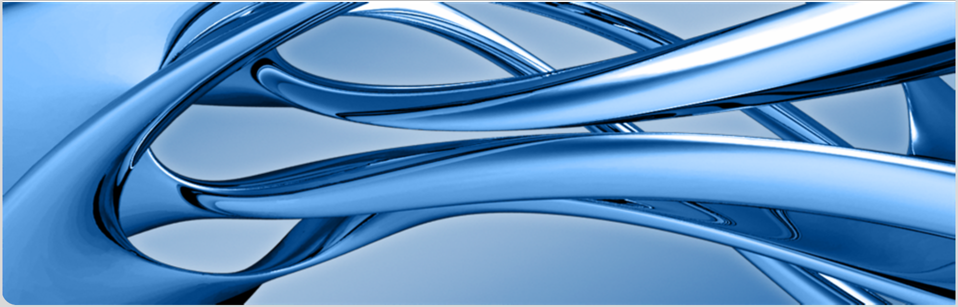
Python Interfaces for Reactor Simulations: Concept, ways to use, examples

INR Seminar ”Nukleare Energieerzeugung”

Anton Travleev | November 9, 2014

INSTITUTE FOR NEUTRON PHYSICS AND REACTOR TECHNOLOGY



Outline

Outline

1. Introduction to PIRS

What is PIRS

PIRS: Python Interfaces for Reactor Simulations

A package for Python programming language, to facilitate interaction with reactor calculation codes.

Python

- www.python.org
- Free
- Interpreted: cross-platform but slow
- Big community: lot of ready-to-use solutions

Interaction with code

- Model description
- Generation of Input file(s)
- Job submission
- Reading of calculation results

What is PIRS

PIRS: Python Interfaces for Reactor Simulations

A package for Python programming language, to facilitate interaction with reactor calculation codes.

Python

- www.python.org
- Free
- Interpreted: cross-platform but slow
- Big community: lot of ready-to-use solutions

Interaction with code

- Model description
- Generation of Input file(s)
- Job submission
- Reading of calculation results

What is PIRS

PIRS: Python Interfaces for Reactor Simulations

A package for Python programming language, to facilitate interaction with reactor calculation codes.

Python

- www.python.org
- Free
- Interpreted: cross-platform but slow
- Big community: lot of ready-to-use solutions

Interaction with code

- Model description
- Generation of Input file(s)
- Job submission
- Reading of calculation results

Background

Prerequisites

- Author's Experience with Python and MCNP
- HPMC project: goal to couple Monte-Carlo with TH for PWR-like geometries from pin to reactor core
- Previous coupling schemes – mix of programming languages (from shell to fortran), higly specific to geometry

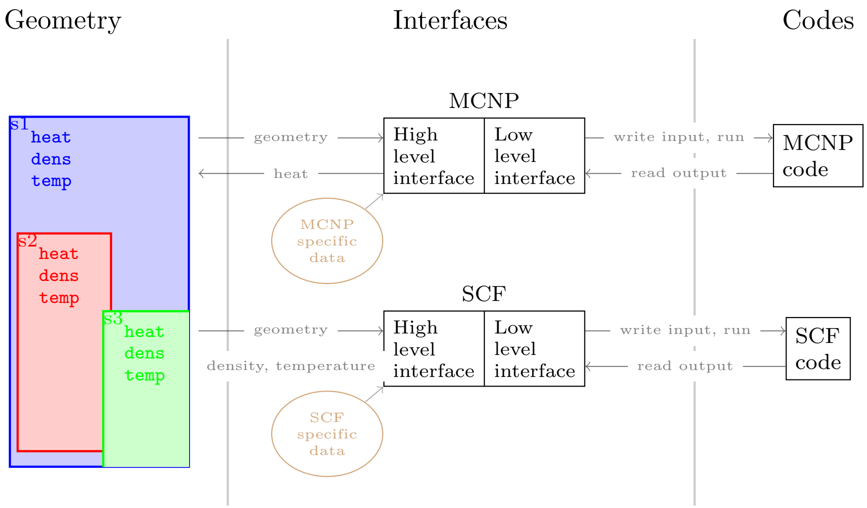
Idea

- Light-weight: e.g. to deploy on cluster's local account
- General tool: geometry-independent, not only for coupled calculations
- Everything should be specified in terms of a programming language

PIRS concept

- Geometry definition: should be reusable in all codes, defined in terms, independent on any particular code.
- Convenien way to handle distribution of dependent variables (heat deposition, temperature and material density axial distributions)
- Interface consists of two parts:
  - Low-level interface: ”knows” syntax of input file(s), can read output files, can start code and wait until it completes
  - High-level interface: converts code-independent geometry together with additionally specified code-specific data to low-level interfaces, and puts results of calculations back to code-independent geometry.

