PyPKE simulation of rod drop method

정지헌
Seoul National University

Contents

Introduction

Reactor theory, is built on the foundation of “point kinetics equation(PKE)”.

Moreover, solving PKE is the key to evaluate or predict the neutron density and precursor density of the reactor.

If we can reduce time and simplify the complexity of the integral-differential equation while maintaining accuracy to a certain level, a more efficient estimation could be made of the reactor behavior.

Using Taylor expansion, PKE can be reduced to a simplified format and discrete steps of small values can lead to reasonable solutions.

Base Idea of PyPKE algorithm

Point kinetics equation

$$\frac{dN(t)}{dt} = \frac{\rho(t) - \beta}{\Lambda} N(t) + \sum_{i=1}^{6} \lambda_i C_i(t)$$

$$\frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda} N(t) - \lambda_i C_i(t)$$

Taylor expansion

$$N(t+h) = N(t) + h\frac{dN}{dt} + \frac{1}{2!}\frac{d^2N}{dt^2} + \cdots$$

$$C_i(t+h) = C_i(t) + h\frac{dC_i}{dt} + \frac{1}{2!}\frac{d^2C_i}{dt^2} + \cdots$$

We take only the terms to first-order

The iteration time step *h*

$$N(t+h) = N(t) + h\left(\frac{\rho(t) - \beta}{\Lambda} N(t) + \sum_{i=1}^{6} \lambda_i C_i(t)\right)$$

$$C_i(t+h) = C_i(t) + h\left(\frac{\beta_i}{\Lambda} N(t) - \lambda_i C_i(t)\right)$$

Goal

1. To make an efficient imitation of the reference paper
2. To make an open-source project where anybody can commit to enhance performance(=accuracy compared to analytical solution)

Components of PyPKE.py

more on GitHub

Basic information about the PyPKE

```python
class PKE(object):
    def __init__(self, reactivity=0, mode_number=0):
        self.beta = [0.000331, 0.002198, 0.001963, 0.003972, 0.001156, 0.000465]
        self.lda = [0.0124, 0.0305, 0.1110, 0.3010, 1.1300, 3.0000]
        self.time_step = 0.001
        self.rho = reactivity
        self.life = 1E-4
        self.mode = mode_number
        self.gen = self.life * (1 - self.rho)
        self.neutron_density = 1
        self.precursor_density = []
        for i in range(6):
            tmp = self.beta[i] / self.lda[i] / self.gen
            self.precursor_density.append(tmp)
```

Point kinetic equation factors

```python
def neutron(self, time):
    time = time // self.time_step * self.time_step
    if time <= 0:
        return self.neutron_density
    else:
        res = self.neutron_density * (1 + self.time_step * (self.rho - sum(self.beta)) / self.gen)
        for i in range(6):
            res += self.time_step * self.lda[i] * self.precursor_density[i]
        self.update_neutron(res)
        return res

def precursor(self, index, time):
    time = time // self.time_step * self.time_step
    if time <= 0:
        return self.precursor_density[index]
    else:
        res = self.precursor_density[index] * (1 - self.time_step * self.lda[index]) \
            + self.time_step * self.beta[index] / self.gen * self.neutron_density
        self.update_precursor(res, index)
        return res
```

Components of PyPKE.py

```python
def update_neutron(self, neutron):
    self.neutron_density = neutron

def update_precursor(self, precursor, i):
    self.precursor_density[i] = precursor

def run(self):
    rho_string = "{:.4f}".format(self.rho)
    file_name = 'PyAGN_rho=' + rho_string + '.dat'
    f = open(file_path + file_name, 'wt')
    # data until 300s
    for val in tqdm(range(int(300 / self.time_step))):
        t = val * self.time_step
        if self.mode == 1:
            self.reactivity_function(t)
        neutron_density = self.neutron(t)
        precursor_density = []
        for j in range(6):
            precursor_density.append(self.precursor(j, t))

        if val % 10 == 0:
            f.write('%.4f' % t + " ")
            f.write('{:.4e}'.format(neutron_density) + " ")
            for j in range(6):
                f.write('{:.4e}'.format(precursor_density[j]) + " ")
            f.write("\n")
    f.close()
```

Execution of the point kinetics equation

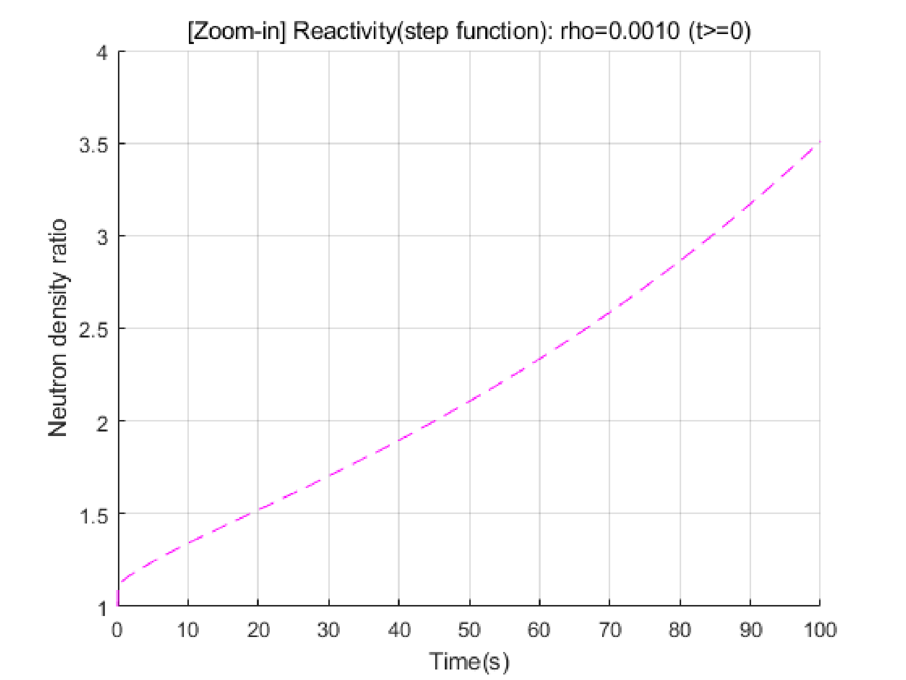
```
(untitled) C:\Users\John\PycharmProjects\untitled>python PyPKE.py --mode 0 --r -.0511
Mode 0 : step function
100%|██████████| 3000000/3000000 [00:24<00:00, 124984.38it/s]

(untitled) C:\Users\John\PycharmProjects\untitled>python PyPKE.py --mode 1 --r -.0511
Mode 1 : ramp function
100%|██████████| 3000000/3000000 [00:25<00:00, 118582.14it/s]

(untitled) C:\Users\John\PycharmProjects\untitled>python PyPKE.py -h
usage: PyPKE.py [-h] [--k K] [--r R] [--mode MODE]

optional arguments:
  -h, --help   show this help message and exit
  --k K        Multiplication factor
  --r R        Reactivity
  --mode MODE  default : step / 1 : ramp_dec / 2: ramp_inc
```

