assignment 3 - multimodel_assignment

April 30, 2021

1 EPA1361 - Model-Based Decision Making

1.1 Multi-model analysis

This exercise uses a simple version of the Lotka-Volterra predator-prey equations to show how the EMA Workbench can be used for a multi-model analysis, in addition to typical parametric/structural uncertainties. This will let you test the connectors provided in the Workbench for Excel, NetLogo, and Vensim / PySD; we'll also use the models for the sensitivity analysis exercise in week 3.

- Using the three model files provided and the Python function below, define model objects for each implementation (Excel, NetLogo, Vensim/PySD, and Python), and test them using a single ensemble. Use 50 experiments sampled from the parameters below (so that each experiment will be executed for the 4 models, for a total of 200), and retrieve outputs for the *TIME*, *predators*, and *prey* variables.
  - excel and vensim are only supported on windows
  - vensim requires the DSS version of Vensim
  - Netlogo supoprt depends on jpype and pynetlogo. Also, if you don't have NetLogo installed, please get it from NetLogo 6.1.1
  - for pysd, see its documentation
  - If possible try to work with all model versions, but even 2 or 3 (pure python and something else should be sufficient).

Parameter	Range or value
prey_birth_rate	0.015 – 0.035
predation_rate	0.0005 – 0.003
predator_efficiency	0.001 – 0.004
predator_loss_rate	0.04 – 0.08
Final time	365
dt	0.25

- Note that your EMA Workbench installation includes example scripts for the different connectors. The different model objects follow a similar syntax but will need to be slightly adjusted depending on the software (e.g. to specify the NetLogo run length or the sheet name in Excel).

- These model objects can be used with a replication functionality (for instance to test the effect of stochastic uncertainty in a NetLogo model), which repeats a given experiment over multiple replications. You can use a single replication in this exercise as the models are not stochastic. By default, each outcome array will then have a shape of (# experiments, # replications, # time steps). Try adapting the outcome arrays so that they can be used with the *lines* plotting function of the Workbench, and plot the results grouped by model.

- To check the graphical results, find the maximum absolute error of the time series you obtained for the *prey* variable in the Excel, NetLogo, and Vensim/PySD models, relative to the Python function.

1.2 The model

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

from ema_workbench import (Model, RealParameter, TimeSeriesOutcome,␣
↪perform_experiments,
                           ema_logging)

from ema_workbench.connectors.netlogo import NetLogoModel
from ema_workbench.connectors.excel import ExcelModel
from ema_workbench.connectors.pysd_connector import PysdModel

from ema_workbench.em_framework.evaluators import LHS, SOBOL, MORRIS

from ema_workbench.analysis.plotting import lines, Density

from functions import fix_format


def PredPrey(prey_birth_rate=0.025, predation_rate=0.0015,␣
↪predator_efficiency=0.002,
             predator_loss_rate=0.06, initial_prey=50, initial_predators=20,␣
↪dt=0.25, final_time=365, reps=1):

    #Initial values
    predators, prey, sim_time = [np.zeros((reps, int(final_time/dt)+1)) for _␣
↪in range(3)]

    for r in range(reps):
        predators[r,0] = initial_predators
        prey[r,0] = initial_prey

        #Calculate the time series
        for t in range(0, sim_time.shape[1]-1):

            dx = (prey_birth_rate*prey[r,t]) -␣
↪(predation_rate*prey[r,t]*predators[r,t])
            dy = (predator_efficiency*predators[r,t]*prey[r,t]) -␣
↪(predator_loss_rate*predators[r,t])
```

```
            prey[r,t+1] = max(prey[r,t] + dx*dt, 0)
            predators[r,t+1] = max(predators[r,t] + dy*dt, 0)
            sim_time[r,t+1] = (t+1)*dt

    #Return outcomes
    return {'TIME':sim_time,
            'predators':predators,
            'prey':prey}
```

