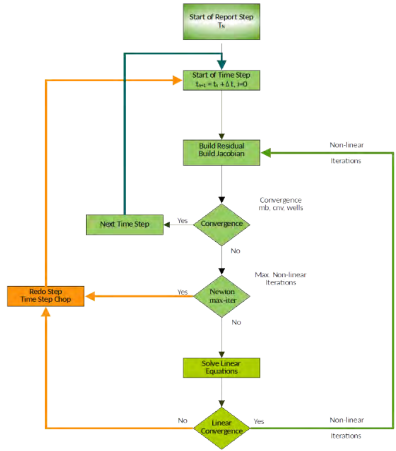
The PYACTION keyword

Lisa Julia Nebel (OPM-OP A/S), Håkon Hægland (NORCE)

April 9th, 2024, OPM-Summit, Oslo

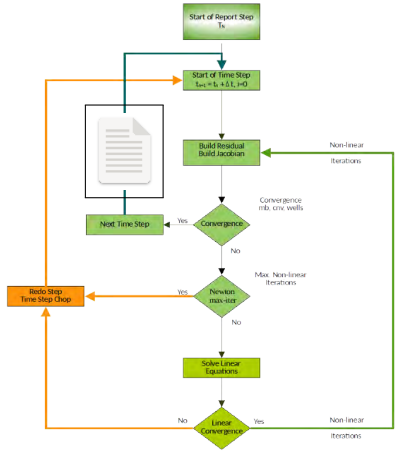
What are the benefits of the PYACTION keyword?

execute a Python script embedded in a running flow simulation



What are the benefits of the PYACTION keyword?

execute a Python script embedded in a running flow simulation



What are the benefits of the PYACTION keyword?

- change properties (well properties, transmissibilites, . . .) depending on conditions
- generate output for post-processing

What are the benefits of the PYACTION keyword?

- change properties (well properties, transmissibilites, ...) depending on conditions
  $\rightarrow$ `ACTIONX` and `UDQ` keywords
- generate output for post-processing

What are the benefits of the PYACTION keyword?

- change properties (well properties, transmissibilites, ...) depending on conditions
  $\rightarrow$ `ACTIONX` and `UDQ` keywords
- generate output for post-processing
  $\rightarrow$ `SUMMARY` keyword, ResInsight, paraview, ...

What are the benefits of the PYACTION keyword?

- change properties (well properties, transmissibilites, ...) depending on conditions
  $\rightarrow$ `ACTIONX` and `UDQ` keywords
- generate output for post-processing
  $\rightarrow$ `SUMMARY` keyword, ResInsight, paraview, ...

$\Rightarrow$ **More flexible and easier with embedded Python code ☺**

What are the benefits of the PYACTION keyword?

- How do I use the `PYACTION` keyword?
- What can I do in the embedded Python code? How does this relate to `ACTIONX`?
- How does this relate to the opm Python bindings?

What are the benefits of the PYACTION keyword?

- How do I use the `PYACTION` keyword?
- What can I do in the embedded Python code? How does this relate to `ACTIONX`?
- How does this relate to the opm Python bindings?

How do I use the PYACTION keyword?

1. Build flow with `OPM_EMBBEDDED_PYTHON=ON`, e.g., in `opm-common/CMakeLists.txt`

CMakeLists.txt

```
cmake_minimum_required (VERSION 3.10)
project(opm-common C CXX)

option(SIBLING_SEARCH "Search for other modules in sibling directories?" ON)
list(APPEND CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake/Modules)
set(OPM_MACROS_ROOT ${PROJECT_SOURCE_DIR})

option(ENABLE_ECL_INPUT "Enable eclipse input support?" ON)
option(ENABLE_ECL_OUTPUT "Enable eclipse output support?" ON)
option(ENABLE_MOCKSIM "Build the mock simulator for io testing" ON)
option(OPM_ENABLE_PYTHON "Enable python bindings?" OFF)
option(OPM_INSTALL_PYTHON "Install python bindings?" ON)
option(OPM_ENABLE_EMBEDDED_PYTHON "Enable embedded python?" ON)
```

How do I use the PYACTION keyword?