PIRS concept scheme



PIRS classes and functions

Geometry construction

```
from pirs.solids import Cylinder, Box
from pirs.solids import zmesh
```

Classes to represent solids – basic elements to describe geometry and axial meshes for dependent variables.

High-level interfaces

```
from pirs import McnpInterface
from pirs import ScfInterface
```

PIRS classes and functions

Low-level interface to MCNP

```
from pirs.mcnp import Material, MaterialCollection
from pirs.mcnp import MeshTally, TallyCollection
from pirs.mcnp import Xsdir
from pirs.mcnp import Surface, Volume, SurfaceCollection
from pirs.mcnp import Cell, Model
```

Classes to represent data for MCNP input file: mateirals, tallies, surfaces, cells, etc.

Low-level interface to SCF

```
from pirs.scf2 import Input, RodMaterial
from pirs.scf2.variables import ScfVariable, ScfTable
from pirs.scf2 import read_output, OutputTable
```

Classes to represent data for SCF input file: variables, tables, switches, etc.

PIRS classes and functions

Tools

```
from pirs.tools import LoadMap
from pirs.tools import load, dump
from pirs.tools.plots import MeshPlotter, colormap
```

Pseudo-graphics definition of core loading maps, functions to dump current calculational state to hard drive and to read it; functions to plot geometry and distribution of variables.

Base classes

```
from pirs.core.tramat import Nuclide, Mixture, zai
from pirs.core.trageom import Vector3, pi, pi2
from pirs.core.scheduler import Job, Scheduler
from pirs.core.scheduler enva, WorkPlace, InputFile
```

Parent classes used in PIRS in several places. Not needed to end-user.

Outline

2 PIRS installation and setup

Dependencies

Python interpreter

- PIRS is developed with Python 2.7 and tested with Python 2.6.
- Python 3.x: ?
- Linux distributions usually have Python 2.6 or 2.7 preinstalled. Under Windows, administrator rights are necessary to install Python.

Optional third-party packages

- uncertainties package, http://pythonhosted.org/uncertainties/. To handle results of Monte-Carlo calculations.
- Matplotlib package, http://matplotlib.org/. To generate geometry and result plots.

PIRS setup

Install package

```
$> tar -xzf pirs-X.Y.Z.tar.gz
$> cd pirs-X.Y.Z
$> python setup.py install --user
```

The `--user` option to install locally

Define environmental variables

```
$> export DATAPATH=/path/to/folder/with/xsdir
$> export MCNP=/path/to/mcnp/executable
$> export SCF=/path/to/scf/executable
```

- `$DATAPATH`: Path to default xsdir used to define available cross-sections.
- `$MCNP` and `$SCF`: paths to code executables.

Outline

3 Examples

K-inf in nat. U

```
from pirs.solids import Box
from pirs.mcnp import Material
from pirs import McnpInterface

# GEOMETRY
b = Box(material='m1')
b.dens.set_values(18.8)

# MCNP-SPECIFIC DATA
i = McnpInterface(b)
i.materials['m1'] = Material('U')  # material
i.bc['axial'] = '*'                # b.c.
i.bc['radial'] = '*'
i.adc.append('ksrc 0 0 0')         # kcode source
i.adc.append('kcode 100 1 30 100')

if __name__ == '__main__':
    # RUN MCNP
    i.run('R')                 # start MCNP
    print 'K-inf:', i.keff()   # print Keff
```

- Default box dimensions 1x1x1 cm
- `b.dens` represents density axial distribution.
- general model contain material names, which meaning/properties are specified in the code interface.
- `McnpInterface` – MCNP high-level interface.
- `Material` – part of low-level interface.

Output

```
   MCNP input file generated in 0.000495910644531 seconds
Started mcnp <job cd 'mcnp0'; ./batch.bat> mode=R, kwargs={}
scheduler queued job <job cd 'mcnp0'; ./batch.bat>
scheduler waits with parameters {'files': ['i_o'], 'llines': ['^ mcnp  *version .
Thu Sep 25 13:30:44 2014
     mcnp0/i_o doesn't exist
Thu Sep 25 13:30:49 2014
     mcnp0/i_o exists last line mismatch
Thu Sep 25 13:30:54 2014
     mcnp0/i_o exists last line matches
   MCNP run took 15.0220649242 seconds
K-inf: [0.417872, 0.00414795]
```