```
C:\Users\joren\anaconda3\envs\MBDM\lib\site-
packages\ema_workbench\connectors\__init__.py:17: ImportWarning: vensim
connector not available
  warnings.warn("vensim connector not available", ImportWarning)
```

[2]:
```
from ema_workbench import Model, RealParameter, ScalarOutcome, Constant,␣
↪MultiprocessingEvaluator, SequentialEvaluator

uncertainties = [RealParameter('prey_birth_rate', 0.015, 0.035),
                       RealParameter('predation_rate', 0.0005, 0.003),
                       RealParameter('predator_efficiency', 0.001, 0.004),
                      RealParameter('predator_loss_rate', 0.04, 0.08)]

constants = [Constant('Final Time', 365),
                  Constant('Time step', 0.25)]

outcomes = [TimeSeriesOutcome('prey')]

constants2 = [Constant('dt', 0.25)]

constants3 = [Constant('final_time', 365),
                  Constant('dt', 0.25)]
```

[3]:
```
#LHS, SOBOL or MORRIS
US = LHS
processes = 10
```

In the file functions.py is a function that is used to fix the output format of the different model experimentations.

1.2.1 Python model test

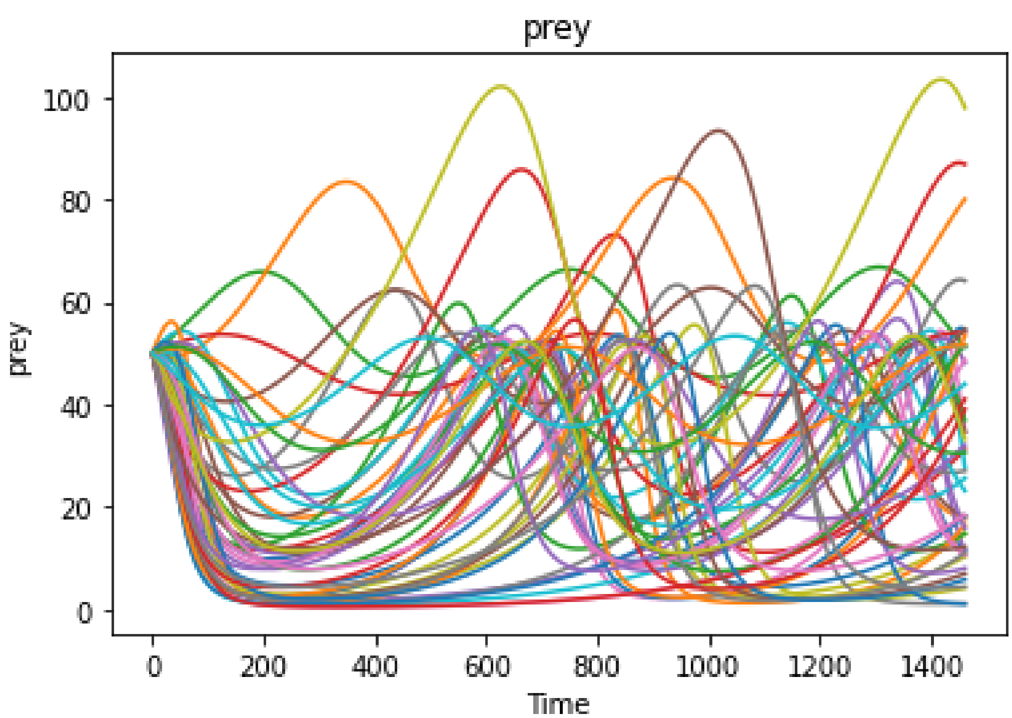
[4]:
```
import sys
sys.path.insert(0, './model/')
import PredPreyCode as PP