Adding options to PyPKE

PyPKE features

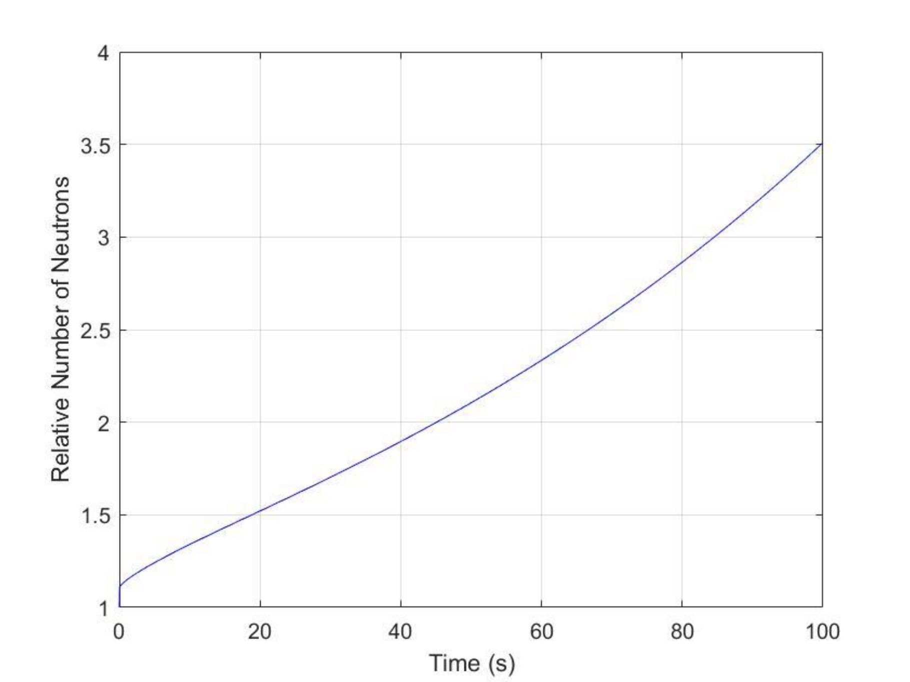
Prints out data into *.dat file format
⇒ Easy to use in other programs
⇒ Each column consists of time, neutron density ratio and precursor density ratios from 1 to 6

Results and Discussions
PyPKE($h = 0.001$) vs Analytical solution



PyPKE data on step reactivity function

$$\rho = \begin{cases} 0.0010 & t \geq 0 \\ 0 & else \end{cases}$$



Reference graph from 'Effect of Ramp Rate on Number of Neutrons(2020)'- 조규행

Imitation of the CR rod drop experiment

Conditions)

- Final reactivity = -0.0511

  ⇒ NOT accurate. Human obtained result.

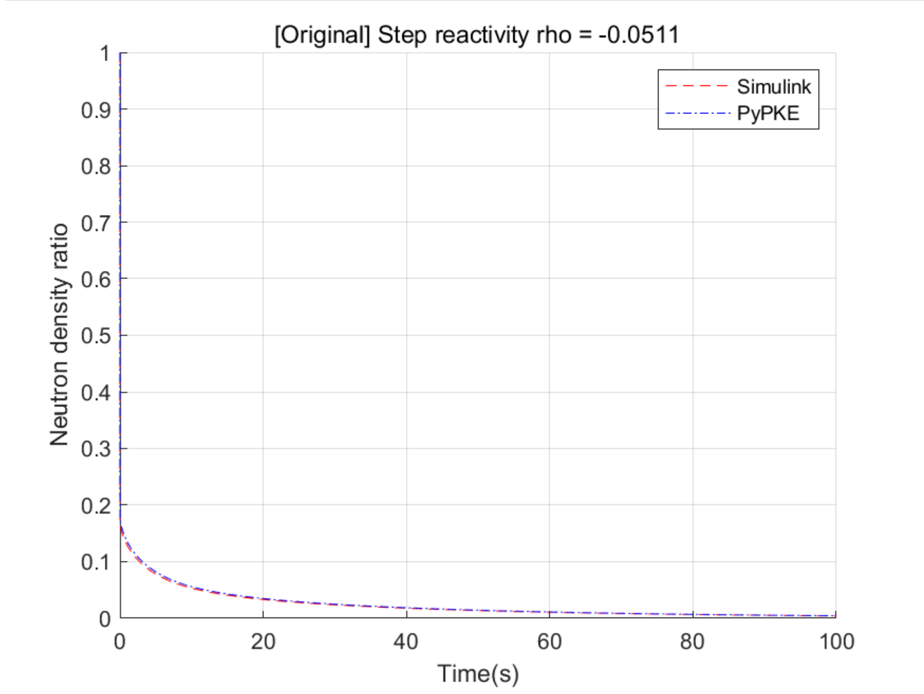
- Drop time = 47 seconds

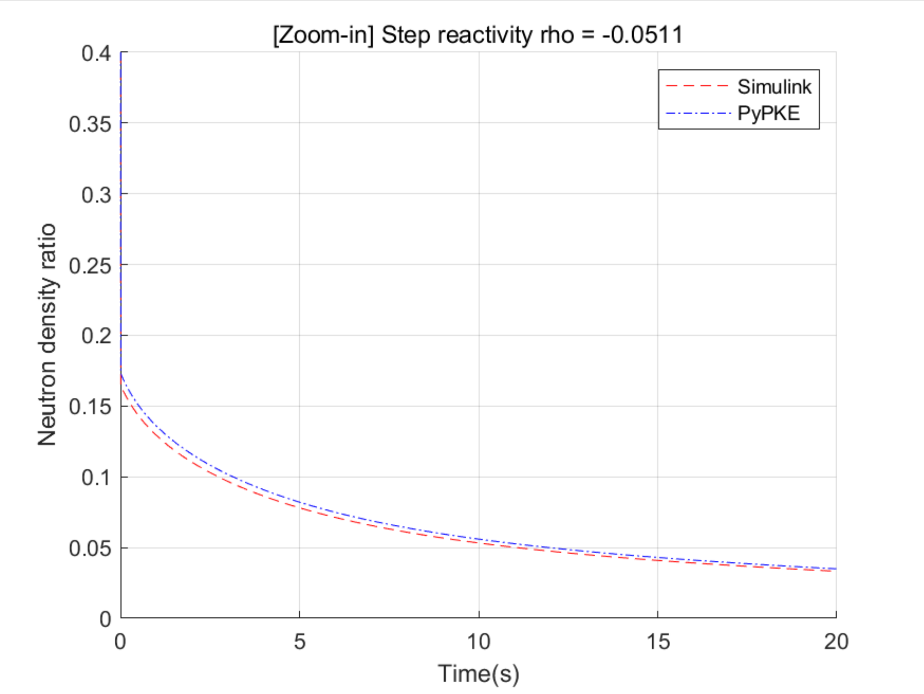
  ⇒ (time when all rod drop finished) – (time when rod drop started)

Results and Discussions

PyPKE($h = 0.0001$) vs Simulink($h = 0.01$)

