 2: Gauss-Lobatto; 3: Gauss-Legendre. Must be set only if `htype=RT`. |
| `isn` | *int* | order of angular quadrature with SN method. 2: S2; 4: S4; 6: S6; 8: S8, etc. Must be set only if `htype=SN`. |
| `ispn` | *int* | order of angular quadrature with SPN method. 0: diffusion theory; 1: SP1, 3: SP3, 5: SP5, etc. Must be set only if `htype=RT`. |
| `iscat` | *int* | scattering anisotropy order. 1: isotropic in LAB; 2: P1; 3: P2; 4: P2, etc. Must be set only if `htype=SN` or if `ispn`$> 0$. |

### *1.3.3 o.exec()*

This method execute the SPH procedure.

```
o.exec()
```

*1.3.4 example*

```
#
# Equivalence_SPH_APEX: incorporation of SPH factors in an APEX file
#
from SPH_APEX import SPH_APEX

# recover the macro-geometry
geom_text='''
   CAR2D 5 5 (*ASSEMBLY 5 X 5*)
   X- REFL X+ REFL
   Y- REFL Y+ REFL
   MESHX 0.0 1.26 2.52 3.78 5.04 6.3
   MESHY 0.0 1.26 2.52 3.78 5.04 6.3
   MIX
   6  5   4   5  6
   5  3   2   3  5

   4  2   1   2  4

   5  3   2   3  5
   6  5   4   5  6
'''

MySPH = SPH_APEX(geom_text, 'UOX_5x5_TG6_sym8_multiDom.h5', 'SN', ipoly=0, isn=8, \
   iscat=4)
MySPH.exec()

print("test Equivalence_SPH_APEX_S8P3 completed")
```

### 1.4 The `REFL_MPO` script

The `REFL_MPO` utility perform a 1D *reflector equivalence procedure* so as to produce a reflector MPO file, as depicted in Figs. 3 and 4. A fine-group 1D $S_{16}$ $P_1$ reference calculation is first performed over the complete geometry, including the *residual reflector*. Many equivalence techniques are available. This utility script takes a fine-group *feeding assembly* MPO file at input and produces a coarse-group reflector MPO file at output. The reference 1D SN geometry and the macro-geometry are depicted in Fig. 5.

The `REFL_MPO` utility, accessible from Python3, is imported by the command

```
from REFL_MPO import REFL_MPO
```

It has one constructor: `REFL_MPO()`, used to create an *object instance* `o` and one method to execute the utility function.

*1.4.1 Attribute Variables*

A REFL_MPO object `o` contains thirteen attribute variables:

`o.geom_text` name of the `'''...'''`-delimited text object containing the definition of the 1D reference SN geometry, as specified in Sect. 3.3 of Ref 7.

`o.nameFeed` name of the input MPO file containing the fine-group feeding assembly data.

`o.nameRefl` name of the output MPO file containing the coarse-group reflector data.

`o.palier` string describing the type of reflector.

`o.htype` string describing the type of equivalence.

`o.nlf` order of SPN approximation (odd integer). Set to 1 for diffusion theory.

`o.FVol` LCM object containing volume fractions of the reflector materials in 8 mixtures.

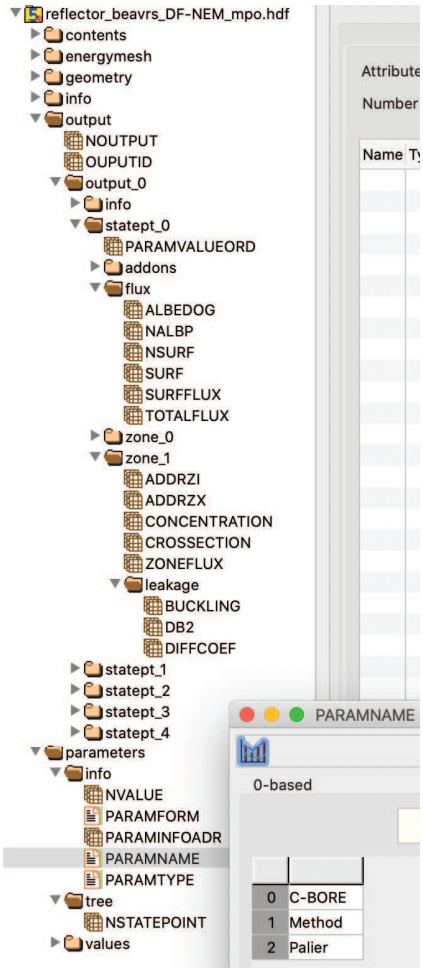


Figure 3: reflector MPO file for the Beavrs benchmark – DF-NEM equivalence

`o.Param1` LCM object containing global parameters used to interpolate cross sections in the feeding assembly.

`o.Param2` LCM object containing a second set of global parameters used to interpolate cross sections in the feeding assembly.

`o.sph` boolean variable set to True to transform discontinuity factors into SPH factors and to correct the reflector properties accordingly.

