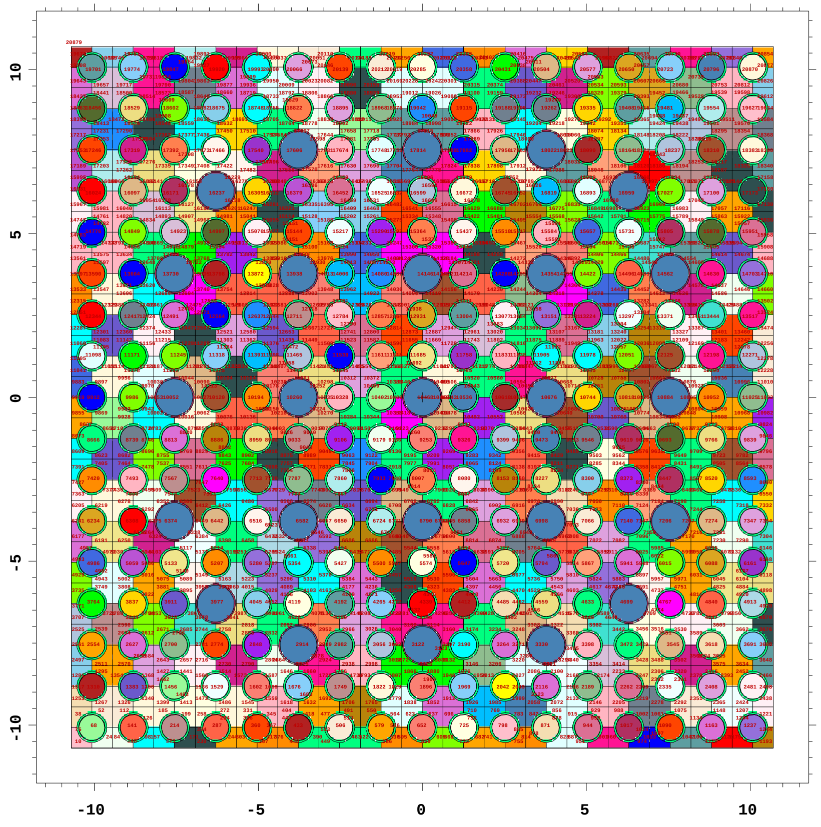
Geometry: horizontal cross-section

```
 09/10/13 17:32:56
c title

probid =  09/10/13 17:32:29
basis:   XY
( 1.000000, 0.000000, 0.000000)
( 0.000000, 1.000000, 0.000000)
origin:
(      0.00,      0.00,      0.00)
extent = (    11.78,     11.78)
cell labels are
 cell names
```