See appendix for more data





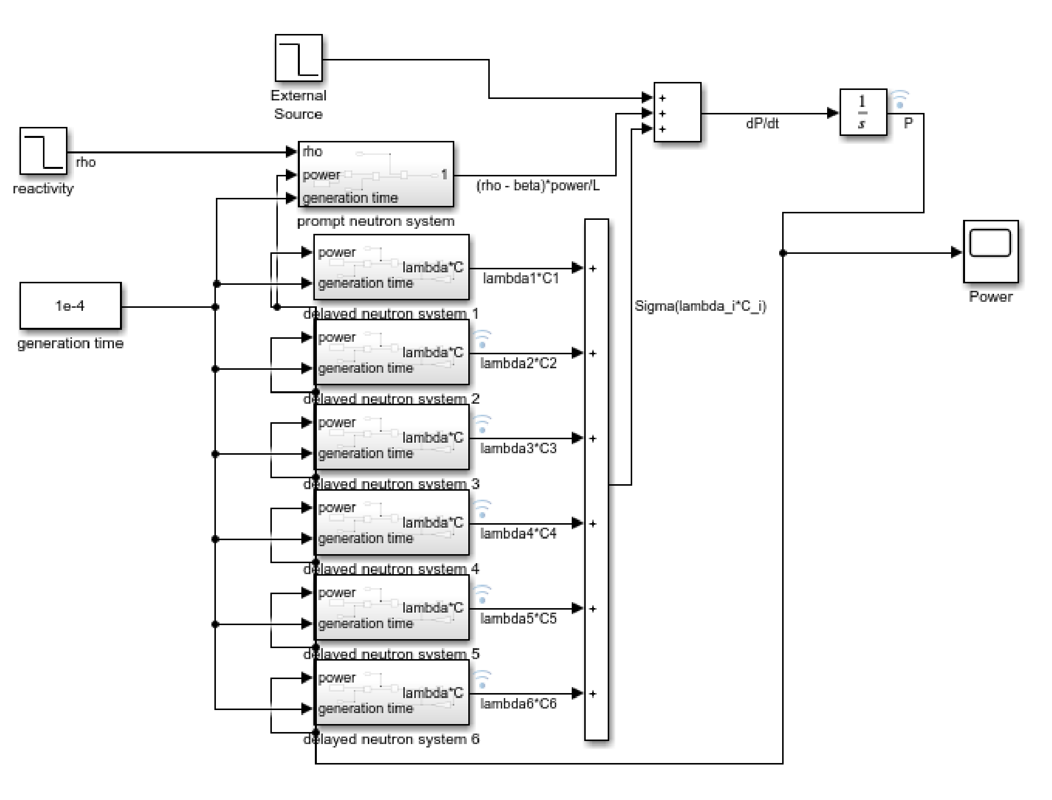
Very small difference at early evolution, but converges as time passes
Odd because precursor density ratio matched with negligible difference

Example) Precursor #6

Simulink	1	0.97952	0.955446	0.932049	0.909331	0.887224	0.865794	0.844975
PyPKE	1	0.979297	0.955215	0.931819	0.909099	0.887034	0.865605	0.844794

Results and Discussions

PyPKE vs Simulink both $h = 0.001$



Fundamentally, Simulink and PyPKE shares the same algorithm; addition by iteration.

The only difference is that in Simulink, the derivative passes an integrator which then accumulates the value with the time-step(optional) and ODE options.

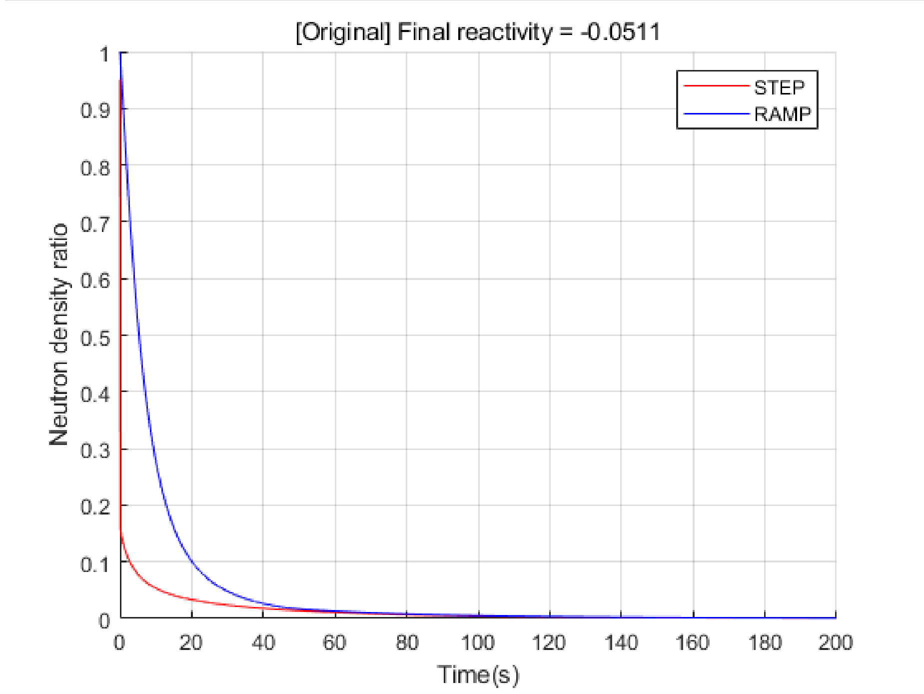
Here the options used were time-step 0.01 and ODE 14x(extrapolation).

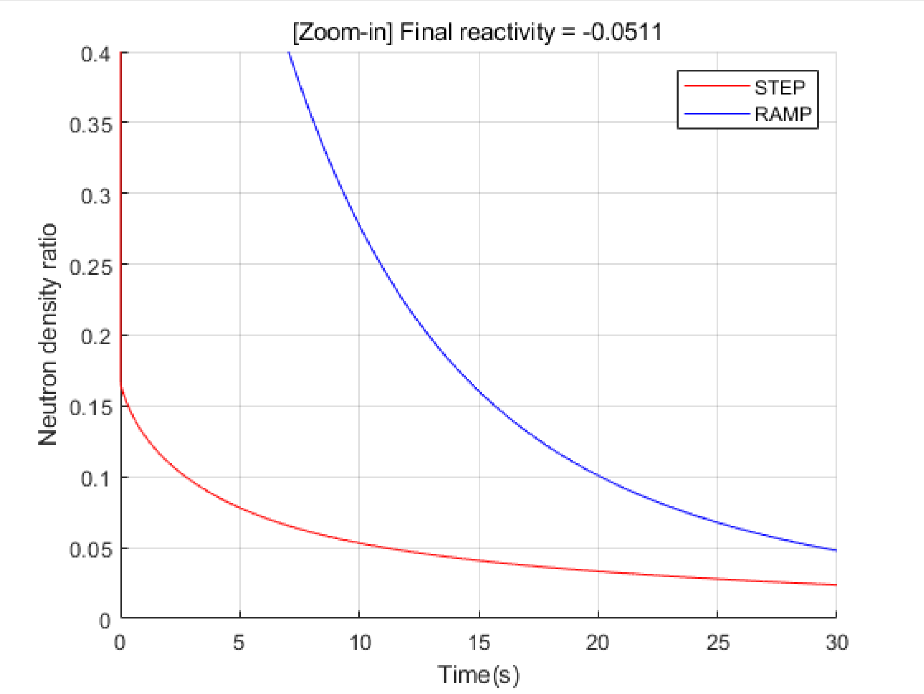
Revising the options to time-step 0.001 gave more precision to Simulink data, however matching PyPKE with the same data is not efficient when it came to comparison. Each algorithm should have different time-step with respect to the model.

PyPKE needs more delicate steps since it only handles the first order term. This should be taken into account.

Also, Simulink has various models for solving ODEs. Some options do not solve the problem at all, where some does. This variety of solution techniques make Simulink unique from PyPKE since it can handle more complex form of equations.