`o.noal` boolean variable set to True to force the reflector albedo to zero on the right-most coordinate.

`o.LibType` type of cross-section library used for the reflector. `o.LibType='CLA99CEA93'` by default.

`o.NuclData` name of the cross-section library used for the reflector as set in the `.access script` file. `o.NuclData='CLA99CEA93:CLA99CEA93_SS'` by default.

The fine-group reference calculation in the reflector uses microscopic cross-section data recovered from files `CLA99CEA93` and `CLA99CEA93_SS` located in the `libraries` directory and recovered with the `.access script` file. The `libraries` directory is generally located on the same directory as PyKit.

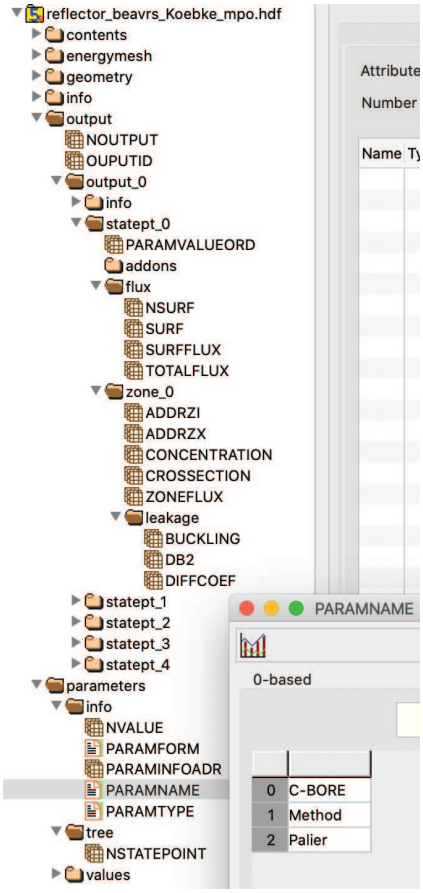


Figure 4: reflector MPO file for the Beavrs benchmark – KOEBKE equivalence

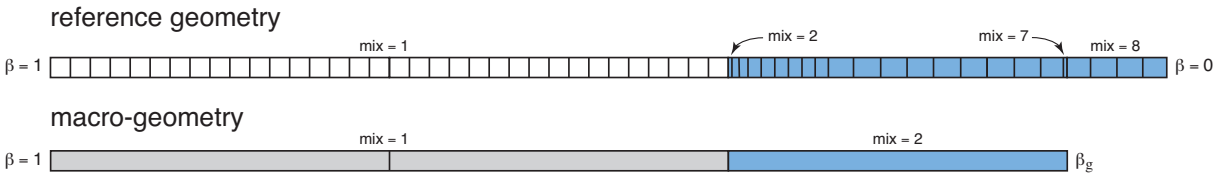


Figure 5: Reference and macro geometries of a 1D reflector model.

### *1.4.2 REFL_MPO()*

```
o = REFL_MPO(geom_text, nameFeed, nameRefl, palier, htype, nlf, FVol, Param, [Param2], \
    [sph], [noal])
```

| **input parameter:** | | |
|---|---|---|
| `geom_text` | *string* | name of the `'''...'''`-delimited text object containing the definition of the 1D reference SN geometry, as specified in Sect. 3.3 of Ref 7. |
| `nameFeed` | *string* | name of the input MPO file containing the fine-group feeding assembly data. |
| `nameRefl` | *string* | name of the output MPO file containing the coarse-group reflector data. |
| `palier` | *string* | string describing the type of reflector. |
| `htype` | *string* | string describing the type of equivalence. `DF-NEM`: equivalence with the nodal expansion method;[9, 10] `DF-ANM`: equivalence with the analytic nodal method; `DF-RT`: equivalence with the Raviart-Thomas finite element method for diffusion theory; `DF-RT-SPH`: equivalence with the Raviart-Thomas finite element method for the $SP_n$ method; `KOEBKE`: Koebke 2-group equivalence; `LEFEBVRE-LEB`: Lefebvre-Lebigot 2-group equivalence.[11] |
| `nlf` | *int* | order of SPN approximation (odd integer). Set to 1 for diffusion theory. |
| `FVol` | *LCM* | LCM object containing volume fractions of the reflector materials in 8 mixtures. `FVOL(1)`=0 corresponds to the feeding assembly; `FVOL(2)` corresponds to a small gap between the feeding assembly and the reflector; `FVOL(7)` corresponds to a small gap between the right-most coordinate of the reflector and the reflector residual; `FVOL(8)` corresponds to the reflector residual. Allowed materials are: `H2O`, `Zr4` (Zirconium 4), `Inc` (Inconel), `SS` (stainless steel 304) and He (void). |
| `Param1` | *LCM* | LCM object containing global parameters used to interpolate cross sections in the feeding assembly. |
| `Param2` | *LCM* | LCM object containing a second set of global parameters used to interpolate cross sections in the feeding assembly. This parameter is required with `KOEBKE` and `LEFEBVRE-LEB` response matrix equivalence techniques. |
| `sph` | *bool* | boolean variable set to True to transform discontinuity factors into SPH factors and to correct the reflector properties accordingly. Default = False. |
| `noal` | *bool* | boolean variable set to True to force the reflector albedo to zero on the right-most coordinate. Default = False. |

### *1.4.3 o.exec()*

This method execute the reflector equivalence procedure.

```
o.exec()
```

### *1.4.4 DF-NEM BEAVRS example*

The BEAVRS benchmark is a water reflector with stainless steel plates, typical of second generation pressurized water reactors. Geometries corresponding to a `DF-NEM` equivalence are depicted in Fig. 6.