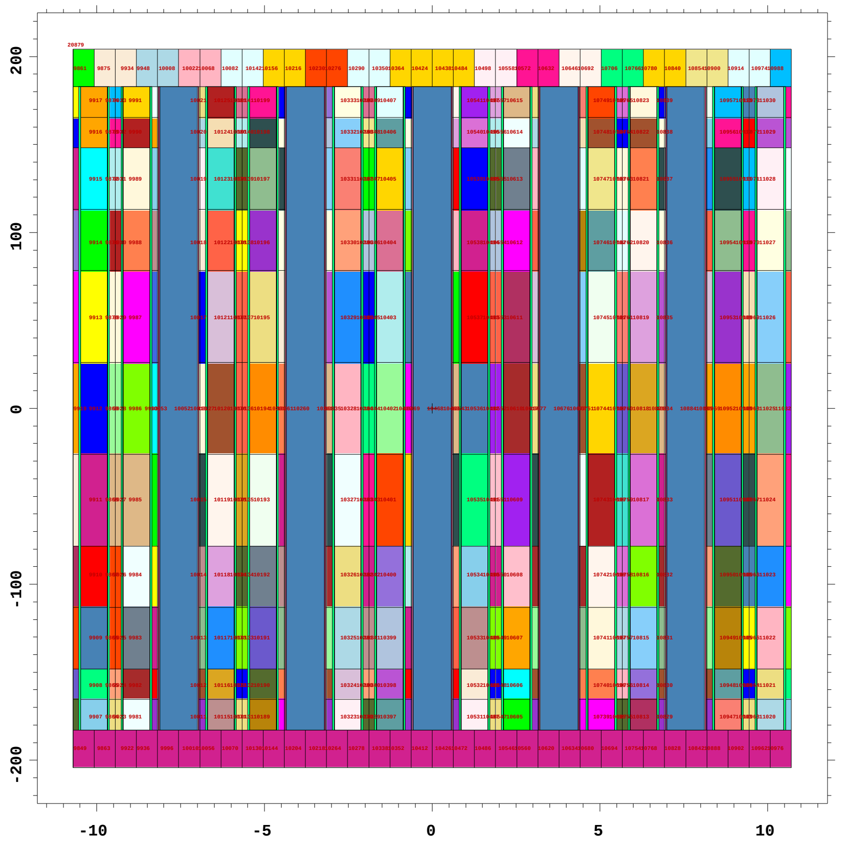


- lattice card
- coolant-centered channels
- 2 fuel types
- water channels

Geometry: vertical cross-section

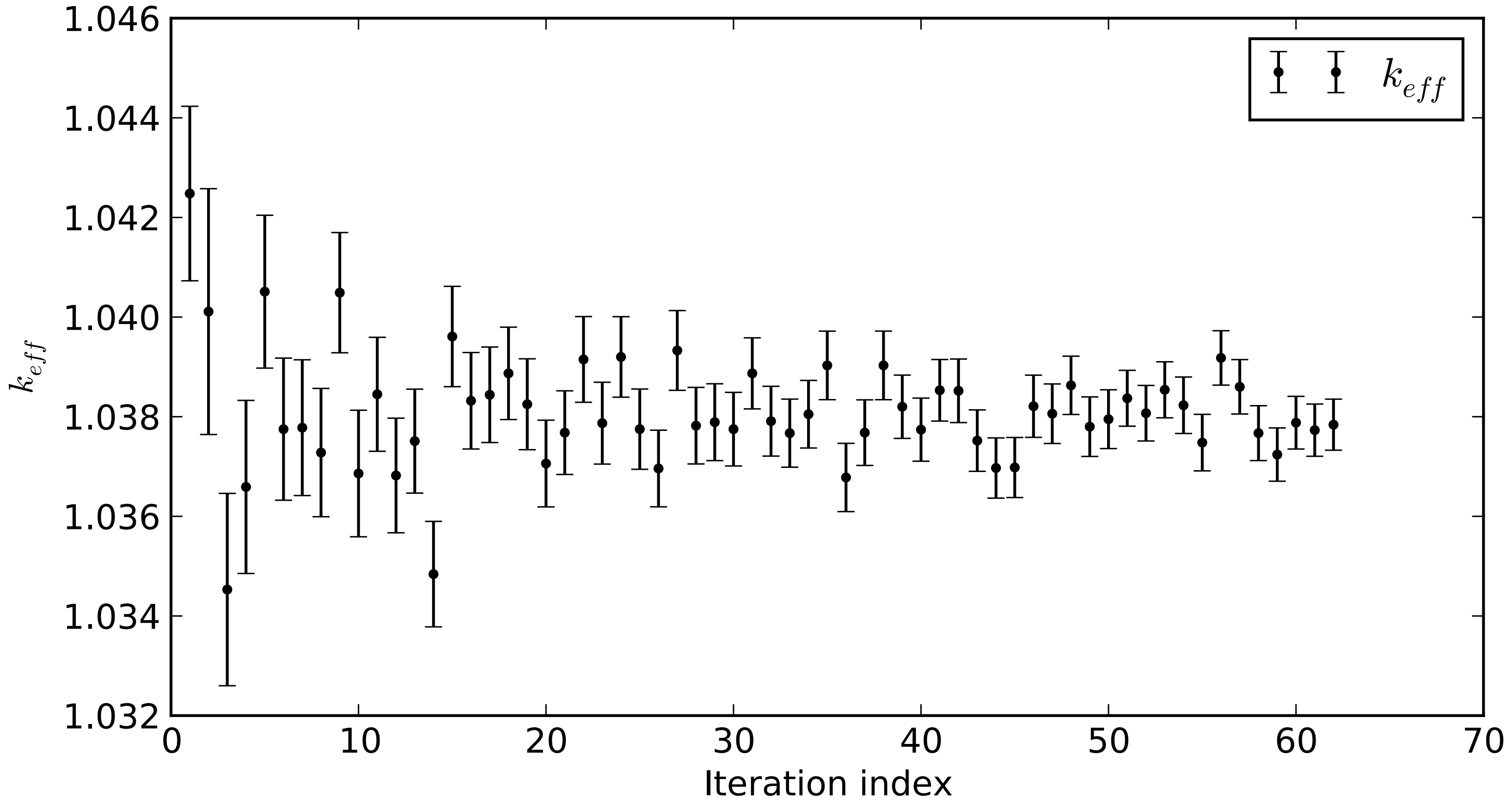
```
 09/10/13 17:32:58
c title


probid =  09/10/13 17:32:29
basis:   XZ
( 1.000000, 0.000000, 0.000000)
( 0.000000, 0.000000, 1.000000)
origin:
(     0.00,     0.00,     0.00)
extent = (    11.78,    224.73)
cell labels are
 cell names
```