Results and Discussions
Rod drop method: Step vs Ramp

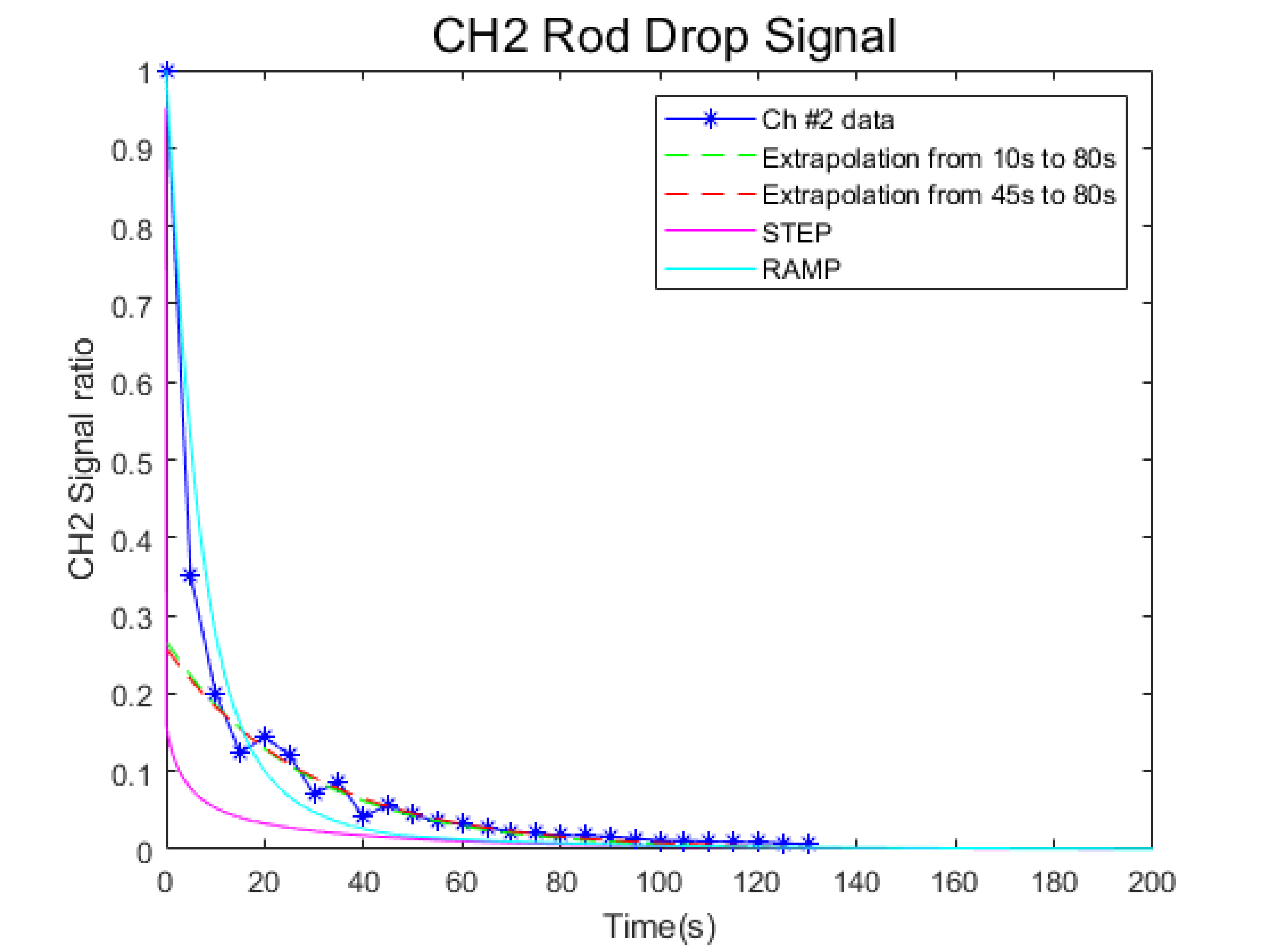




Expected result)
Step function reactivity immediately shifts the neutron density to a much lower level, whereas the ramp function slowly decays compared to the step function.

