01a_model_setup

March 3, 2021

1 Loading PyMT models

In this tutorial we will learn how to use *PyMT* to: * Find and load models into a PyMT environment * Look at a model's metadata * Set up a model simulation

We will also learn about the common interface for *PyMT* models.

For more information you can look at the pymt documentation

1.1 The *PyMT* model library

All of the models that are available through *pymt* are held in a variable called, `MODELS`.

To have a look at what models are currently available, we'll import the model collection and print the names of all of the models.

[ ]:
```
from pymt import MODELS
```

You can think of `MODELS` as a Python dictionary where the keys are the names of the models and the values the model classes. For example, to see the names of all of the available models, you can use the `keys` method, just as you would with a `dict`.

[ ]:
```
MODELS.keys()
```

The class associated with each model can be accessed either as an attribute of `MODELS` or, in the usual way, using square brackets. That is, the following two methods are equivalent.

[ ]:
```
MODELS.Child is MODELS["Child"]
```

As with other Python objects, you can access a model's docstring with the `help` command.

[ ]:
```
help(MODELS.Child)
```

1.2 Exercise: Load a model and have a look at its documention

Load a couple other models and have a look at their documentation. You should notice some similarities in how the models are presented.

[ ]:
```
# Your code here
```

[ ]:
```
help(MODELS.Cem)
```

[ ]:
```
help(MODELS.FrostNumber)
```

1.3 Model documentation

As you will hopefull have seen, each model comes with: * A brief description of what it does * A list of the model authors * Other model metadata such as, * version number * Open Source License * DOI * Link to additional model documentation * A list of, *bibtex* formatted, references to cite * The usual docstring sections with method descriptions, parameters, etc.

Much of the above metadata is also available programmatically through attributes of a model instance. First we create an instance of the model.

(for those who might not be familiar with Python conventions, *camelcase* indicates a *class*, lowercase an *instance*, and a *constant*)

[ ]:
```
child = MODELS.Child()
```

Now try to access metadata as attributes. Hopefully, you'll be able to guess the attribute names.

[ ]:
```
print(child.version)
```

[ ]:
```
for ref in child.cite_as:
    print(ref)
```

2 Set up a model simulation

Model simulations can be either setup by hand or using the `setup` method of a *PyMT* model. If using `setup`, *PyMT* will take one of more model template input files and fill in values provided as keyword arguments to `setup`.

You can see what the valid keywords for model's `setup` method is under the *Parameters* section in a models docstring. Each parameter has a description, data type, default value and units.

[ ]:
```
help(child)
```

You can obtain a description of the default parameters programmatically through the *defaults* attribute.

[ ]:
```
for param in child.defaults:
    print(param)
```

By default, `setup` will create a set of input files in a temporary folder but you can change this behavior by providing the name of a another folder to put them into. To demonstrate what's going on, let's create a new folder in the current directory and see what happens.

[ ]:
```
child.setup("child-sim-0")
```

`setup` returns a tuple of the main configuration file and the simulation folder. Within that folder will be one or more input files specific to the model you're using.

[ ]:
```
ls child-sim-0
```

[ ]:
```
cat child-sim-0/child.in
```

3 Exercise: Setup a simulation using non-default values

Use a model's `setup` method to create a set of input files using non-default values and verify the input files were created correctly.

[ ]:
```
# Your code here
```

[ ]:
```
from pymt import MODELS

hydrotrend = MODELS.Hydrotrend()
dict(hydrotrend.parameters)
```

[ ]:
```
hydrotrend.setup("hydrotrend-sim-0", hydraulic_conductivity=200.0)
```

[ ]:
```
dict(hydrotrend.parameters)["hydraulic_conductivity"]
```

[ ]:
```
ls hydrotrend-sim-0
```

[ ]:
```
cat hydrotrend-sim-0/HYDRO.IN
```

4 Exercise: As a bonus, set up a series of simulations

Pull parameters from a uniform distribution, which could be part of a sensitivity study.

[ ]:
```
# Your code here
```

[ ]:
```
import numpy as np

hydrotrend = MODELS.Hydrotrend()

values = np.linspace(50., 200.)
for sim_no, value in enumerate(values):
```

```
    hydrotrend.setup(f"hydrotrend-sim-{sim_no}", hydraulic_conductivity=value)
```

[ ]:
```
cat hydrotrend-sim-12/HYDRO.IN | grep "hydraulic conductivity"
```

[ ]:
```
print(values[12])
```

[ ]:
```
cat hydrotrend-sim-24/HYDRO.IN | grep "hydraulic conductivity"
```

[ ]:
```
print(values[24])
```