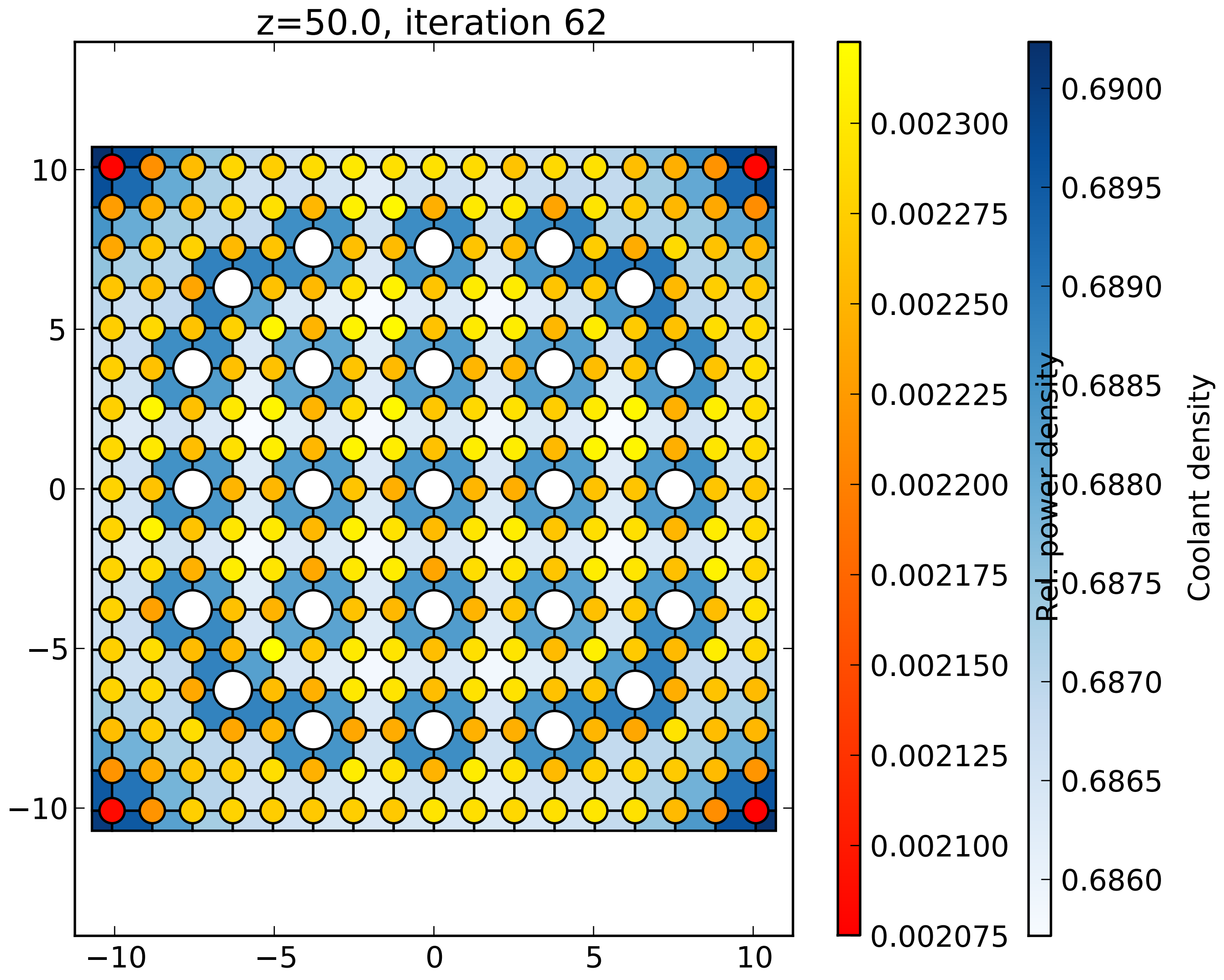


- non-uniform axial mesh
- changing T and density of coolant water
- constant water properties in water channels