py_model = Model('Py', function=PP.PredPrey)
```

```
py_model.uncertainties = uncertainties
py_model.constants = constants3
py_model.outcomes = outcomes
```

[5]:
```
with MultiprocessingEvaluator(py_model, n_processes=processes) as evaluator:
    results_py, outcome_py = evaluator.perform_experiments(scenarios=50,␣
 ↪uncertainty_sampling=US)
```

[6]:
```
outcome_py =  fix_format(outcome_py)
```

[7]:
```
# Now making the figure works.
fig, axes = lines(results_py, outcome_py)
```



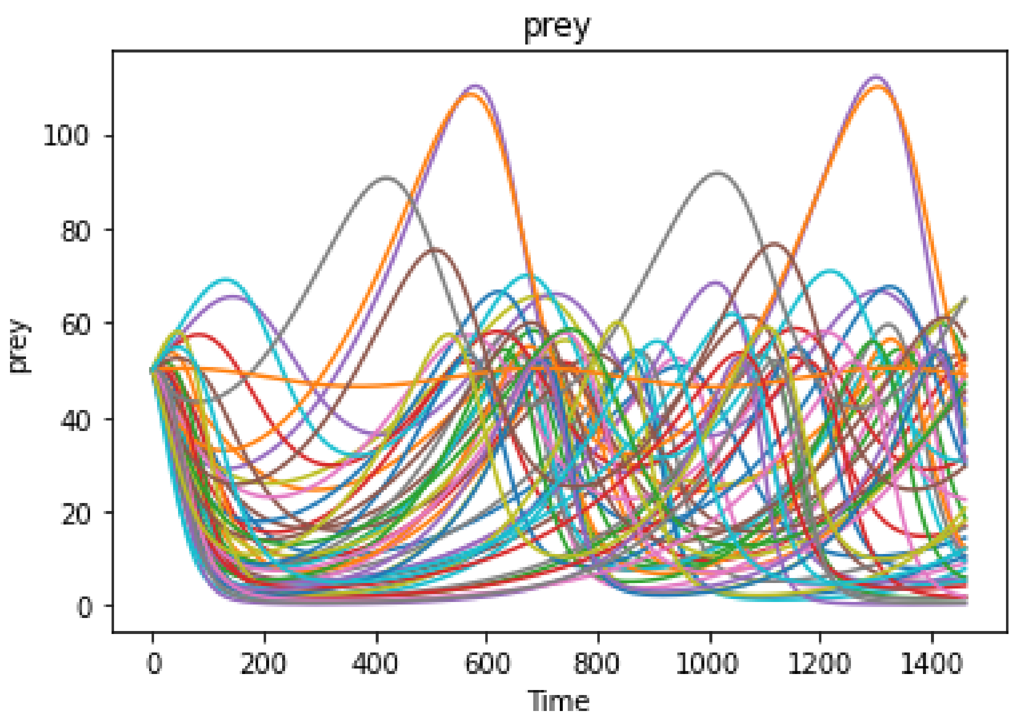
1.3 Vensim

[8]:
```
vensim_model = PysdModel(name= "Vensim", mdl_file=r'./model/PredPrey.mdl')
vensim_model.uncertainties = uncertainties
vensim_model.constants = constants
vensim_model.outcomes = outcomes
```

[9]:
```
with MultiprocessingEvaluator(vensim_model, n_processes=processes) as evaluator:
```

