How to integrate Dakota into your model

1 Introduction

Dakota is a powerful tool that "enable design exploration, model calibration, risk analysis, and quantification of margins and uncertainty with computational models."(`https://dakota.sandia.gov/`)

ABM4py contains some helpers to get your model running as a "analysis$_{\text{driver}}$", those are described here.

An "dakotarized" example is `examples/bass_diffusion/01_bass_network_dakota.py` where the `01_bass_network.py` example was changed so that it can be controlled by Dakota.

The dakota helpers are in the `dakota` subfolder. It's a good idea to keep it seperated from your model code.

2 Create a response-script

The results of a simulation are returned to Dakota via an own python script that is called after the simulation has run. As the script is called in the same python process that also called the simulation, you can access the results directly from the memory. The `global` namespace can be adjusted for the script if necessary (see `core.dakota.finished`), but if it's set to `globals()` the content of the script behave exactly the same as it was part of the model code itself. If you e.g. calculate the error by comparing the simulation results with real data in a function called calcError, the response-script could look like:

```
reportResponse('error', world.calcError())
```

The seperation of the response-script to the model-code allows to create different Dakota studies without the need to change any line in the model.

3 Adjust your model

You need to add two calls to `core.dakota` in your model, the first one is `core.dakota.activate()`, which should go to the start (e.g. just behind the `import` statements), as it also change the directory to the directory of the model code, so that e.g. reading resources will be behave as have you started the model manually. Calling `activate` will fail, when the model is run manually, as the `activate` method tries to read files which are created by Dakota. But you can switch your model between "manual" mode and "Dakota" mode just by uncomment this method call.

Add the end of your model you must add `core.dakota.finished(globals())` whereby `globals()` can be also replaced by another namespace. `finished` can be also called without `activate` before, in this case, the call will be ignored.

The parameters that are set by Dakota must be accessed in the model via `world.getParameter()`. The calls to `world.setParameter()` are ignored for parameters that are under Dakota's control. So if you have e.g. a lines like

```
IMITATION = 0.2
foo = IMITATION
```

you must rewrite this as

```
world.setParameter('imitation', 0.2)
foo = world.getParameter('imitation')
```

4 Create a config file

Dakota must know which parameters should be set by Dakota (e.g. with a uniform random value), and which results (respones) are returned. This in done via a "Dakota input file" (which has the file ending `.in`). This files can be written by hand (see the Dakota documentation), but we have added also some scripts that supports this process (you can also uses this scripts as a starting point and tweak the `.in` files afterwards). This is especially usefull if you have many parameters or responses which you can generate automatically (e.g. for a range of Years).

The following is the config file for the `bass_network_dakota` example

```
modelFileName = '../examples/bass_diffusion/01_bass_network_dakota.py'

responsesScript = './response-scripts/bass-sampling.py'

# (name, initial points, lowerLimit, upperLimit)
continuousVariables = [
    ('imitation', 0.2, 0.0, 0.5),
    ('innovation', 15, 10, 20)
]

# (name, text)
staticStrings = []

# (name, weight)
responses = [
    ('switchFraction', 1)
]
```

`modelFileName` is the file that will get executed by Dakota. The `responseScript` is called at the end of the simulation as described above.

`continuousVariables` are a list of variables that by Dakota with a uniform distribution between the lowerLimit and the upperLimit (the 3rd and 4th entry of the tuple). Some Dakota studies are needing a default value, this is the 2nd entry of the tuple.

`staticStrings` are a list of static strings, which will be the same for all simulations of the study. E.g. this can be used to define a parameter file that should be loaded by the simulation for the parameters that are not controlled by Dakota.

`responses` is a list of the results that are returned to Dakota after the simulation has finished via the `responseScript`, using the `reportResponse` function. The weight is used for some optimization studies.

5 Generate the Dakota input

Run the script `create-infile-from-template.py` to generate the Dakota input file. The script needs three arguments

1. fileName of template (dakota input file)
2. configuration fileName (python file)
3. fileName of generated dakota input file

so e.g.

```
python create-infile-from-template.py
    templates/sampling-template.in
    configs/examples-01-bass-network-dakota.py
    sample-bass-network.in
```

would generate the dakota input file `sample-bass-network.in`

Open this file to edit the study specific parts, e.g. the number of samples that should be drawn, or the `evaluation_concurrency` (which is set to 8 by default).

6 Run dakota

On the workstation just run: `dakota -i DAKOTA_INPUT_FILE`

On the eagle you must do this from a worker node which must load the `dakota/6.8` module.

7 Convert results for shiny

After running Dakota, you can create a parallel coordinate plot using shiny. The .dat table file created by Dakota has some strange formatting, but you can create use the `dakota_restart_util` to create a file that is readable from shiny:

```
dakota_restart_util to_tabular DAKOTA_INPUT_FILE.rst dakota.tabular
```

Then switch to the `shiny` subfolder and start shiny be entering

```
R -e "shiny::runApp(launch.browse=TRUE)"
```

This should open a brower (or add a new tab to an already open browser) which shows the plot. The plot is interactive, you can brush intervals of a coordinate to select runs that are in this interval.

There are two tabs, in the "combined" tab all inputs and outputs are drawn into a single plot. In the case that there are many inputs and outputs, it can be usefull to seperated them, as it is done in the second tab.