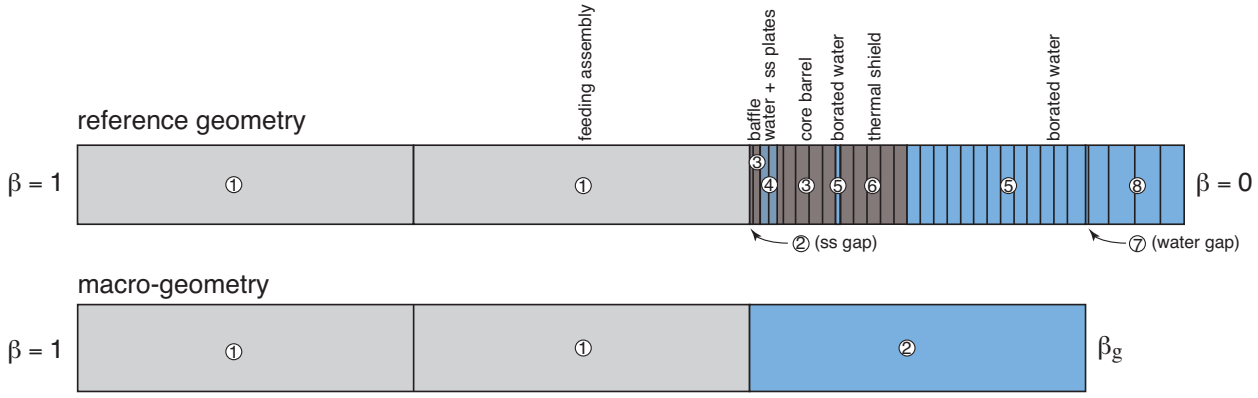


Figure 6: Geometries of the Beavrs reflector – DF-NEM equivalence.

```
#
# Nodal Expansion Reflector DF-NEM equivalence prodedure for the BEAVRS benchmark
#
from REFL_MPO import REFL_MPO
import lcm
import numpy as np

# recover the reflector macro-geometry
geom_text='''
   CAR1D 12
   X- REFL X+ VOID
   MESHX -43.0 -21.5 -5.0 0.0 0.005 2.2225 3.9825 9.6975 10.1975 15.9125 21.5 21.505
         35.9125
   MIX     1  1  1     2        3         4         3          5          6          5
           !  fuel   gap       ss    ss+h2o        ss        h2o  ss_shield        h2o
                7            8
           !   gap          h2o
   SPLITX  20 15 20  1         25        24        10          5          6          5
                1            5
'''

# Define volume fractions in 8 mixtures
# mix=1 : feeding assembly (first fraction is always set to 0)
# mix=2 : fuel-reflector gap
# mix=7 : right-most gap to compute albedo
# mix=8 : residual right reflector
FVol = lcm.new('LCM','volume_fractions')
fvacier = 0.05
fveau = 1.0 - fvacier
FVol['H2O'] = np.array([0., 0., 0., fveau,   1., 0., 1., 1.], dtype='f')
FVol['SS']  = np.array([0., 1., 1., fvacier, 0., 1., 0., 0.], dtype='f')
FVol.close() # close without erasing

# Define feeding assembly parameters (0 to 3 parameters allowed)
Param = lcm.new('LCM','global_parameters')
Param.lis('PARAMNAME',1)
Param['PARAMNAME'][0] = 'C-BORE'
Param['PARAMVALU'] = np.array([975. ,], dtype='f')
Param.close() # close without erasing

MyREFL = REFL_MPO(geom_text, 'assbly_caseA_mpo_boron.hdf', \
    'reflector_beavrs_DF-NEM_mpo.hdf', 'BEAVRS', 'DF-NEM', 1, FVol, Param)
MyREFL.exec()
print("test Reflector_beavrs_DF-NEM completed")
```

### *1.4.5 DF-NEM EPR example*

The EPR example is a heavy reflector, typical of third generation pressurized water reactors. Geometries corresponding to a `DF-NEM` equivalence are depicted in Fig. 7.