```
    vensim_results, vensim_outcomes = evaluator.
↪perform_experiments(scenarios=50, uncertainty_sampling=US)
```

[10]:
```
figure = lines(vensim_results, vensim_outcomes) #show lines, and end state␣
↪density
plt.show() #show figure
```



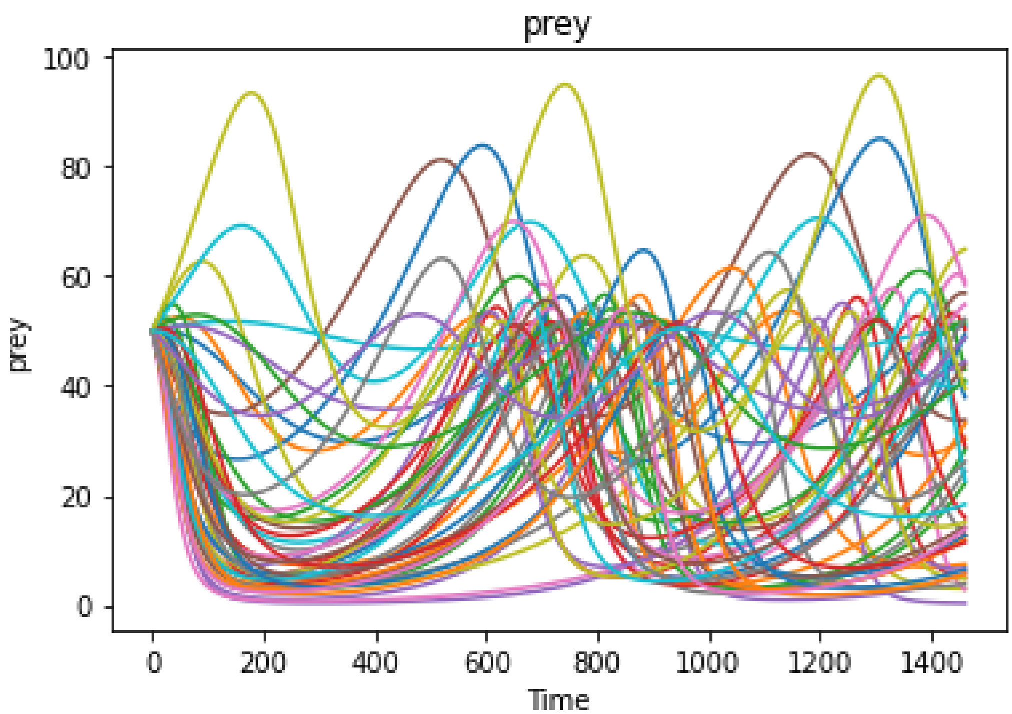
1.4 Excel

[11]:
```
excel_model = ExcelModel(name="Excel", wd='./model/', model_file='../model/
↪PredPrey.xlsx', default_sheet="Sheet1")



excel_model.uncertainties = uncertainties
excel_model.constants = constants2
excel_model.outcomes = outcomes
```

[12]:
```
with SequentialEvaluator(excel_model) as evaluator:
    excel_results, excel_outcomes = evaluator.perform_experiments(scenarios=50,␣
↪reporting_interval=1, uncertainty_sampling=US)
```

[13]:
```
excel_outcomes = fix_format(excel_outcomes)
```