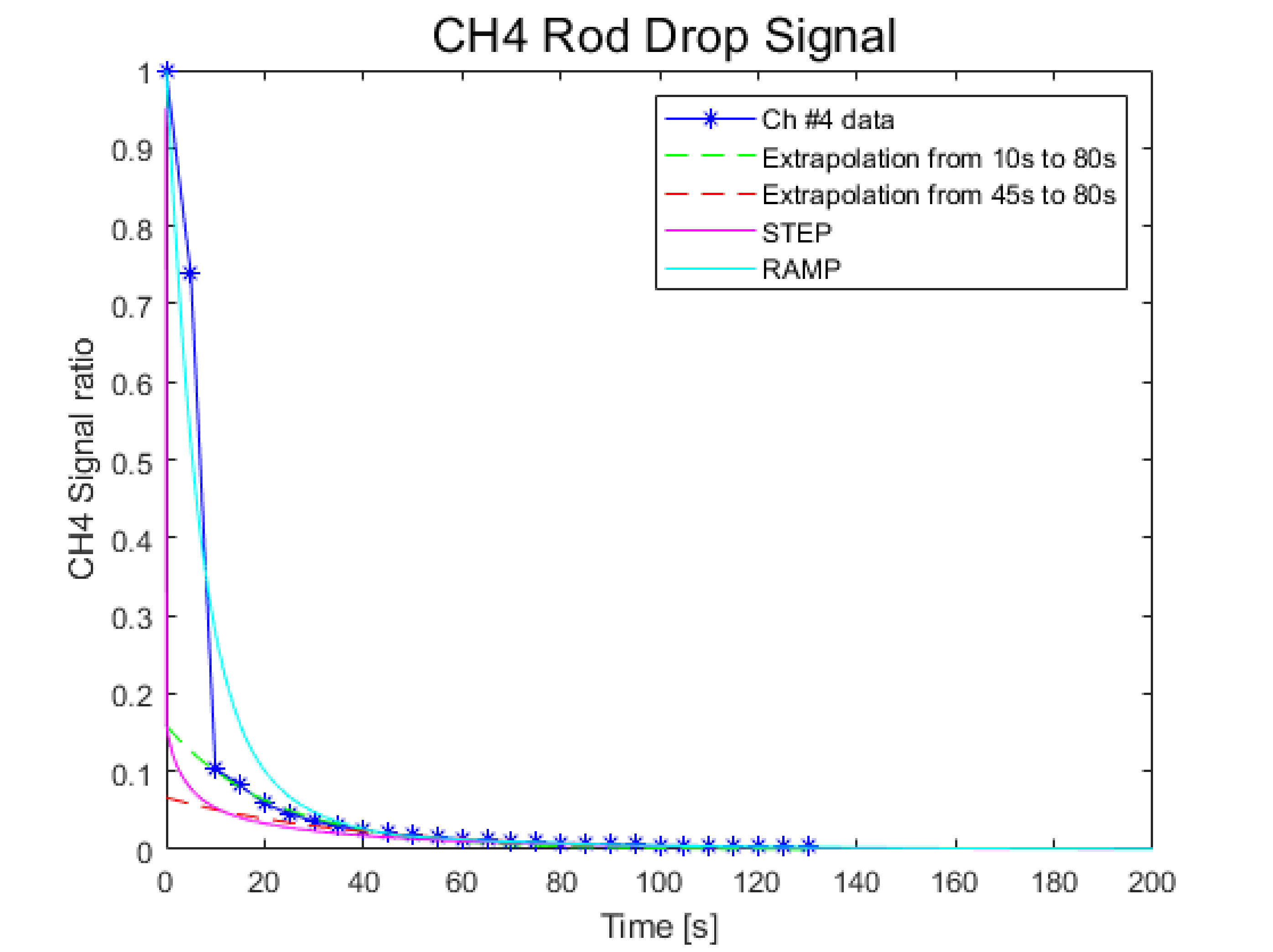
Results and Discussions

Rod drop method(Ch2): Human vs PyPKE



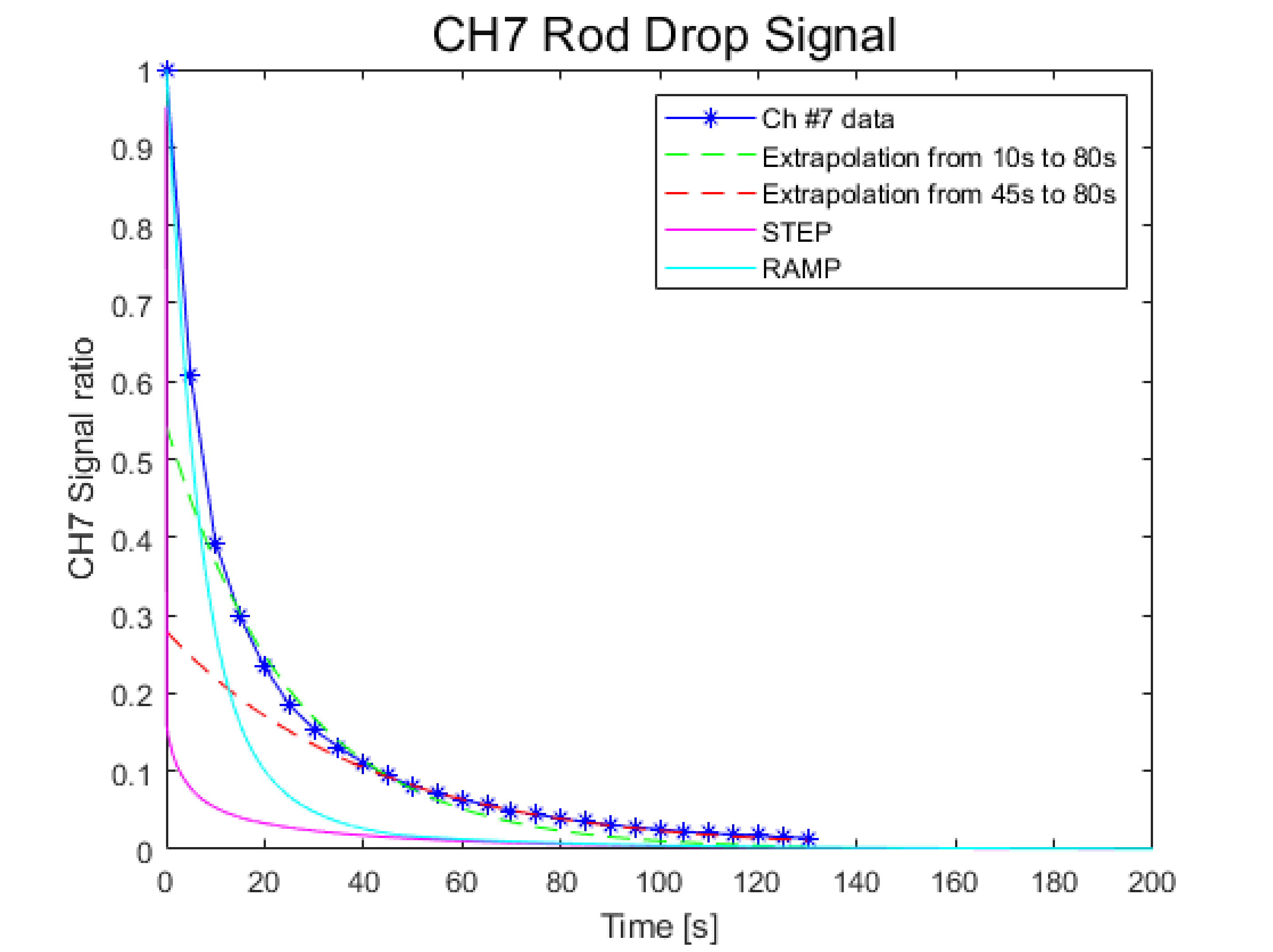
Results and Discussions

Rod drop method(Ch4): Human vs PyPKE



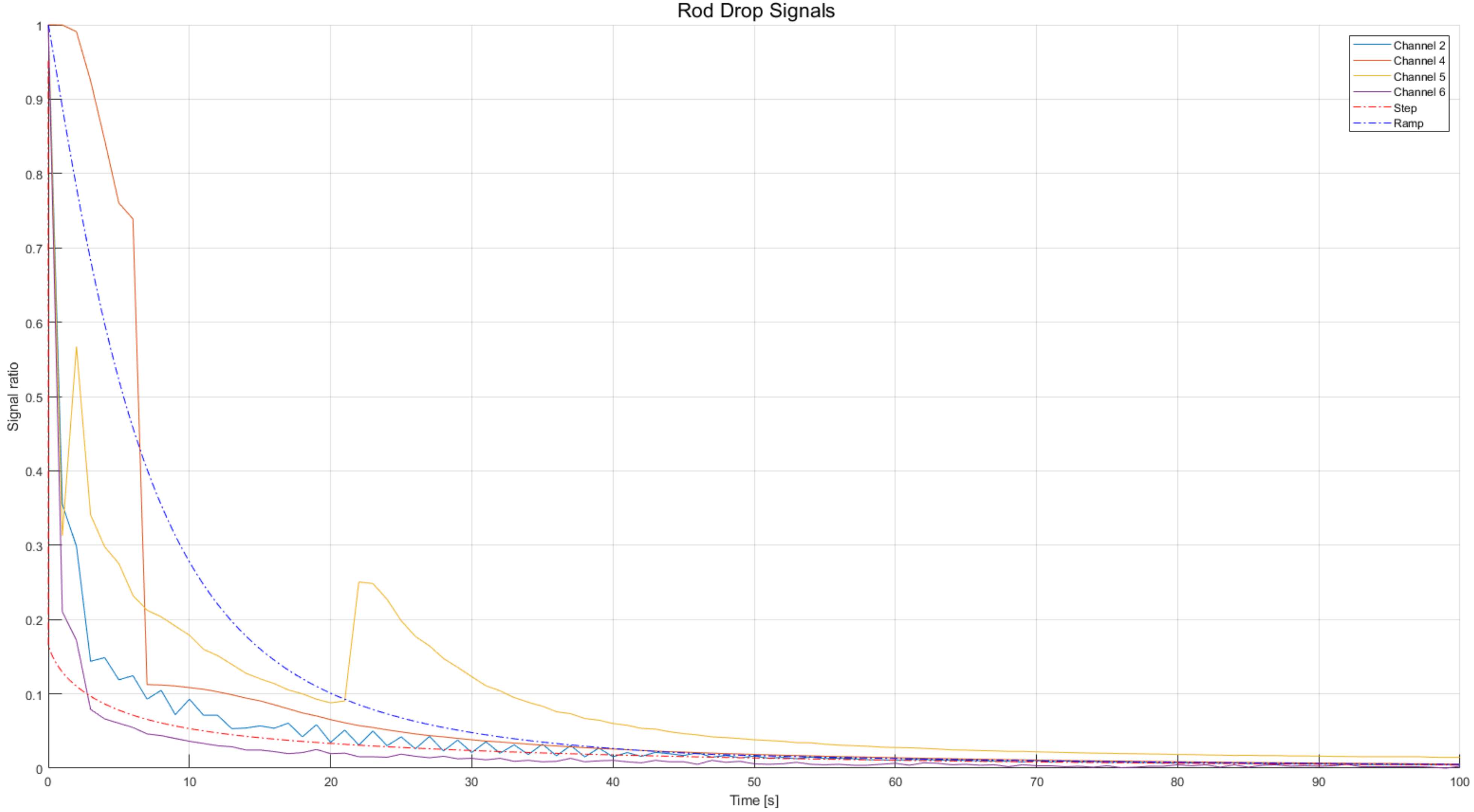
Results and Discussions

Rod drop method(Ch7): Human vs PyPKE



Results and Discussions

Rod drop method: Detectors vs PyPKE



Results and Discussions

Comparison to other methods($h = 0.001$)