MCNP input file

```
MESSAGE:  datapath=/home/data/mcnp/all_jeff

c title
1 1 -18.8 -1 imp:n=1 tmp=2.526174e-08
2 0  1 imp:n=0 tmp=2.526174e-08

c surfaces
*1 rpp  -0.5  0.5  -0.5  0.5  -0.5  0.5

c data cards
c materials
m1
     92235.31c  7.20400e-03
     92234.31c  5.40000e-05
     92238.31c  9.92742e-01
c tallies
c kcode 500  1.0  20  100  j  j  100000  j
prdmp j j 1
ksrc 0 0 0
kcode 100 1 30 100
```

- Cell, surface and material numbers assigned automatically
- Macrobodies are used when possible
- Cross-section suffix chosen for default temperature from existing xs in xsdir

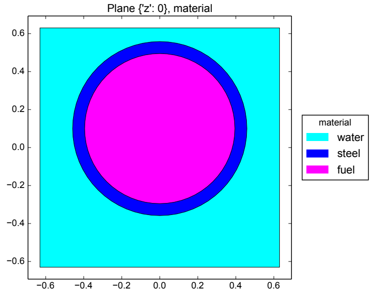
Geometry of single pin cell

```
from pirs.solids import Box, Cylinder

# surrounding water
b = Box(material='water')
b.X = 1.26
b.Y = b.X
b.Z = 400

# clad
c = Cylinder(material='steel')
c.R = 0.4583
c.Z = 360

# fuel
f = Cylinder(material='fuel')
f.R = 0.3951
f.Z = 350

# construct model
b.insert(c)     # put clad into box
c.insert(f)     # put fuel into clad
c.pos.y = 0.1   # shift clad with resp. to container

if __name__ == '__main__':
    from pirs.tools.plots import colormap
    colormap(b, {'z':0}, filename='ex2z.pdf')
    colormap(b, {'x':0}, filename='ex2x.pdf', aspect='auto')
```

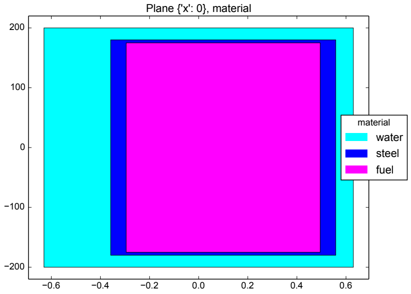
- Solid's dimensions can be specified using correspondent attributes
- Solid can be inserted into another
- Solid can be positioned with respect to its container
- `colormap` function uses Matplotlib

Plots

z plane



x plane

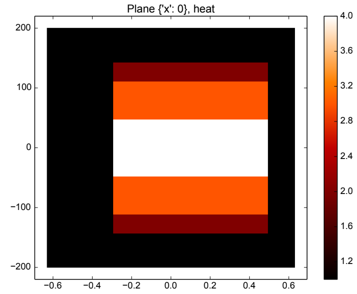


Axial distributions

```python
from ex2_geom import b

f = b.get_child((0, 0))

# fuel temperature axial profile
f.temp.set_grid([1, 1, 1])
f.temp.set_values(350)

# heat deposition axial profile
f.heat.set_grid([1, 1, 2, 3, 2, 1, 1])
f.heat.set_values([1, 2, 3, 4, 3, 2, 1])

# density axial profile in water
b.dens.set_grid([1, 1, 1])
b.dens.set_values([0.7, 0.65, 0.6])

if __name__ == '__main__':
    from pirs.tools.plots import colormap
    colormap(b, {'x':0}, var='heat', filename='ex2t.pdf', aspect='auto')
```

- `get_child()` method refers to one of the solids used in geometry definition.
- each solid has `temp`, `dens` and `heat` attributes to represent axial profiles of temperature, density and heat, respectively.
- `set_grid()` method sets amount and relative thickness of axial mesh layers; first list element corresponds lower layer.
- `colormap()` can generate colormaps of axial profiles.

Plots


Plane {'x': 0}, heat
200
100
0
−100
−200
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
4.0
3.6
3.2
2.8
2.4
2.0
1.6
1.2

Coupled calculations

```
from model import a
from mncp_data import MI  # McnpInterface
from scf_data import SI   # ScfInteface

# assign geometry with MCNP interface
MI.gm = a
# start MCNP
b = MI.run('R')

# assign ScfInterface model containing MCNP results
SI.gm = b
c = SI.run('R')
```

- given geometry `a` and code-specific data in `MI` and `SI` were already defined
- `run()` method generates input, starts code, waits until it completes, reads results and returns copy of `a` that contain in `heat` attributes MCNP results.