```
[14]: figure = lines(excel_results, excel_outcomes) #show lines, and end state density
plt.show() #show figure
```



1.5 Netlogo

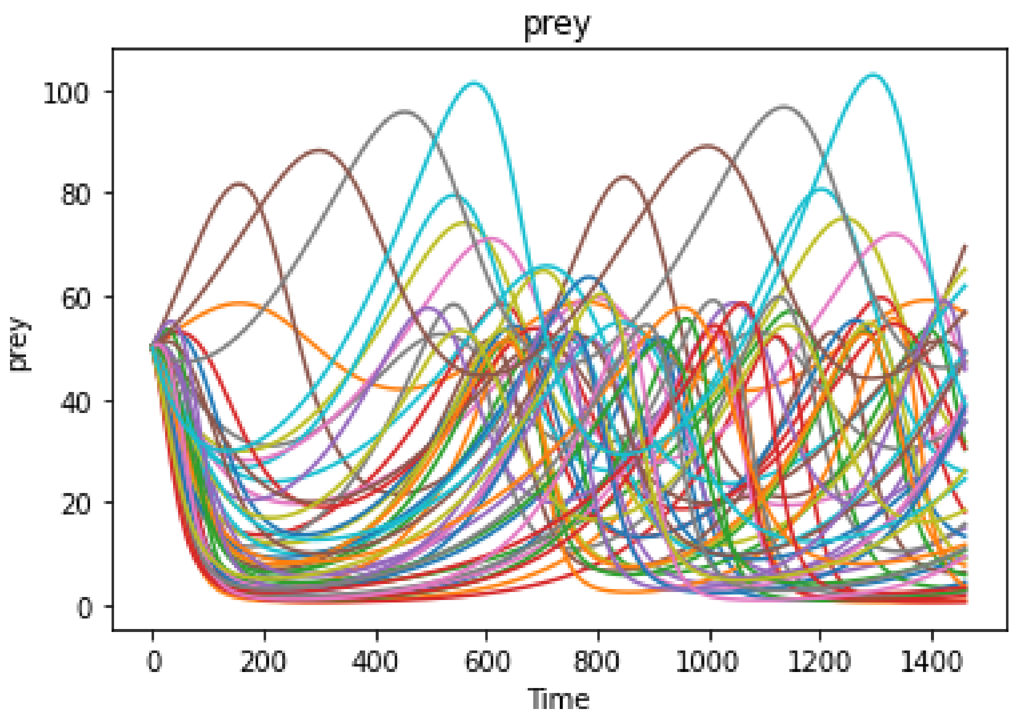
```
[15]: netlogo_model = NetLogoModel(name="Netlogo", wd=r'./model/', model_file=r'../
 ↪model/PredPrey.nlogo')
netlogo_model.run_length = 1460 #final time/dt
netlogo_model.replications = 1

netlogo_model.uncertainties = uncertainties
netlogo_model.constants = constants2
netlogo_model.outcomes = outcomes
```

```
[16]: with MultiprocessingEvaluator(netlogo_model, n_processes=processes) as␣
 ↪evaluator:
    netlogo_results, netlogo_outcomes = evaluator.
 ↪perform_experiments(scenarios=50, uncertainty_sampling=US)
```

```
[17]: netlogo_outcomes = fix_format(netlogo_outcomes)
```

```
[18]: figure = lines(netlogo_results, netlogo_outcomes) #show lines, and end state␣
 ↪density
```

```
plt.show() #show figure
```



1.6 Comparison

[19]:

```
# Make results easier to refer to
python_outcomes = outcome_py['prey']
vensim_outcomes = vensim_outcomes['prey']
excel_outcomes = excel_outcomes['prey']
netlogo_outcomes = netlogo_outcomes['prey']
```

[20]:

```
from sklearn.metrics import mean_absolute_error

vensim_errors = []
excel_errors = []
netlogo_errors = []
for python_outcome in python_outcomes:
    for vensim_outcome in vensim_outcomes:
        vensim_errors.append(mean_absolute_error(python_outcome,␣
 ↪vensim_outcome))
    for excel_outcome in excel_outcomes:
        excel_errors.append(mean_absolute_error(python_outcome, excel_outcome))
    for netlogo_outcome in netlogo_outcomes:
```

```
        netlogo_errors.append(mean_absolute_error(python_outcome,␣
↪netlogo_outcome))

errors = {}
errors['Vensim'] = max(vensim_errors)
errors['Excel'] = max(excel_errors)
errors['Netlogo'] = max(netlogo_errors)

for model, error in errors.items():
    print('Maximum error', model, ':', str(error))
print('------------------------------------------')
print()
print('The model with the highest absolute error is', str(max(errors,␣
↪key=errors.get)))
```

```
Maximum error Vensim : 54.476250035954
Maximum error Excel : 53.28141856121107
Maximum error Netlogo : 58.247759857939315
------------------------------------------

The model with the highest absolute error is Netlogo
```

Analysis - outcomes of interest

The different results show that there is quite a big difference between the different implementations of the same model. Statistically this method is not completly sound, but is gives a good indication of the effects of software used to implement models.

[ ]:

```

```