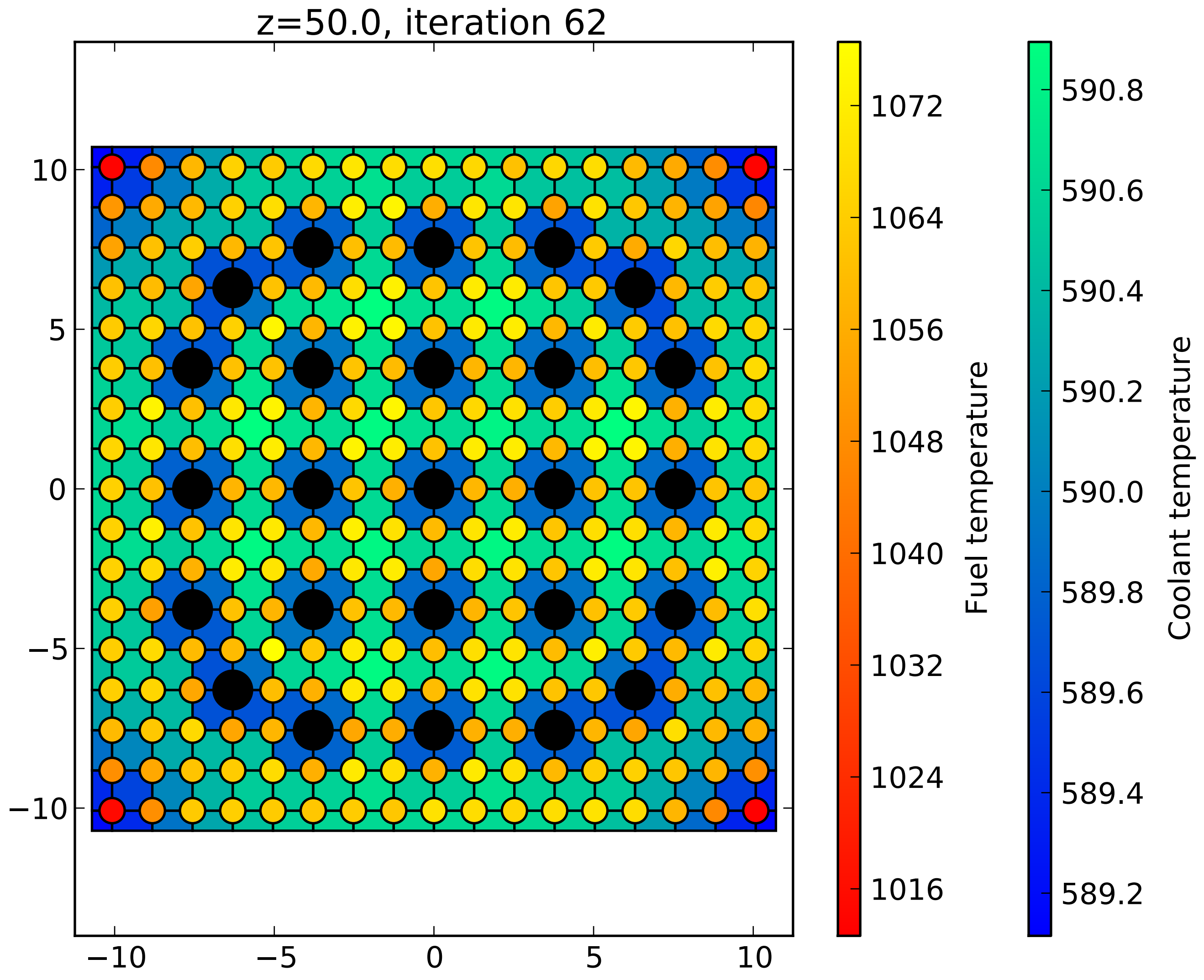
Keff behaviour



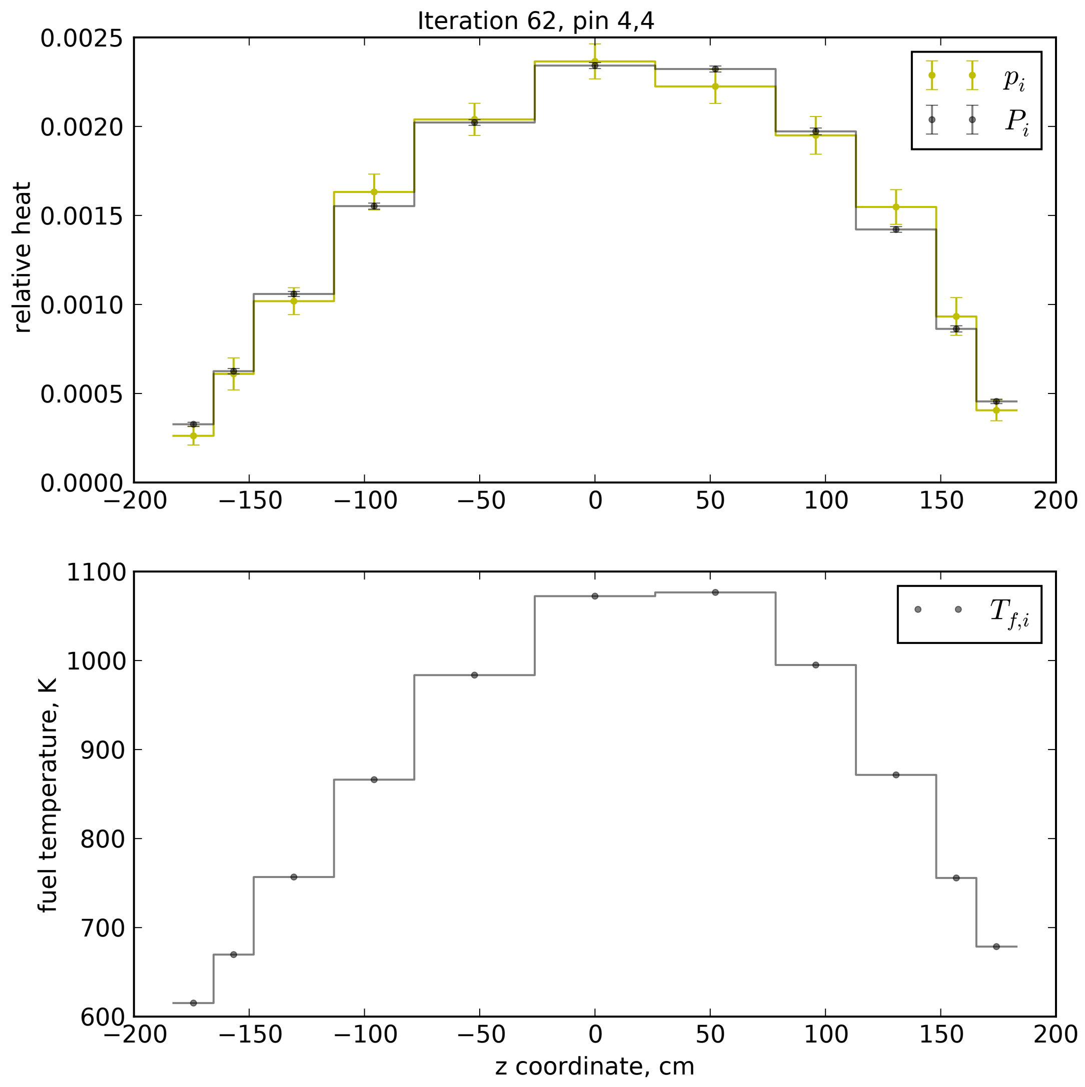
Heat and water density



Fuel and water temperature



Axial distribution in pin (4,4)



Python scripts

`driver.py`
Main script that controls calculation flow and describes relaxation scheme