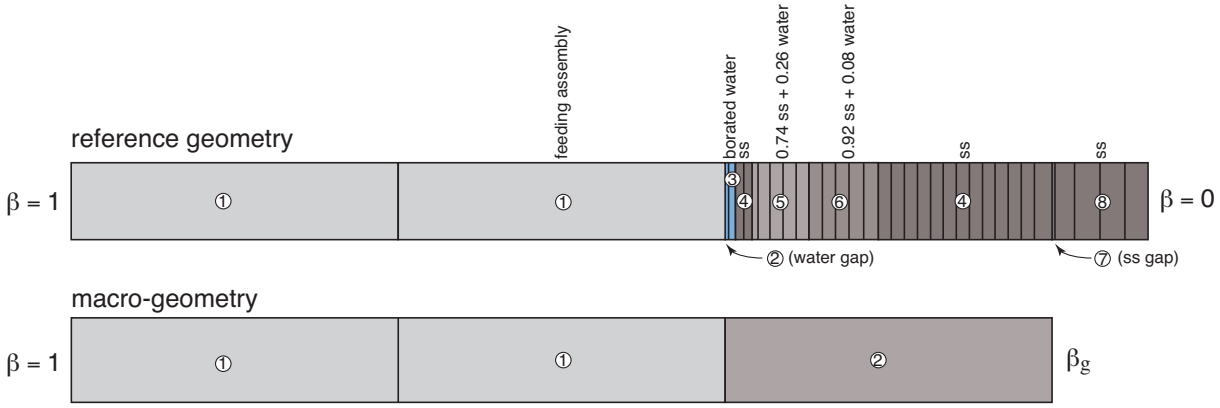


Figure 7: Geometries of the EPR reflector – DF-NEM equivalence.

```
#
# Nodal Expansion Reflector DF-NEM equivalence prodedure for the EPR
#
from REFL_MPO import REFL_MPO
import lcm
import numpy as np

# recover the reflector macro-geometry
geom_text='''
  CAR1D 11
  X- ALBE 1.0 X+ VOID
  MESHX -43.0 -21.5 -5.0 0.0 0.005 0.1 2.38 3.68 11.68 21.5 21.505 30.12
  MIX 1 1 1 2 3 4 5 6 4 7 8
  SPLITX 20 15 20 1 3 10 5 8 9 1 8
'''

# Define volume fractions in 8 mixtures
# mix=1 : feeding assembly (first fraction is always set to 0)
# mix=2 : fuel-reflector gap
# mix=7 : right-most gap to compute albedo
# mix=8 : residual right reflector
FVol = lcm.new('LCM','volume_fractions')
fvaci1 = 0.74
fveau1 = 1.0 - fvaci1
fvaci2 = 0.92
fveau2 = 1.0 - fvaci2
FVol['H2O'] = np.array([0., 1., 1., 0., fveau1, fveau2, 0., 0.], dtype='f')
FVol['SS']  = np.array([0., 0., 0., 1., fvaci1, fvaci2, 1., 1.], dtype='f')
FVol.close() # close without erasing

# Define feeding assembly parameters (0 to 3 parameters allowed)
Param = lcm.new('LCM','global_parameters')
Param.lis('PARAMNAME',2)
Param['PARAMNAME'][0] = 'BURN'
Param['PARAMNAME'][1] = 'C_B10_WATER'
Param['PARAMVALU'] = np.array([0., 4.7402E-6], dtype='f')
Param.close() # close without erasing

MyREFL = REFL_MPO(geom_text, 'Ass_3GLE_420_16Gd_99g-mpo.hdf', \
   'Reflector_EPR_DF-NEM_mpo.hdf', 'BEAVRS', 'DF-NEM', 1, FVol, Param)
MyREFL.exec()
print("test Reflector_EPR_DF-NEM completed")
```

### *1.4.6 KOEBKE BEAVRS example*

The BEAVRS benchmark is a water reflector with stainless steel plates, typical of second generation pressurized water reactors. Geometries corresponding to a `KOEBKE` equivalence are depicted in Fig. 8.