1. Build flow with `OPM_EMBBEDDED_PYTHON=ON`

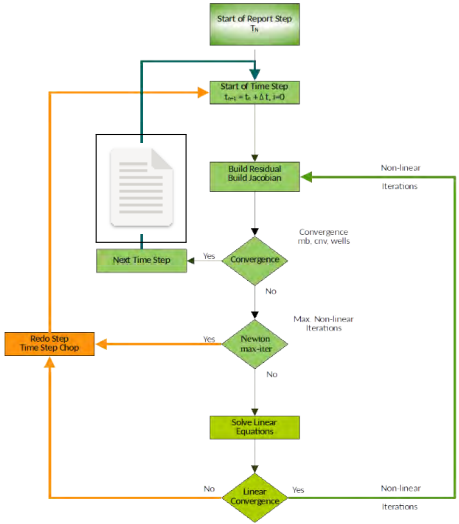
2. Add `PYACTION` keyword to `DATA`-file in `SCHEDULE`-section

```
...
SCHEDULE
...
PYACTION
<PYACTION_NAME> <SINGLE/UNLIMITED> /
<pythonscript> /
...
```

How do I use the PYACTION keyword?

1. Build flow with `OPM_EMBBEDDED_PYTHON=ON`
2. Add `PYACTION` keyword to `DATA`-file in `SCHEDULE`-section
3. Create / Provide the python file
4. Run flow

How do I use the PYACTION keyword?



What are the benefits of the PYACTION keyword?

- How do I use the `PYACTION` keyword?
- What can I do in the embedded Python code? How does this relate to `ACTIONX`?
- How does this relate to the opm Python bindings?

What can I do in the embedded Python code?

```python
# Python module opm_embedded
import opm_embedded

# The current Opm::EclipseState
ecl_state = opm_embedded.current_ecl_state

# The current Opm::Schedule
schedule = opm_embedded.current_schedule

# The current Opm::SummaryState
summary_state = opm_embedded.current_summary_state

# The current report step
report_step = opm_embedded.current_report_step
```

What can I do in the embedded Python code?

- insert keywords

```
kw = """
GCONPROD
FIELD     GRAT   1*      1*     250E3    1*      RATE    1*     1*
  1*      1*       /
/"""
schedule.insert_keywords(kw)
```

- tested with `GCONPROD`, `WCONPROD`, `MULTXYZ`, `WEFAC`, `WELOPEN`, `NEXTSTEP`
- open/shut/stop a well:

```
schedule.open_well("P1")
schedule.shut_well("P2")
schedule.stop_well("P3")
```

What can I do in the embedded Python code?

- insert keywords

```
kw = """
GCONPROD
FIELD     GRAT   1*      1*     250E3    1*      RATE    1*     1*
  1*      1*       /
/"""
schedule.insert_keywords(kw, report_step)
```

- tested with `GCONPROD`, `WCONPROD`, `MULTXYZ`, `WEFAC`, `WELOPEN`, `NEXTSTEP`
- open/shut/stop a well:

```
schedule.open_well("P1", report_step)
schedule.shut_well("P2", report_step)
schedule.stop_well("P3", report_step)
```

What can I do in the embedded Python code?

- access / set variables of the SummaryState

```
FGPR = summary_state["FGPR"]

WGOR_OP01 = summary_state.well_var("OP01", "WGOR")
WGOR_OP02 = summary_state.well_var("OP02", "WGOR")
GOPR_FIELD = summary_state.group_var("FIELD", "GOPR")

if (not "FU_GASFL" in summary_state):
  summary_state["FU_GASFL"] = 0
```

- ... and normal Python code

How does this relate to ACTIONX and UDQ?

- insert_keyword does the same as an `ACTIONX`
- examples in opm-test: $\approx$ 50% shorter, easier Python code replaces `ACTIONX` and `UDQ`
- some embedded Python code is probably very hard to realize with `ACTIONX` and `UDQ`

How does this relate to ACTIONX and UDQ?

- insert_keyword does the same as an `ACTIONX`
- examples in opm-test: $\approx$ 50% shorter, easier Python code replaces `ACTIONX` and `UDQ`
- some embedded Python code is probably very hard to realize with `ACTIONX` and `UDQ`

$\Rightarrow$ **Embedded Python code is more flexible and easier ☺**

How does this relate to ACTIONX and UDQ?

- former signature is still available

```
def run(ecl_state, schedule, report_step, summary_state, actionx_callback):
  ...
```

- run function (with actionx_callback) is not necessary anymore

What are the benefits of the PYACTION keyword?

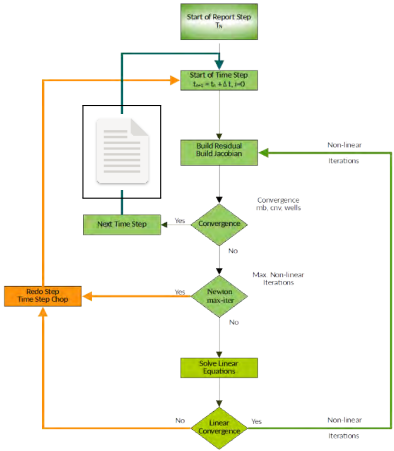
- How do I use the `PYACTION` keyword?
- What can I do in the embedded Python code? How does this relate to `ACTIONX`?
- How does this relate to the opm Python bindings?