`rod_models.py`
Dimensions, description of UOX and IFBA fuel pins

`pin_model.py`
Geometry of one-pin model

`pin_mcnp.py`
MCNP-specific data for one-pin model

`pin_scf.py`
SCF-specific data for one-pin model

`assembly_model.py`
Geometry of assembly model

`assembly_map.py`
Pseudo-graphics definition of the assembly map

`assembly_mcnp.py`
MCNP-specific data for assembly model

`assembly_scf.py`
SCF-specific data for assembly model

driver.py

```
# import section
from sys import argv
import gc
from datetime import datetime

from hpmc import dump, load
from some_funcs import relaxed, have_zeroes

# 1

# process command line argument
if argv[1][0] in 'pqr':
    # import model, SCF- anf MCNP-specific data for a pin model:
    from pin_model import model
    from pin_scf import SI
    from pin_mcnp import MI
    prefix = argv[1][0] + '_'
elif argv[1][0] in 'abcde':
    # import description of the assembly
    from assembly_model import model
    from assembly_scf import SI
    from assembly_mcnp import MI
    prefix = argv[1][0] + '_'
new_start = True

# read dump from previous run
if '.dump' in argv[1]:
    # argv[1] is the dump file. Read it and continue calculations.
    dmp = load(argv[1])
    Ic = dmp['Ic']
    s1 = dmp['s1']
    Ss = dmp['Ss']
    MI.kcode = dmp['kcode']
    sres = dmp['scf_result']
    Rm = dmp['relaxed']
    MI.wp = dmp['mcnp_wp']
    SI.wp = dmp['scf_wp']
    Keff = dmp['Keff']
    Kerr = dmp['Kerr']
    Emax = dmp['Emax']
    if 'Emax' in argv:
        Emax = float(argv[argv.index('Emax') + 1])
    new_start = False

# 2

# define initial parameters
if new_start:
    SI.wp.prefix = prefix + 'scf_'
    MI.wp.prefix = prefix + 'mcnp_'
    # Iteration initialization
    s1 = MI.kcode.Nct * MI.kcode.Nh # total number of neutron histories.
    Ss = 0 # cumulative number of neutron histories
    Ic = 0 # iteration counter

    # plot initial model with MCNP
    MI.gm = model
    MI.run('P')

    # SCF run to get initial temperature distribution
    SI.gm = model
    sres = SI.run('R')

    # Plot SCF results with MCNP
    MI.gm = sres
    MI.run('P')

    # model containing relaxed power. Initially, power iz 0.
    Rm = relaxed(0., 0., sres, sres)

    # Criteria for Keff difference and std.dev:
    Emax =  50.e-5  # 1e-5 is one pcm
    # 'initial' values for Keff and std.dev. Values chosen to ensure that
    # first two interations take place
    Keff = [-1, -2]
    Kerr = [1., 1.]

# 3

# iterations
while max(Kerr[-2:]) > Emax or abs(Keff[-1] - Keff[-2]) > Emax:
        Ic += 1

        print
        print
        print '----- Iteration {} --- {}'.format(Ic, datetime.now().strftime('%H:%M:%S'))

        # 4

        # ----------------------------------------------------------------------
        # MC-RUN
        # ----------------------------------------------------------------------
        # Compute new number of cycles
        s = 0.5*(s1 + (s1**2 + 4.*s1*Ss)**0.5)
        MI.kcode.Nct = int(s / MI.kcode.Nh)  # new number of cycles

        # Specify model with SCF results to MCNP interface:
        MI.gm = sres
        # remove explicit source specs. The previous srctp will be used.
        if MI.wp.srctp.defined and 'ksrc' in MI.adc[-1]:
            MI.adc.pop()
        # run MCNP and save results to mres model
        mres = MI.run('R')
        print '    MCNP input file generated in {} seconds'.format(MI.process_model_time)
        print '    MCNP run took {} seconds'.format(MI.wp.run_time)

        # continue until all tallies are non-zero. Otherwise SCF might fail.
        while have_zeroes(mres):
            MI.kcode.Nct += 100
            s += 100 * MI.kcode.Nh
```

```
            mres = MI.run('C', ccard=str(MI.kcode))

        # get Keff of last MCNP run and append it to Keff list:
        if MI.wp.mctal.defined:
            keff, err = MI.keff()
        else:
            keff = 1. + Ic * 0.01
            err = 0.01 / (Ic + 10)
        Keff.append(keff)
        Kerr.append(err)

        # 5

        # ------------------------------------------------------------------------
        # compute relaxed power
        # ------------------------------------------------------------------------
        Ss += s
        a = float(s)/float(Ss) # convert to float, otherwise s/Ss is allways zero.
        Rm = relaxed(a, 1.-a, mres, Rm)

        # 6

        # ------------------------------------------------------------------------
        # TH-RUN
        # ------------------------------------------------------------------------
        SI.gm = Rm
        sres = SI.run('R')

        # 7

        # ------------------------------------------------------------------------
        # dump iteration results
        # ------------------------------------------------------------------------
        dump(prefix + 'iteration_{:03d}.dump'.format(Ic),
                mcnp_result = mres,
                scf_result = sres,
                relaxed = Rm,
                mcnp_wp = MI.wp,
                scf_wp = SI.wp,
                kcode = MI.kcode,
                Ss = Ss,
                s1 = s1,
```

```
                Keff = Keff,
                Kerr = Kerr,
                Emax = Emax,
                Ic = Ic)
```

```
        # 8

        # summary info
        print 'Keff:'
        print 'Current iteration: {:9.6f}  +-  {:8.6f}'.format(Keff[-1], Kerr[-1])
        print 'prev.   iteravion  {:9.6f}  +-  {:8.6f}'.format(Keff[-2], Kerr[-2])
        print '             diff: {:9.6f} goal:{:8.6f}'.format(Keff[-1] - Keff[-2], Emax)
        print 'amount of collected garbage: ', gc.collect()
```

rod_models.py

```python
"""
Geometry of rods used in OECD NEA benchmark.
"""

# 1

# Dimensions form the benchmark
# www.oecd-nea.org/science/wprs/MOX-UOX-transients/benchmark_documents/
# specifications/mox_bench_spec.pdf

# Table 2, p.5:
ah = 365.76 # active height, cm
ap = 21.42  # assembly pitch, cm
pp = 1.26   # pin pitch, cm

# Table 6, p.8:
pin_r1 = 0.3951  # fuel pellets radius, cm
pin_r2 = 0.4010  # clad inner radius, cm
pin_r3 = 0.4583  # clad outer radius, cm

ifba_r1 = 0.3951
ifba_r2 = 0.3991
ifba_r3 = 0.4010
ifba_r4 = 0.4583

tube_r1 = 0.5624
tube_r2 = 0.6032

# 2

from hpmc import Cylinder

# 2a

# pin model
```

```
clad = Cylinder(R=pin_r3, Z=ah)
gap  = Cylinder(R=pin_r2, Z=ah)
fuel = Cylinder(R=pin_r1, Z=ah)
clad.insert('gap', gap)
gap.insert('fuel', fuel)

clad.material = 'zirc'
gap.material = 'oxygen'
fuel.material = 'uo2'

clad.dens.set_values(6.504)
clad.temp.set_values(600.)

gap.dens.set_values(0.001)
gap.temp.set_values(600.)

fuel.temp.set_values(1200)
fuel.dens.set_values(10.24)
fuel.heat.set_grid([1, 1, 2, 2, 3, 3, 3, 2, 2, 1, 1])
fuel.heat.set_values(0.5)

pin = clad.copy_tree()

# 3

# ifba model 1: in SCF model, gap is from fuel to clad
coat = Cylinder(R=ifba_r2, Z=ah, material='ifba')
coat.dens.set_values(1.69)
coat.temp.set_values(600.)

ifba = pin.copy_tree()
gap = ifba.get_child('gap')

gap.insert('coat', coat)
gap.shift_child('coat', 0)

# 3a
```

```python
# ifba model 2: in SCF model, gap is from ifba coating to clad
ifba2 = pin.copy_tree()
coat = ifba2.get_child(('gap', 'fuel'))
newf = coat.insert('pellets', coat.copy_node())
newf.heat.clear()
coat.R = ifba_r2
coat.material = 'ifba'
coat.dens.set_values(1.69)
coat.temp.set_values(600)

# ifba = ifba2

# 4

# guide tube model
tube = Cylinder(R=tube_r2, Z=ah, material='zirc')
tube.temp.set_values(580.)
tube.dens.set_values(6.504)

wach = Cylinder(R=tube_r1, Z=ah, material='water') # water channel
wach.temp.set_values(580.)
wach.dens.set_values(0.71187)

tube.insert('water channel', wach)
```