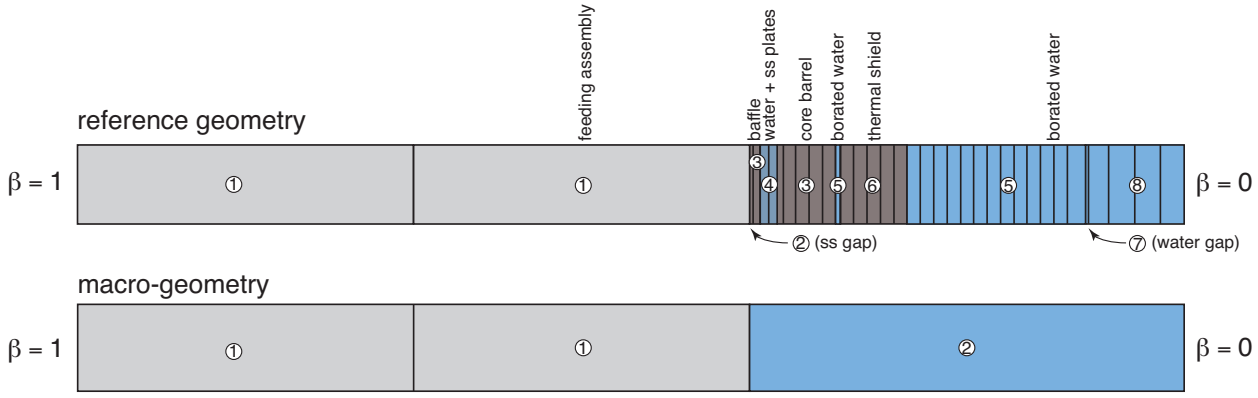


Figure 8: Geometries of the Beavrs reflector – KOEBKE equivalence.

```
#
# Nodal Expansion Reflector Koebke equivalence prodedure for the BEAVRS benchmark
#
from REFL_MPO import REFL_MPO
import lcm
import numpy as np

# recover the reflector macro-geometry
geom_text='''
   CAR1D 12
   X- REFL X+ VOID
   MESHX -43.0 -21.5 -5.0 0.0 0.005 2.2225 3.9825 9.6975 10.1975 15.9125 21.5 21.505
         35.9125
   MIX     1  1  1     2        3         4         3          5          6          5
           !  fuel   gap       ss    ss+h2o        ss        h2o  ss_shield        h2o
                 7            8
           !   gap          h2o
   SPLITX  20 15 20  1         25        24        10          5          6          5
                 1            5
'''

# Define volume fractions in 8 mixtures
# mix=1 : feeding assembly (first fraction is always set to 0)
# mix=2 : fuel-reflector gap
# mix=7 : right-most gap to compute albedo
# mix=8 : residual right reflector
FVol = lcm.new('LCM','volume_fractions')
fvacier = 0.05
fveau = 1.0 - fvacier
FVol['H2O'] = np.array([0., 0., 0., fveau,   1., 0., 1., 1.], dtype='f')
FVol['SS']  = np.array([0., 1., 1., fvacier, 0., 1., 0., 0.], dtype='f')
FVol.close() # close without erasing

# Define first feeding assembly parameters (0 to 3 parameters allowed)
Param1 = lcm.new('LCM','global_parameters')
Param1.lis('PARAMNAME',1)
Param1['PARAMNAME'][0] = 'C-BORE'
Param1['PARAMVALU'] = np.array([500. ,], dtype='f')
Param1.close() # close without erasing

# Define second feeding assembly parameters (0 to 3 parameters allowed)
Param2 = lcm.new('LCM','global_parameters')
Param2.lis('PARAMNAME',1)
Param2['PARAMNAME'][0] = 'C-BORE'
Param2['PARAMVALU'] = np.array([0. ,], dtype='f')
Param2.close() # close without erasing

MyREFL = REFL_MPO(geom_text, 'assbly_caseA_mpo_boron.hdf', \
   Reflector_beavrs_Koebke_mpo.hdf', 'BEAVRS', 'KOEBKE', 1 FVol, Param1, Param2)
MyREFL.exec()
print("test Reflector_beavrs_Koebke completed")
```

### 1.5 The `REFL_APEX` script

The `REFL_APEX` utility perform a 1D *reflector equivalence procedure* so as to produce a reflector APEX file, as depicted in Fig. 9. A fine-group 1D $S_{16}$ $P_1$ reference calculation is first performed over the complete geometry, including the *residual reflector*. Many equivalence techniques are available. This utility script takes a fine-group *feeding assembly* APEX file at input and produces a coarse-group reflector APEX file at output. The reference 1D SN geometry and the macro-geometry are depicted in Fig. 10.

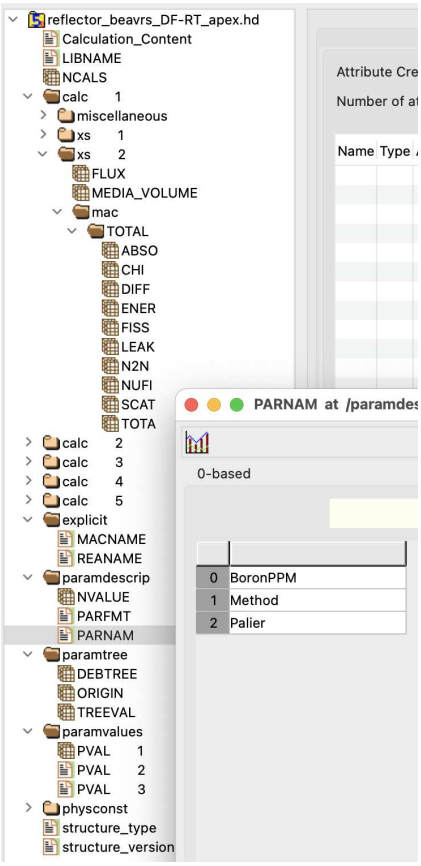


Figure 9: reflector APEX file for the Beavrs benchmark – DF-RT equivalence

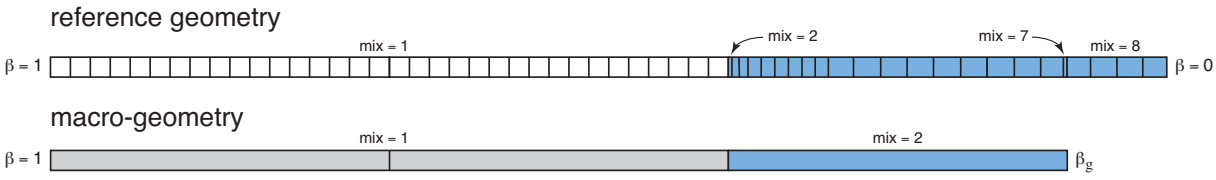


Figure 10: Reference and macro geometries of a 1D reflector model.

