Tutorial-1_IntroTo_emodpy-hiv

June 13, 2024

1 Introduction to emodpy-hiv

`emodpy-hiv` is the new way for configuring, running, and analyzing EMOD-HIV simulations. It replaces the previous DtkTools/HIV-Analyzers/run_scenarios.py.

One of the goals of this new tool is to simplify the configuration of EMOD by allowing the user to do it via Python instead of JSON. This provides several benefits, including: - Eliminating the need to worry about where parameters belong in deeply nested JSON. - Detecting configuration errors in the Python code before the simulations are created. - Using a 'for-loop' to vary a single value instead of copy and pasting large chunks of JSON. - Leveraging a Python IDE to provide tool-tips or navigate into functions to see the definition of the arguments/parameters.

1.1 The packages of emodpy

The emodpy "framework" consists of a collection of packages. Each package provides a set of features related to configuring and running EMOD.

- **emod-api**
  - Defines base-level objects and how the input files of EMOD are written or read.
- **emod-hiv** (notice no "py")
  - Contains a schema and Eradication binary.
- **emodpy**
  - Provides the common set of EMOD configuration features used by disease-specific versions. Users generally do not need to know much about this package.
- **emodpy-hiv/malaria**
  - These disease-specific packages should be the primary packages users utilize for configuring EMOD for the disease they are working on. These packages are a "subclass" of emodpy.
- **idmtools**
  - Provides a low-level workflow framework. It offers an abstract way to run EMOD on different platforms and supports a workflow that involves suites, experiments, and simulations. By itself, it is model-agnostic, similar to a model-agnostic DtkTools.

1.2 Installation

- `mkdir emodpy_demo`
- `cd emodpy_demo`
- `python -m venv env`
- `env/Scripts/activate`

- `pip install --index-url https://packages.idmod.org/api/pypi/pypi-production/simple emodpy-hiv`

1.3 Manifest

Manifest.py is used to store settings that are either common across scripts for running experiments or vary for different users.

- The installed package includes a schema and executable.
- Manifest allows you to specify other executables and schema.

[1]:
```
from IPython.display import Code

Code(filename='manifest.py', language='python')
```

[1]:
```
import os

#
# This is a user-modifiable Python file designed to be a set of simple input
# file and directory settings that you can choose and change.
#

# === Directory for Executables ===
# This folder will contain your Eradication binary, schema, SIF files and other
# things that you might need to run EMOD.
executables_dir="executables"

# === Singularity Image ===
# the location of the file containing AssetCollection id for the dtk sif
sif_path = os.path.join(executables_dir, "dtk_sif.id")

# === Schema ===
# The script is going to use this to store the downloaded schema file.
# Create 'download' directory or change to your preferred (existing) location.
schema_file = os.path.join(executables_dir, "schema.json")

# === Executable ===
# The script is going to use this to store the downloaded Eradication binary.
# Create 'download' directory or change to your preferred (existing) location.
eradication_path = os.path.join(executables_dir, "Eradication")

# === Assets ===
# Create 'Assets' directory or change to a path you prefer.
# idmtools will upload files found here.
assets_input_dir = "Assets"

# === Embedded Python Scripts ===
# This is a location for EMOD embedded python scripts.  Scripts included
# here will be included in your simulation directory so that EMOD
# can use them during the simulation.
ep4_path = "embedded_python"
```

1.4 Eradication binary/executable - emod-hiv

The actual disease model for EMOD is an executable/binary file typically called Eradication(.exe). The purpose of emodpy-hiv is to configure and run this binary. Hence, you need a binary before you can do anything. To make this easier, an emodpy-hiv package requirement is that you also get the emod-hiv binary package (notice no "py" after the "emod"). This package contains an executable and an associated schema file that you can use with this version of emodpy-hiv.

Notice how the `dtk.setup()` method has an argument called `local_dir`. The value of this argument will be the path/directory where the executable and schema files will get stored. You can configure this in the `manifest.py` file.

[2]:
```
import manifest
import emod_hiv.bootstrap as dtk

dtk.setup(local_dir=manifest.executables_dir)
```

1.5 Platform

A Platform defines the type of cluster/computing platform that EMOD will run on. It provides an abstract way to run simulations on different hardware.

- `"SLURM_LOCAL"` - This value indicates that you are going to run your simulations on a SLURM-based HPC cluster.
- `job_directory` – The directory where your experiment and simulation directories will be created.
- For more information, see https://docs.idmod.org/projects/idmtools/en/latest/platforms/slurm/index.html

[3]:
```
from idmtools.core.platform_factory import Platform

#platform = Platform( "SLURM_LOCAL", job_directory="experiments" )
platform = Platform( "Calculon", node_group="idm_abcd", priority="Normal" )
```

```
/!\ WARNING: File 'idmtools.ini' Not Found! For details on how to configure
idmtools, see
https://docs.idmod.org/projects/idmtools/en/v1.7.1/configuration.html for
details on how to configure idmtools.


[Calculon]
{

   "endpoint": "https://comps.idmod.org",

   "environment": "Calculon"

}
```

1.6 EMODHIVTask

Configures the files for each simulation in an experiment. It is found in the `emodpy-hiv` package. `EMODHIVTask` is specifically designed to work with the emodpy-hiv country models; therefore, do not use `emodpy.EMODTask` directly.