pin_model.py

```python
# -----------------------------------------------------------------------------------

from hpmc import Box
from rod_models import pp, ah, ap
from rod_models import ifba as pin

w = Box(X=pp, Y=pp, Z=ah + 2*ap)  # water box
w.material = 'water'

w.dens.set_values(1)
w.temp.set_values(580.)

w.insert('pin', pin)


# needed in SCF interface:
fuel_key = ('pin', 'gap', 'fuel')
rod_key = pin.get_key()

# unify names
model = w
```

pin_mcnp.py

```python
from hpmc import McnpInterface
import mcnp

# 1

# material compositions
water = mcnp.Material((1001, 2), ('O'))
water = water*1. + mcnp.Material('B')*1.5e-3 # 1500 ppm of boron
water.thermal = 'lw'

# 2

zirc = mcnp.Material(('Zr', 98.23),
                     ('Sn', 1.50),
                     ('Fe', 0.12),
                     ('Cr', 0.10),
                     ('Ni', 0.05)) # zircaloy-2, Table 5, p.7

# 3

u = mcnp.Material((92235,  4.2, 2),
                  (92238, 95.8, 2)) # mass fractions

o = mcnp.Material('O')
uo2 = u + 2*o

ifba = mcnp.Material(('Zr', 1), ('B', 2))

# 4

# substitution rules for isotopes not in xsdir:
water.sdict[8018] = 8016
uo2.sdict[8018] = 8016
o.sdict[8018] = 8016
```

```python
# 5

# MCNP interface
MI = McnpInterface()

# correspondence of the material names and material compositions:
MI.materials['water'] = water
MI.materials['zirc'] = zirc
MI.materials['uo2'] = uo2
MI.materials['oxygen'] = o
MI.materials['ifba'] = ifba

# reflective bc on the lateral facets:
MI.bc['radial'] = '*'

# kcode parameters:
MI.kcode.Nh = 1000     # histories per cycle
MI.kcode.Ncs = 10      # inactive cycles
MI.kcode.Nct =  50     # total cycles
MI.kcode.active = True
# additional data card to specify kcode source for the first run only:
MI.adc.append('ksrc  0 0 -150  0 0 0  0 0 150')

# 6

# Save MCNP results using uncertainties package:
MI.TallyCollection.use_uncertainties = True
```

pin_scf.py

```python
from hpmc import ScfInterface
from pin_model import rod_key

# 1

# create interface
SI = ScfInterface()

# model elements to be considered as
# coolant container and rods:
SI.keys['rods'].append(rod_key)
SI.keys['coolant'] = ''

# 2

# TH specifications, Table 2, p.5
thp = 3565e6    # Core thermal power, W
Na = 193        # number of assemblies
Np = 264        # number of pins
Tin = 560.      # inlet temperature, K
cflow = 15849.4 * 1e3 # core flow, g/sec
Pin = 15.5e6    # inlet pressure

SI.inlet_temperature = Tin
SI.total_power = thp / Na / Np       # average pin power
SI.inlet_flow_rate = cflow / Na / Np # assuming no bypass
SI.exit_pressure = Pin               # SCF accepts exit pressure
# thi.pressure_drop = 0.02e6

# 3

# SCF calculation control parameter
SI.calcon.get_variable('max_of_axial_flow_iterations').value = 1000

# 4
# Material names correspondence between general model and SCF
SI.materials['uo2'] = 'benpwr' # TH correlations for fuel from OECD benchmark
# SI.materials['uo2'] = 'uo2'
SI.materials['zirc'] = 'zircaloy'

SI.materials['ifba'] = SI.materials['uo2']
```