The `REFL_APEX` utility, accessible from Python3, is imported by the command

```
from REFL_APEX import REFL_APEX
```

It has one constructor: `REFL_APEX()`, used to create an *object instance* `o` and one method to execute the utility function.

*1.5.1 Attribute Variables*

A REFL_APEX object `o` contains thirteen attribute variables:

`o.geom_text` name of the `'''...'''`-delimited text object containing the definition of the 1D reference SN geometry, as specified in Sect. 3.3 of Ref 7.

`o.nameFeed` name of the input APEX file containing the fine-group feeding assembly data.

`o.nameRefl` name of the output APEX file containing the coarse-group reflector data.

`o.palier` string describing the type of reflector.

`o.htype` string describing the type of equivalence.

`o.nlf` order of SPN approximation (odd integer). Set to 1 for diffusion theory.

`o.FVol` LCM object containing volume fractions of the reflector materials in 8 mixtures.

`o.Param1` LCM object containing global parameters used to interpolate cross sections in the feeding assembly.

`o.Param2` LCM object containing a second set of global parameters used to interpolate cross sections in the feeding assembly.

`o.sph` boolean variable set to True to transform discontinuity factors into SPH factors and to correct the reflector properties accordingly.

`o.noal` boolean variable set to True to force the reflector albedo to zero on the right-most coordinate.

`o.LibType` type of cross-section library used for the reflector. `o.LibType='CLA99CEA93'` by default.

`o.NuclData` name of the cross-section library used for the reflector as set in the `.access script` file. `o.NuclData='CLA99CEA93:CLA99CEA93_SS'` by default.

The fine-group reference calculation in the reflector uses microscopic cross-section data recovered from file `CEAV5_1.G281.V5.1.xsm` located in the `libraries` directory and recovered with the `.access script` file. The `libraries` directory is generally located on the same directory as PyKit.

### *1.5.2 REFL_APEX()*

```
o = REFL_APEX(geom_text, nameFeed, nameRefl, palier, htype, nlf, FVol, Param, [Param2], \
    [sph], [noal])
```

| **input parameter:** | | |
|---|---|---|
| `geom_text` | *string* | name of the `'''...'''`-delimited text object containing the definition of the 1D reference SN geometry, as specified in Sect. 3.3 of Ref 7. |
| `nameFeed` | *string* | name of the input APEX file containing the fine-group feeding assembly data. |
| `nameRefl` | *string* | name of the output APEX file containing the coarse-group reflector data. |
| `palier` | *string* | string describing the type of reflector. |
| `htype` | *string* | string describing the type of equivalence. `DF-NEM`: equivalence with the nodal expansion method;[9, 10] `DF-ANM`: equivalence with the analytic nodal method; `DF-RT`: equivalence with the Raviart-Thomas finite element method for diffusion theory; `DF-RT-SPH`: equivalence with the Raviart-Thomas finite element method for the $SP_n$ method; `KOEBKE`: Koebke 2-group equivalence; `LEFEBVRE-LEB`: Lefebvre-Lebigot 2-group equivalence.[11] |
| `nlf` | *int* | order of SPN approximation (odd integer). Set to 1 for diffusion theory. |
| `FVol` | *LCM* | LCM object containing volume fractions of the reflector materials in 8 mixtures. `FVOL(1)`=0 corresponds to the feeding assembly; `FVOL(2)` corresponds to a small gap between the feeding assembly and the reflector; `FVOL(7)` corresponds to a small gap between the right-most coordinate of the reflector and the reflector residual; `FVOL(8)` corresponds to the reflector residual. Allowed materials are: `H2O`, `Zr4` (Zirconium 4), `Inc` (Inconel), `SS` (stainless steel 304) and He (void). |
| `Param1` | *LCM* | LCM object containing global parameters used to interpolate cross sections in the feeding assembly. |
| `Param2` | *LCM* | LCM object containing a second set of global parameters used to interpolate cross sections in the feeding assembly. This parameter is required with `KOEBKE` and `LEFEBVRE-LEB` response matrix equivalence techniques. |
| `sph` | *bool* | boolean variable set to True to transform discontinuity factors into SPH factors and to correct the reflector properties accordingly. Default = False. |
| `noal` | *bool* | boolean variable set to True to force the reflector albedo to zero on the right-most coordinate. Default = False. |

### *1.5.3 o.exec()*

This method execute the reflector equivalence procedure.

```
o.exec()
```

### *1.5.4 DF-RT BEAVRS example*

The BEAVRS benchmark is a water reflector with stainless steel plates, typical of second generation pressurized water reactors. Geometries corresponding to a `DF-RT` equivalence are depicted in Fig. 11.