Parameters used

- $\lambda_i = 0.0127, 0.0317, 0.155, 0.311, 1.4, 3.87$
- $\beta_i = 0.000266, 0.001491, 0.001316, 0.002849, 0.000896, 0.000182$
- $\Lambda = 0.00002$

Results and Discussions

Comparison to other methods($h = 0.001$)

Results from prompt step reactivity $\rho = 0.003$

Time (s)	CORE	PCA	Taylor(McMahon)	Taylor(PyPKE)	Analytical
T = 1	2.2098	2.2098	2.2099	2.2256	2.2098
T = 10	8.0192	8.0192	8.0192	8.8059	8.0192
T = 20	28.297	28.297	28.297	34.372	28.297

Results from prompt step reactivity $\rho = 0.007$

Time (s)	CORE	PCA	Taylor(McMahon)	Taylor(PyPKE)	Analytical
T = 0.01	4.5088	4.5088	4.5086	4.5337	4.5088
T = 0.5	5.3458E+03	5.3459E+03	5.3447E+03	5.7525E+03	5.3459E+03
T = 2	2.0600E+11	2.0591E+11	2.0566E+11	2.8288E+11	2.0591E+11

Results and Discussions
Comparison to step size

Results from prompt step reactivity $\rho = 0.003$

Time (s)	h = 0.001	h = 0.0001	h = 0.00001	Analytical
T = 1	2.2248	2.2248	2.2248	2.2098
T = 10	8.7987	8.7994	8.7995	8.0192
T = 20	34.325	34.330	34.331	28.297

Additional note)

According to McMahon and Professor Shim, the time-step $\boldsymbol{h}$ is important when it comes to numerical analysis. In McMahon's paper, the scale of the generation time is dependent to time-step $\boldsymbol{h}$ when it comes to accuracy.

Conclusion

1. PyPKE shows good accuracy compared to Simulink and eigen-value solutions.
2. However more research must be done for precision of the software.

   ⇒Considering C language based calculation for more elementary control of variables.
3. The complexity of the algorithm is $O(n)$, compared to the eigenvalue solution which computes the inverse matrix making the complexity $O(n^2)$ (Gauss-Jordan method) , PyPKE is more efficient
4. Still, it does not show excellent solutions compared to other algorithms.

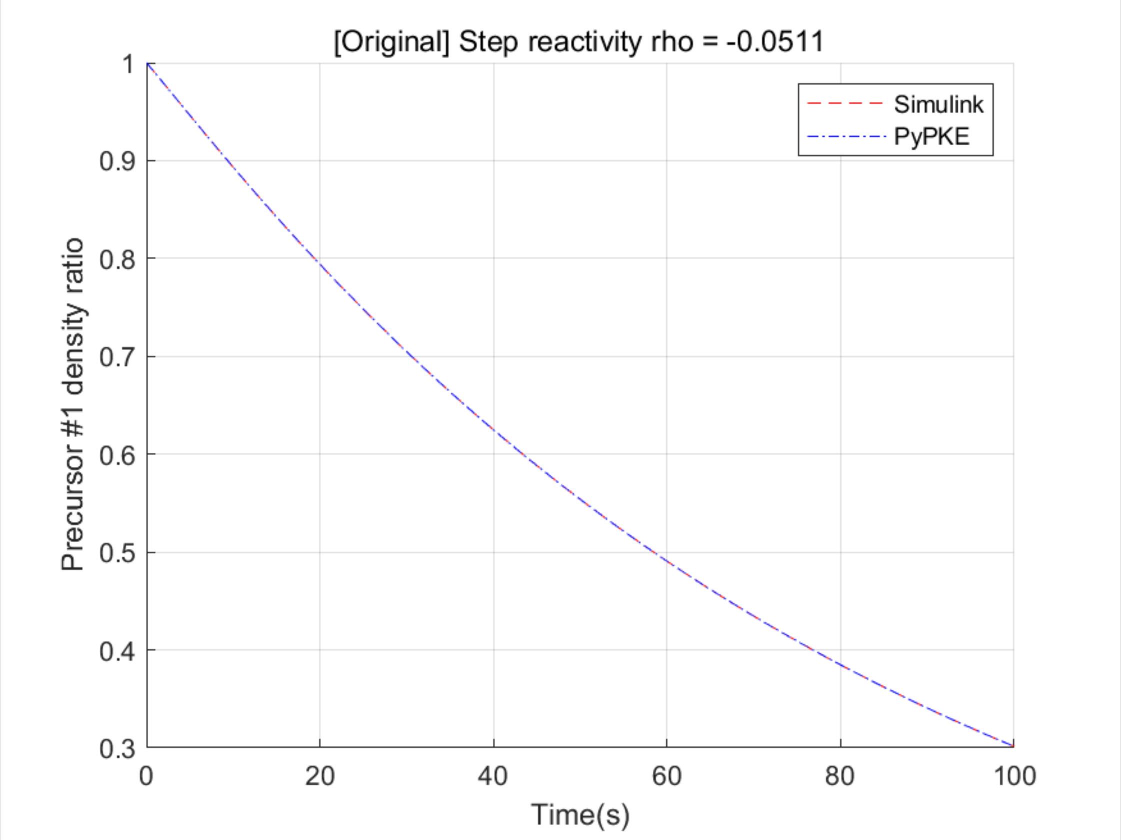
   ⇒ Still couldn't figure this out, will keep trying to debug