- `eradication_path` – The path to the Eradication binary.
- `schema_path` – The path to a processed `schema.json` file that comes from the Eradication binary.
- `param_custom_cb` – The function used to set the configuration parameters.
- `campaign_builder` – The function that will build a campaign file for each simulation.
- `demog_builder` – The function that will build a demographics file for each simulation.
- `ep4_path` – The path to a directory with EMOD embedded Python scripts

For this initial example, we will use the build methods that come with the Zambia country model. Later, we will create our verions of the build methods to change these defaults.

`emodpy_hiv.country_model` - This module contains code and data associated with different baseline country models. These models provide you with default build methods that you can use as-is or customize to change the default behavior.

[4]:
```
# Will make the warnings off by default in 2.0
import emod_api.schema_to_class as s2c
s2c.show_warnings = False

import emodpy_hiv.emod_task as emod_task
import emodpy_hiv.country_model as cm

zambia = cm.Zambia()
task = emod_task.EMODHIVTask.from_default(
    eradication_path = manifest.eradication_path,
    schema_path      = manifest.schema_file,
    param_custom_cb  = zambia.build_config,
    campaign_builder = zambia.build_campaign,
    demog_builder    = zambia.build_demographics,
    ep4_path         = None
)
```

```
setting schema_path: executables\schema.json to campaign_builder function.
Generating demographics file demographics.json.
Telling emod-api to use executables\schema.json as schema.
```

1.7 Singularity image

We use a a Singularity image as a container within which EMOD will run. This container provides all of the prerequisites (i.e., installation requirements) required by EMOD. On SLURM, you must define the path to your image file. Note that we define this path in your manifest file.

[5]:
```
#task.set_sif( path_to_sif=manifest.sif_path, platform=platform )
task.set_sif( path_to_sif=manifest.sif_path )
```

1.8 SimulationBuilder

One more key element that we need for our experiment is `SimulationBuilder`. We use it to create ("build") the collection of simulations to include in the experiment. The builder uses the concept of "parameter sweeps" to determine which simulations to create. This means that we can define sweeps of different parameters to create various combinations of parameters. The builder will create one simulation for each combination.

For our first time creating a builder, we will do a simple sweep on `Run_Number`. `Run_Number` is the random number seed to help us get different statistical realizations of our scenario.

First, we define a function for changing the parameter. Configuration parameters are very easy to change, but as we will see in a later tutorial, the campaign and demographics require a bit more work. It is not much, but it is different. Notice that the function returns a "tag" name and its corresponding value. These tags can be used to sort or query your simulations - based on the tags.

Once we have this function, notice that we add it to the builder saying that it is a "sweep definition". In this definition, we specify our function and the list of values we want for the parameter. We list three values, so we should end up with three simulation runs where all of the inputs are the same except for this `Run_Number` parameter.

[6]:
```python
from idmtools.builders import SimulationBuilder

def sweep_run_number( simulation, value ):
    simulation.task.config.parameters.Run_Number = value
    return { "Run_Number": value }


builder = SimulationBuilder()
builder.add_sweep_definition( sweep_run_number, [1,2,3] )
```

1.9 Experiment

In EMOD vocabulary, an experiment is a collection of simulations or relaizations. Typically, a user decides that a collection of parameter sweeps belong together and defines an "experiement".

The first thing we do is to create an experiment using our `builder` and `task`. In this collection of tutorials, we will be doing it "from a builder", but there are other ways to create experiments.

Once we have our experiment created, we are ready to run our simulations. Our experiement object will handle everything required to run the simulations on our platform. It will perform tasks such as: - Submitting jobs on the platform/cluster - Managing the running of all the simulations - Retrying a simulation if it fails - Allowing you to relax while it is doing the work

Notice that we set the `wait_until_done` parameter to `True`. This means that this Python program will wait at this line of code until all of the simulations have completed. Therefore, the execution of that line may take several minutes.

[7]:
```python
from idmtools.entities.experiment import Experiment

experiment = Experiment.from_builder( builder, task, name="Tutorial_1" )
```

```
experiment.run( wait_until_done=True, platform=platform )
```

```
The created experiment can be viewed at https://comps.idmod.org/#explore/Simulat
ions?filters=ExperimentId=29b36272-f829-ef11-aa14-b88303911bc1
Simulations are still being created
```

```
Creating Simulations on Comps:
100%|                        | 3/3 [00:00<00:00,
5.30simulation/s]
Waiting on Experiment Tutorial_1 to Finish running:
100%|            | 3/3 [01:48<00:00, 36.09s/simulation]
```

1.10 Results

For Tutorial #1, you need to check "experiments" directory that that we defined when we created `Platform`. You should see a folder in there for your first experiment. In that folder, you should see three simulation folders. In each of those folders, there should be an "output" folder containing a file called "InsetChart.json". If you have that, you are ready to start Tutorial #2.