assembly_model.py

```python
from hpmc import Box
from rod_models import pin, ifba, tube
from rod_models import ap, ah, pp
from assembly_map import map_dict

# 1

# Number of rod rows and columns:
Nx = 17
Ny = Nx

# 2

model = Box(Z=ah + 2*ap)
model.X = Nx*pp - 0.000001
model.Y = Ny*pp - 0.000001

model.material = 'water'
model.temp.set_values(580.)
model.dens.set_values(1.)

model.grid.x = pp
model.grid.y = pp
model.grid.z = model.Z

# 3

# prepare rods
rods = []
for j in range(Ny):
    for i in range(Nx):
        rod_type = map_dict[(i,j)]
        if rod_type == 'u':
            key = 'pin {},{}'
            rod = pin.copy_tree()
        elif rod_type == 'i':
            key = 'ifba {},{}'
            rod = ifba.copy_tree()
        elif rod_type in 'gc':
            key = 'tube {},{}'
            rod = tube.copy_tree()
        key = key.format(i,j)
        rods.append((key, rod))

# insert rods to the model:
rod_keys = []
for j in range(Ny):
    for i in range(Nx):
        key, rod = rods.pop(0)
        model.insert(key, rod, (i,j,0))
        rod_keys.append(rod.get_key())

# put lattice to center:
model.grid.center()
```

assembly_map.py

```python
"""
Pseudo-graphics input of the assembly map.
"""


# g -- guide tube
# u -- uox pins
# i -- ifba pins
# c -- control rods or guide tubes

# map with guide tubes and IFBA pins
map_string1 = """
    i  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  i
    u  u  u  u  u  i  u  u  i  u  u  i  u  u  u  u  u
    u  u  u  i  i  c  i  i  c  i  i  c  i  i  u  u  u
    u  u  i  c  i  i  u  u  i  u  u  i  i  c  i  u  u
    u  u  i  i  u  i  u  u  i  u  u  i  u  i  i  u  u
    u  i  c  i  i  c  i  i  c  i  i  c  i  i  c  i  u
    u  u  i  u  u  i  u  u  i  u  u  i  u  u  i  u  u
    u  u  i  u  u  i  u  u  i  u  u  i  u  u  i  u  u
    u  i  c  i  i  c  i  i  g  i  i  c  i  i  c  i  u
    u  u  i  u  u  i  u  u  i  u  u  i  u  u  i  u  u
    u  u  i  u  u  i  u  u  i  u  u  i  u  u  i  u  u
    u  i  c  i  i  c  i  i  c  i  i  c  i  i  c  i  u
    u  u  i  i  u  i  u  u  i  u  u  i  u  i  i  u  u
    u  u  i  c  i  i  u  u  i  u  u  i  i  c  i  u  u
    u  u  u  i  i  c  i  i  c  i  i  c  i  i  u  u  u
    u  u  u  u  u  i  u  u  i  u  u  i  u  u  u  u  u
    i  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  i
    """

# map with guide tubes, without IFBA
map_string2 = """
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  c  u  u  c  u  u  c  u  u  u  u  u
    u  u  u  c  u  u  u  u  u  u  u  u  u  c  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  c  u  u  c  u  u  c  u  u  c  u  u  c  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  c  u  u  c  u  u  g  u  u  c  u  u  c  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  c  u  u  c  u  u  c  u  u  c  u  u  c  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  c  u  u  u  u  u  u  u  u  u  c  u  u  u
    u  u  u  u  u  c  u  u  c  u  u  c  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    """

# trivial map, only usual pins
map_string3 = """
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u  u
    """
```

```python
# what map is modelled.The map_string2 was
# used to get results for D1.4, part II.
map_string = map_string1

map_dict = {}
j = 0
for l in reversed(map_string.splitlines()):
    row = l.split()
    if len(row) > 0:
        i = 0
        for e in row:
            map_dict[(i,j)] = e
            i += 1
        j += 1
```

assembly_mcnp.py

```python
from mcnp import Material
from pin_mcnp import MI
from assembly_model import model

# ifba material:
ifba = Material(('Zr', 1), ('B', 2))
MI.materials['ifba'] = ifba

# Optionally, one can provide ksrc point
# for each fuel element:
ksrc = 'ksrc'
for e in model.values():
    if 'fuel' in e.get_key():
        x, y, z = e.abspos().car
        ksrc += '   {} {} {}'.format(x, y, z)
MI.adc[-1] = ksrc

# 1

if __name__ == '__main__':
    MI.gm = model
    MI.run('P')
```

assembly_scf.py

```python
from pin_scf import SI, thp, Na, Np, cflow
from assembly_model import rod_keys, Nx, Ny

# 1

# change keys, specifying rod elements
# in the general model:
SI.keys['rods'] = rod_keys

# 2

# adjust total power:
SI.total_power = thp / Na # / Np * Nx*Ny

# adjust flow rate:
SI.inlet_flow_rate = cflow / Na #  / Np * Nx*Ny
```