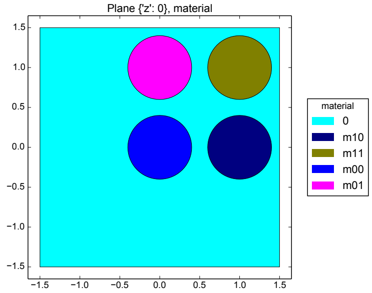
Square lattices

```
from pirs.solids import Box, Cylinder

b = Box(X=3, Y=3)

c = Cylinder(R=0.4)

b.grid.x = 1
b.grid.y = 1

for i in [0, 1]:
    for j in [0, 1]:
        cc = c.copy_tree()
        b.grid.insert((i, j, 0), cc)
        cc.material = 'm{}{}'.format(i,j)

from pirs.tools.plots import colormap
colormap(b, filename='ex3_1.pdf')

b.grid.center()
colormap(b, filename='ex3_2.pdf')
```

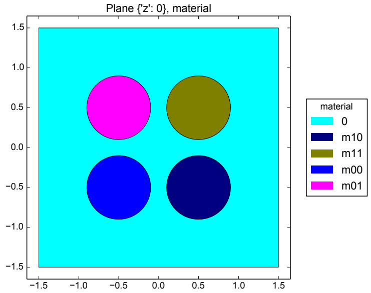
- `grid` attribute describes superimposed rectangular lattice, which can be used to position inserted solids.
- `grid.insert()` method is similar to `insert()` method, but inserted solid is placed in lattice element specified as 1-st agrument.
- `copy_tree()` method returns deep copy of a solid (i.e. all inserted solids are copied as well)
- `grid.center()` method positions lattice with respect to solid in a way that rectangle circumscribing all solids inserted into lattice elements, is centered.

Plots

Lattice not centered: (0,0,0)-th element is at origin



After `grid.center()` method



Features of high-level interfaces

- Complex geometries can be defined with operations of insertion and translation with respect to container.
- MCNP high-level interface can handle any geometry (theoretically).
  - `grid` is modelled with lattices in MCNP input file
  - solids are compared: equal solids represented using the same universe
  - If possible, macrobodies are used
  - non-trivial heat axial distributions considered as definition of heat deposition meshtally
  - temperature axial meshes define `tmp` cell options and are used to find xs data.
- SCF high-level interface is limited:
  - only square bundles of heated rods or unheated cylinder channels
  - all solids in model must have same height
  - coolant-centerd sub-channels are modelled in SCF, but
  - rod-centered temperatures and densities are returned back to model

Compound materials

```
from pirs.mcnp import Material

# water
h = Material('H')
o = Material('O')

w = 2*h + o
print w.report()

w.thermal = 'lwtr'
w.T = 450
w.sdict[8018] = 8016
print w.card(comments=False)

# Zircaloy
s = Material( ('Zr', 98.23, 1),
              ('Sn', 1.50,  1),
              ('Fe', 0.12,  1))
```

- Materials can be multiplied by scalar and added
- `thermal` attribute specifies part of thermal data name. Particular table is chosen to fit temperature.
- `sdict` dictionary specifies substitutions if cross-sections not available.
- `card` method returns multi-line string containg definition of material for MCNP input file. Can be used separately.
- Generally, `Material` class constructor takes a list of tuples of the form `(spec, amount, unit)`, where `spec` – string, integer or Material instance, `amount` – amount of ingredient and `unit` –flag specifying units.

Compound materials, output

```
Mixture H-O
       <    1001   0.9992>: 1.99977 mol
       <    1002   1.9968>: 0.00023 mol
       <    8016  15.8575>: 0.99757 mol
       <    8017  16.8531>: 0.00038 mol
       <    8018  17.8445>: 0.00205 mol
       total: 3.0 mol or 18.0152870569 g
Nuclide composition:
                 Nuclide        At.frac      Wgt.frac
       <    1001   0.9992>   6.66590e-01   1.11873e-01
       <    1002   1.9968>   7.66667e-05   2.57139e-05
       <    8016  15.8575>   3.32523e-01   8.85695e-01
       <    8017  16.8531>   1.26667e-04   3.58566e-04
       <    8018  17.8445>   6.83333e-04   2.04817e-03
m{0:<}
     1001.32c  9.72153e-01   1001.33c  1.02762e+00
     1002.32c  1.11810e-04   1002.33c  1.18190e-04
     8016.32c  4.84951e-01   8016.33c  5.12619e-01
     8017.32c  1.84730e-04   8017.33c  1.95270e-04
     8016.32c  9.96571e-04   8016.33c  1.05343e-03
mt{0:<} lwtr05.31t
```

- Material `w` contains nuclide O-18, but in MCNP material card it is substituted with O-16.
- Material card is formatting string, thus actual material numbers is simple to insert.
- Thremal data are chosen among cross-sections containing 'lwtr' in its name with closest temperature.