Thank you

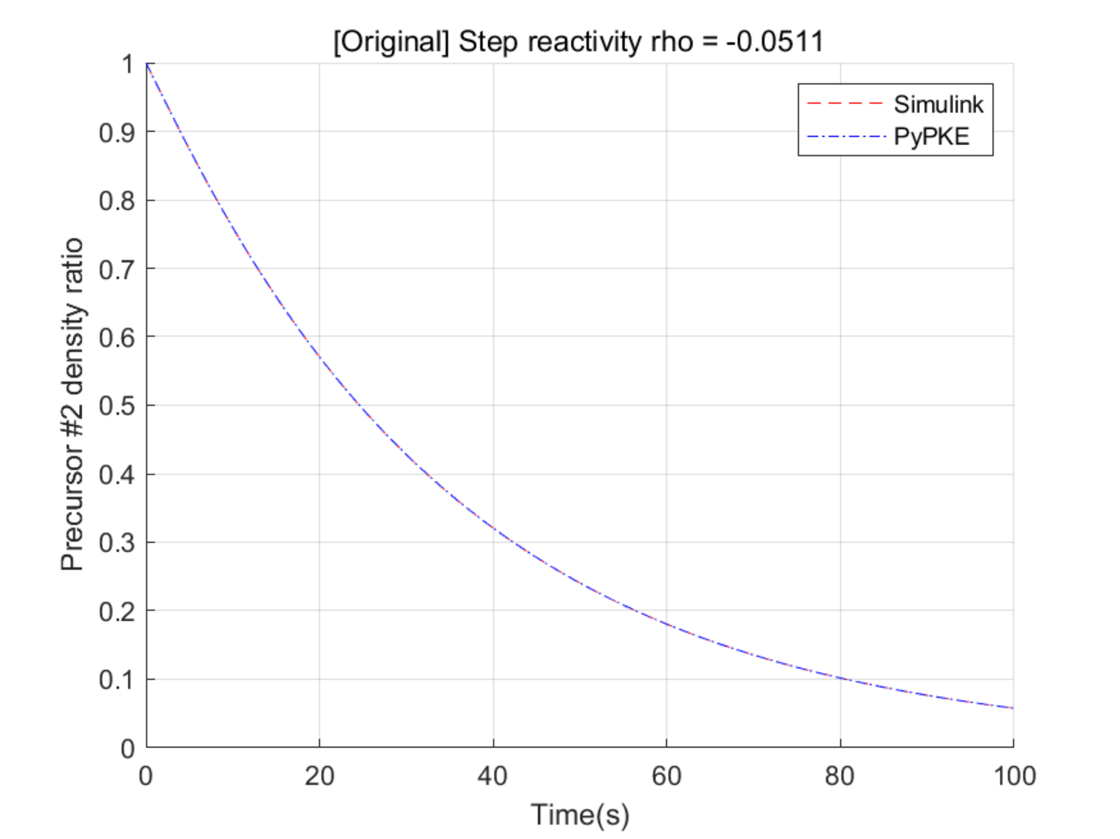
Reference

1. H. J. Shim (2012) McCARD: Monte Carlo code for advanced reactor design and analysis. Korean Nuclear S ociety, Daejeon
2. McMahon, D., & Pierson, A. (2010). A Taylor series solution of the reactor point kinetics equations. arXiv pr eprint arXiv:1001.4100.
3. K. H. Jo (2020) Effect of Ramp Rate on Number of Neutrons.
4. Petersen, C. Z., Dulla, S., Vilhena, M. T., & Ravetto, P. (2011). An analytical solution of the point kinetics eq uations with time-variable reactivity by the decomposition method. Progress in Nuclear Energy, 53(8), 109 1-1094.
5. Kinard, M. (2003). Efficient numerical solution of the point kinetics equation (Doctoral dissertation, Texas T ech University).

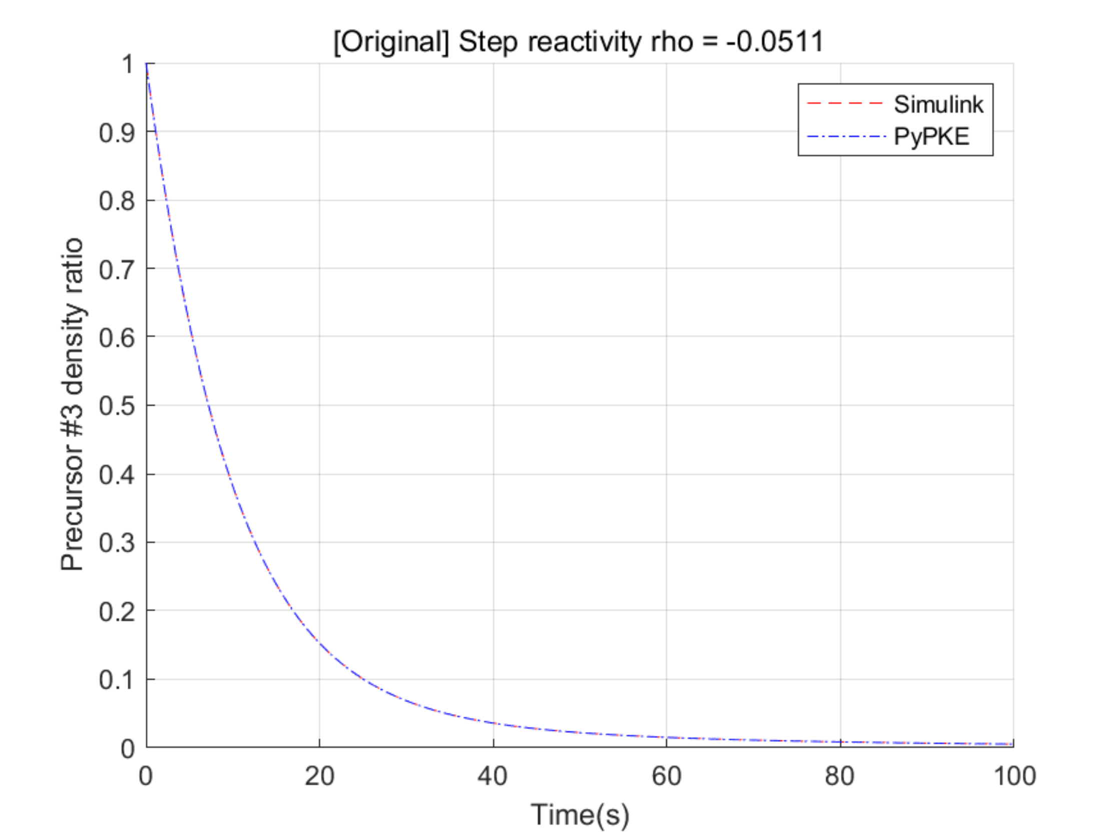
Appendix
PyPKE vs Simulink – Precursor #1



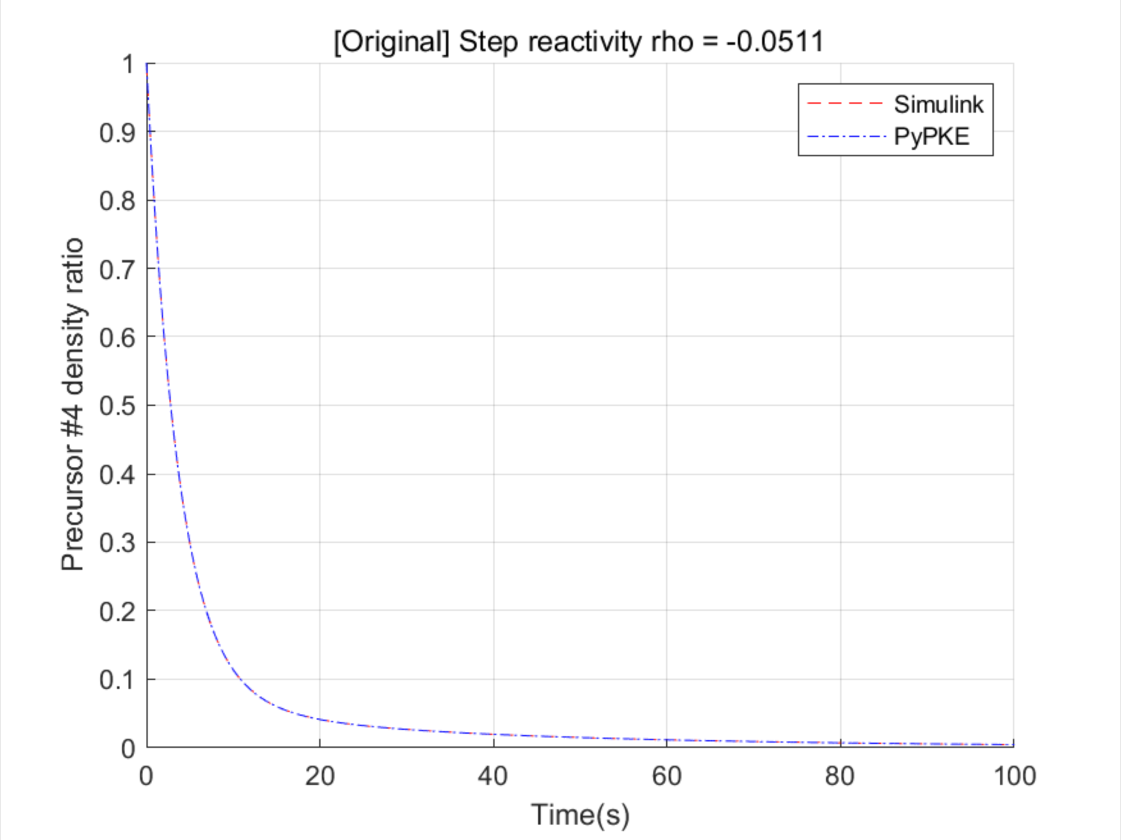
Appendix
PyPKE vs Simulink – Precursor #2