How does this relate to the Python bindings?

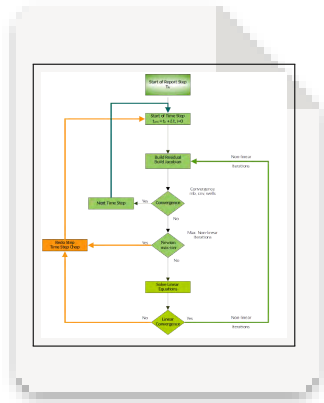
```python
import os
import sys
from opm.simulators import BlackOilSimulator
from opm.io.parser import Parser
from opm.io.ecl_state import EclipseState
from opm.io.schedule import Schedule
from opm.io.summary import SummaryConfig

deck  = Parser().parse('SAMPLE_DECK.DATA')
state = EclipseState(deck)
schedule = Schedule( deck, state )
summary_config = SummaryConfig(deck, state, schedule)

sim = BlackOilSimulator(deck, state, schedule, summary_config)
sim.step_init()
sim.step()
poro = sim.get_porosity()
poro = poro *.95
sim.set_porosity(poro)
sim.step()
sim.step()
sim.step()
sim.step_cleanup()
```

How does this relate to the Python bindings?

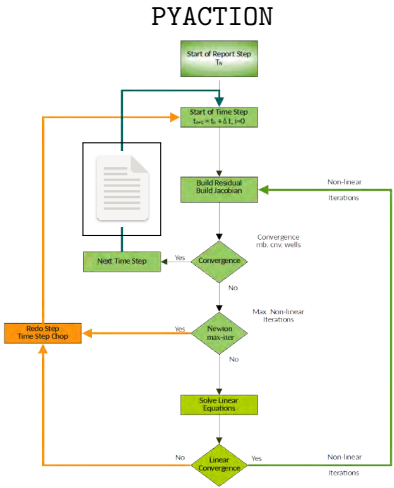
PYACTION



Python bindings

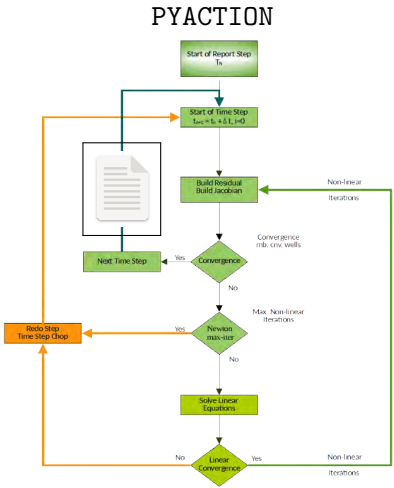


Summary



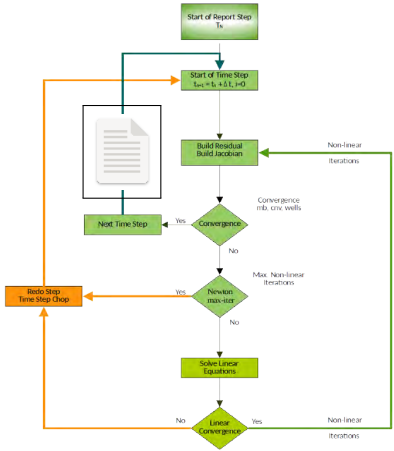
- **execute a Python script embedded** in a running simulation
- **opm_embedded module**: access to current EclipseState, SummaryState, Schedule and Report Step

Outlook



- online Sphinx documentation?
- parallel use of OPM?
- some code is shared between the Python bindings and embedded Python $\rightarrow$ remove or specifically test that
- enable the use of all `ACTIONX` keywords for insert_keywords

Thank you ☺


Start of Report Step
T_n
Start of Time Step
$t_{n+1} = t_n + \Delta t_i$, i=0
Build Residual
Build Jacobian
Non-linear
Iterations
Convergence
mb, cnv, wells
Yes
Convergence
Next Time Step
No
Max. Non-linear
Iterations
Redo Step
Time Step Chop
Yes
Newton
max-iter
No
Solve Linear
Equations
No
Linear
Convergence
Yes
Non-linear
Iterations