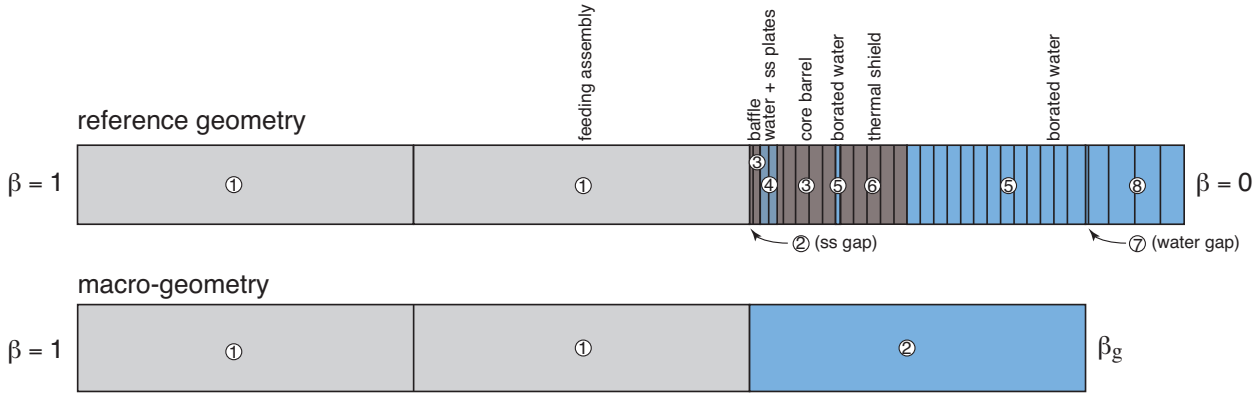


Figure 11: Geometries of the Beavrs reflector – DF-RT equivalence.

```
#
# Nodal Expansion Reflector DF-RT equivalence prodedure for the BEAVRS benchmark
#
from REFL_APEX import REFL_APEX
import lcm
import numpy as np

# recover the reflector macro-geometry
geom_text='''
   CAR1D 12
   X- REFL X+ VOID
   MESHX -43.0 -21.5 -5.0 0.0 0.005 2.2225 3.9825 9.6975 10.1975 15.9125 21.5 21.505
         35.9125
   MIX     1  1  1     2        3         4         3          5          6          5
           !  fuel   gap       ss    ss+h2o        ss        h2o  ss_shield        h2o
                7            8
           !   gap          h2o
   SPLITX  20 15 20  1         25        24        10          5          6          5
                1            5
'''

# Define volume fractions in 8 mixtures
# mix=1 : feeding assembly (first fraction is always set to 0)
# mix=2 : fuel-reflector gap
# mix=7 : right-most gap to compute albedo
# mix=8 : residual right reflector
FVol = lcm.new('LCM','volume_fractions')
fvacier = 0.05
fveau = 1.0 - fvacier
FVol['H2O'] = np.array([0., 0., 0., fveau,   1., 0., 1., 1.], dtype='f')
FVol['SS']  = np.array([0., 1., 1., fvacier, 0., 1., 0., 0.], dtype='f')
FVol.close() # close without erasing

# Define feeding assembly parameters (0 to 3 parameters allowed)
Param = lcm.new('LCM','global_parameters')
Param.lis('PARAMNAME',1)
Param['PARAMNAME'][0] = 'BoronPPM'
Param['PARAMVALU'] = np.array([975. ,], dtype='f')
Param.close() # close without erasing

MyREFL = REFL_APEX(geom_text, 'assbly_caseA_apex.hdf', \
   'reflector_beavrs_DF-RT_apex.hdf', 'BEAVRS', 'DF-RT', 1, FVol, Param)
MyREFL.exec()
print("test Reflector_beavrs_APEX_DF-RT completed")
```

## 1.6 The `Concat_MPO` script

The `Concat_MPO` utility perform the concatenation of two MPO files, taking into account the reorganization of global parameters and the update of `parameters/values` datasets.

The `Concat_MPO` utility, accessible from Python3, is imported by the command

```
from Concat_MPO import Concat_MPO
```

It has one constructor: `Concat_MPO()`, used to create an *object instance* `o` and one method to execute the utility function.

### *1.6.1 Attribute Variables*

A Concat_MPO object `o` contains three attribute variables:

`o.MyMpo1` name of the first input MPO file.

`o.MyMpo2` name of the second input MPO file.

`o.MyMpoCat` name of the concatenated output MPO file.