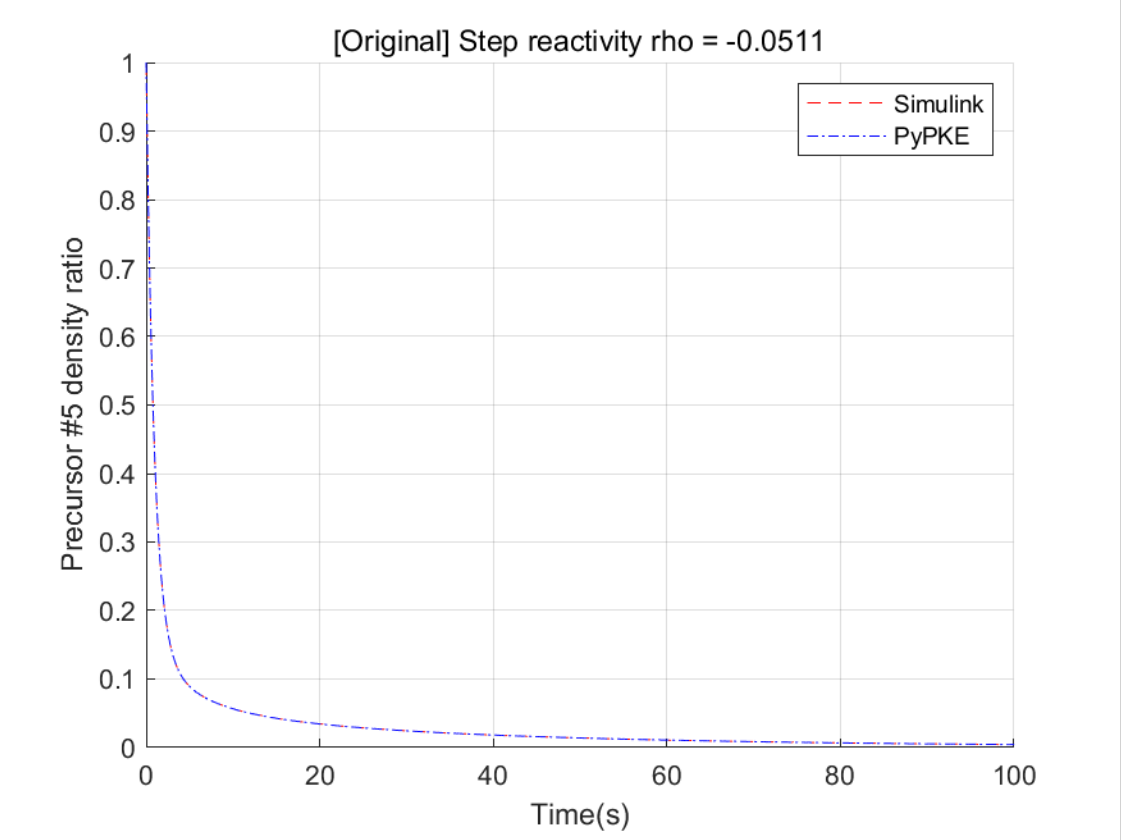
Appendix
PyPKE vs Simulink – Precursor #3



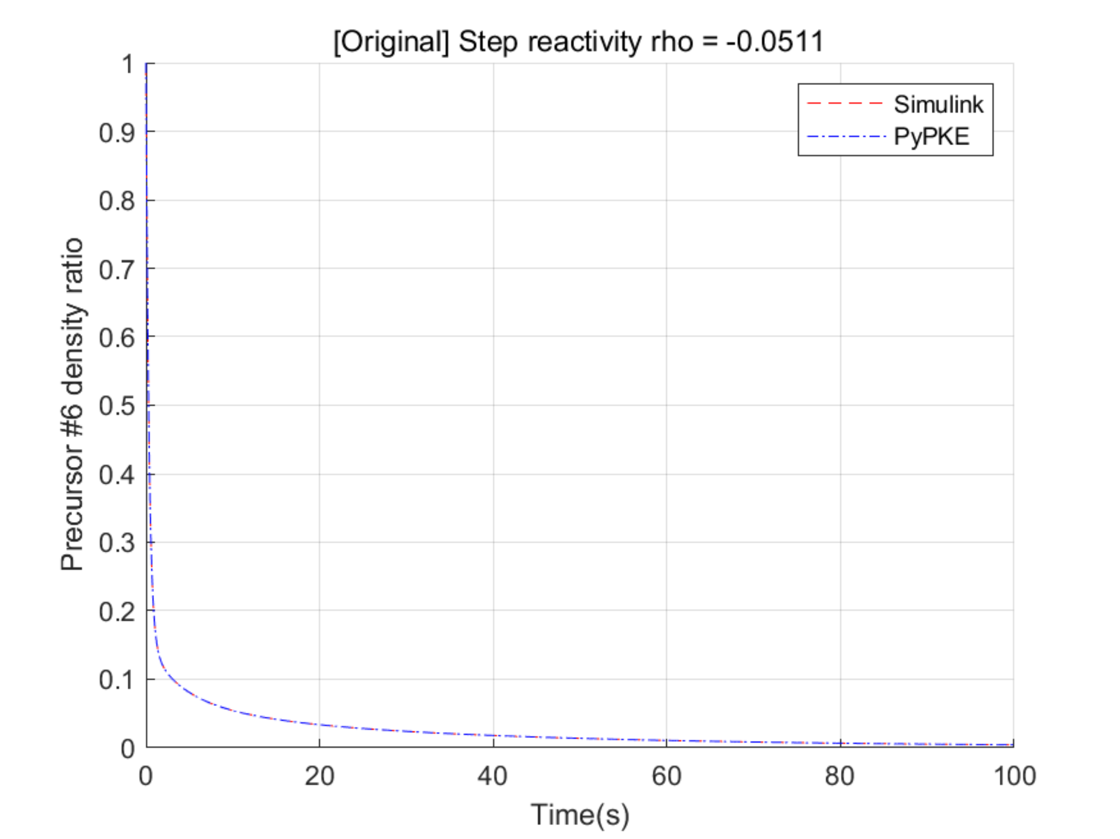
Appendix
PyPKE vs Simulink – Precursor #4



Appendix
PyPKE vs Simulink – Precursor #5



Appendix
PyPKE vs Simulink – Precursor #6



Appendix
PyPKE.py

https://github.com/ComputelessComputer/PyPKE

1. **Will provide README.md with specific details**
2. **Will update with feedback algorithm**
3. **Will update with reactivity evolution algorithm**