Implicit material definition

Problem: Given U and Pu isotopic vectors in % by weight, define MOX having 10% at. of fissile nuclides.

```
from pirs.mcnp import Material

u = Material((92235,  4, 2), (92238, 96, 2))
p = Material((94239, 90, 2), (94240, 10, 2))
o = Material(8016)

uox = u + 2*o
pox = p + 2*o

mox = Material((uox, 1), (pox, 1))
print mox.report()

def of(m):
    a1 = m.how_much(1, ZAID=[92235, 94239])
    a2 = m.how_much(1, Z=[92, 94])
    return a1 / a2 - 0.10

mox.tune(of, [uox, pox])
print mox.report()
print mox.how_much(1, ZAID=[92235, 94239])
print mox.how_much(1, Z=[92, 94])
```

- U and Pu elements, `u` and `p`, are defined using grams.
- Initially, `mox` is defined using equal amounts (moles) of U and Pu oxide.
- Objective function `of()` takes a material instance as argument and returns deviation of ratio of fissile nuclides to all heavy metal nucled from 5%
- Method `tune()` changes amount of specified ingredients until objective function returns (almost) zero.

Implicit material definition, output

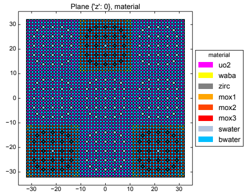
```
Mixture O-U-Pu
        <O-U        54.0662>: 1.0 mol
        <O-Pu       54.1142>: 1.0 mol
        total: 2.0 mol or 109.117752148 g
Nuclide composition:
                    Nuclide         At.frac       Wgt.frac
        <    8016  15.8575>    8.26713e-01   2.42366e-01
        <   92235 233.0248>    3.51571e-03   1.51460e-02
        <   92238 236.0058>    8.33112e-02   3.63503e-01
        <   94239 236.9986>    7.78462e-02   3.41087e-01
        <   94240 237.9916>    8.61349e-03   3.78985e-02
Mixture O-U-Pu
        <O-U        54.0662>: 1.86102294922 mol
        <O-Pu       54.1142>: 0.138977050781 mol
        total: 2.0 mol or 109.076019162 g
Nuclide composition:
                    Nuclide         At.frac       Wgt.frac
        <    8016  15.8575>    8.26397e-01   2.42366e-01
        <   92235 233.0248>    6.54282e-03   2.81977e-02
        <   92238 236.0058>    1.55044e-01   6.76746e-01
        <   94239 236.9986>    1.08188e-02   4.74214e-02
        <   94240 237.9916>    1.19708e-03   5.26904e-03
0.0347233122458 mol
0.347205722978 mol
```

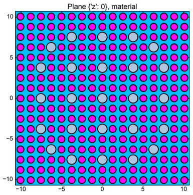
After `tune()` method, amount of fissile nuclides is 0.0347 mol and amount of all heavy metal nuclides is 0.347 mol in definition of `mox` material.

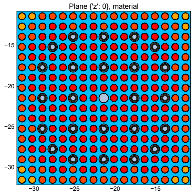
Outline

3x3 minicore

- Based on NEA PWR transient benchmark
- 6 UOX and 3 MOX 17x17 assemblies, 20 axial layers
- UOX has 2 types of rods: fuel pins and water channels
- MOX has 5 types of rods: 3 fuel pins, water channels and WABA
- Can be represented as single bundle with 51x51 rods
- To show performance of codes, provide benchmark for comparison







Calculations

- IC2 cluster, 1 node with 16 cores
- Standard meshtally to compute heat deposition

Iteration	kcode				wall time	Num of cells
0	500000	1.0	30	100	0:05:27	3544
1	809016	1.0	30	100	1:38:43	107634
2	1096763	1.0	30	100	1:52:51	107634
3	1374895	1.0	30	100	2:06:26	...
4	1647439	1.0	30	100	2:19:16	
5	1916300	1.0	30	100	2:29:45	...
20	5549120	1.0	30	100	5:05:40	...
21	5804749	1.0	30	100	5:15:52	
22	6060130	1.0	30	100	5:27:00	
23	6315284	1.0	30	100	5:38:33	
24	6570230	1.0	30	100	5:49:10	
25	6824985	1.0	30	100	6:00:09	...
26	7079562	1.0	30	100	6:12:28	107634

- MCNP initialization time 15 – 50 min