### *1.6.2 Concat_MPO()*

```
o = Concat_MPO(MyMpo1, MyMpo2, MyMpoCat)
```

| **input parameter:** | | |
|---|---|---|
| `MyMpo1` | *string* | name of the first input MPO file. |
| `MyMpo2` | *string* | name of the second input MPO file. |
| `MyMpoCat` | *string* | name of the concatenated output MPO file. |

### *1.6.3 o.exec()*

This method execute the concatenation procedure.

```
o.exec()
```

### *1.6.4 Concatenation example*

In this example, two MPO files are concatenated:

```
#
# Concatenation of two MPO files
#
from Concat_MPO import Concat_MPO
#
# Perform concatenation
MyConcat_Proc = Concat_MPO('MyMpo1.hdf', 'MyMpo2.hdf', 'MyMpoCat.hdf')
MyConcat_Proc.exec()
#
print("test pincell_mpo_concat completed")
```

## 1.7 The `Concat_APEX` script

The `Concat_APEX` utility perform the concatenation of two Apex files, taking into account the reorganization of global parameters and the update of `paramvalues` datasets.

The `Concat_APEX` utility, accessible from Python3, is imported by the command

```
from Concat_APEX import Concat_APEX
```

It has one constructor: `Concat_APEX()`, used to create an *object instance* `o` and one method to execute the utility function.

### *1.7.1 Attribute Variables*

A Concat_APEX object `o` contains three attribute variables:

`o.MyApex1` name of the first input Apex file.

`o.MyApex2` name of the second input Apex file.

`o.MyApexCat` name of the concatenated output Apex file.

### *1.7.2 Concat_APEX()*

```
o = Concat_MPO(MyApex1, MyApex2, MyApexCat)
```

| **input parameter:** | | |
|---|---|---|
| `MyApex1` | *string* | name of the first input Apex file. |
| `MyApex2` | *string* | name of the second input Apex file. |
| `MyApexCat` | *string* | name of the concatenated output Apex file. |

### *1.7.3 o.exec()*

This method execute the concatenation procedure.

```
o.exec()
```

### *1.7.4 Concatenation example*

In this example, two Apex files are concatenated:

```
#
# Concatenation of two Apex files
#
from Concat_APEX import Concat_APEX
#
# Perform concatenation
MyConcat_Proc = Concat_APEX('MyApex1.hdf', 'MyApex2.hdf', 'MyApexCat.hdf')
MyConcat_Proc.exec()
#
print("test pincell_apex_concat completed")
```

## References

[1] A. Janet, "APOLLO2-A User's Manual Version 1.26.0," Report D02-DTIPD-F-17-0503, CONV17_0026, Framatome, 2018.

[2] I. Zmijarevic, N. Huot, F. Auffret and P. Bellier, "Description of the APOLLO3 Multi parameter Output Library for the version AP3-2.0," Report DEN/DANS/DM2S/SERMA/LTSD/RT/17-6237/A , Commissariat À l'Énergie Atomique, 2017.

[3] S. Loubière, R. Sanchez, M. Coste, A. Hébert, Z. Stankovski, C. Van Der Gucht and I. Zmijarevic, "APOLLO2, Twelve Years Later," paper presented at the *Int. Conf. on Mathematics and Computation, Reactor Physics and Environmental Analysis in Nuclear Applications*, Madrid, Spain, September 27–30, 1999.

[4] D. Schneider *et al.*,"APOLLO3: CEA/DEN Deterministic Multi-Purpose Code for Reactor Physics Analysis," *PHYSOR 2016 – Int. Conf. on Advances in Reactor Physics for Unifying Theory and Experiments in the 21st Century*, Sun Valley, Idaho, USA, May 1–5, 2016.

[5] The HDF Group, https://www.hdfgroup.org.

[6] A. Hébert and R. Roy, "The GANLIB5 kernel guide (64–bit clean version)," Report IGE-332, Polytechnique Montréal (2022). See the home page at http://merlin.polymtl.ca/downloads/IGE332.pdf

[7] G. Marleau, A. Hébert and R. Roy, "A user guide for DRAGON version5 (64–bit clean version)," Report IGE-335, Polytechnique Montréal (2022). See the home page at http://merlin.polymtl.ca/downloads/IGE335.pdf

[8] A. Hébert, D. Sekki and R. Chambon, "A user guide for DONJON version5 (64–bit clean version)," Report IGE-344, Polytechnique Montréal (2022). See the home page at http://merlin.polymtl.ca/downloads/IGE344.pdf

[9] M. Gamarino, "Modal methods for rehomogenization of nodal cross sections in nuclear reactor core analysis," Ph. D. thesis, Delft University of Technology (2018).

[10] A. Hébert, "The Brisingr theory and user guide," Report IGE-380, Polytechnique Montréal (2022).

[11] K. Fröhlicher, V. Salino and A. Hébert, "Investigating fission distribution behavior under various homogenization techniques for asymmetrical fuel assemblies and different reflector equivalence methods," *Ann. nucl. Energy*, **157**, 1–12 (2021).