seed_test

May 8, 2019

```
In [1]: '''
        Created on May 2, 2017

        @author: jhkwakkel
        '''
        from __future__ import division

        import math
        import numpy as np

        from scipy.optimize import brentq

        def lake_problem(
                 b = 0.42,          # decay rate for P in lake (0.42 = irreversible)
                 q = 2.0,           # recycling exponent
                 mean = 0.02,       # mean of natural inflows
                 stdev = 0.0017,     # future utility discount rate
                 delta = 0.98,      # standard deviation of natural inflows
                 alpha = 0.4,       # utility from pollution
                 nsamples = 100,    # Monte Carlo sampling of natural inflows
                 steps=100,
                 **kwargs):
            decisions = [kwargs[str(i)] for i in range(steps)]

            Pcrit = brentq(lambda x: x**q/(1+x**q) - b*x, 0.01, 1.5)
            nvars = len(decisions)
            X = np.zeros((nvars,))
            average_daily_P = np.zeros((nvars,))
            decisions = np.array(decisions)
            reliability = 0.0

            for _ in range(nsamples):
                X[0] = 0.0

                natural_inflows = np.random.lognormal(
                        math.log(mean**2 / math.sqrt(stdev**2 + mean**2)),
                        math.sqrt(math.log(1.0 + stdev**2 / mean**2)),
                        size = nvars)
                for t in range(1,nvars):
                    X[t] = (1-b)*X[t-1] + X[t-1]**q/(1+X[t-1]**q) + decisions[t-1] +\
                            natural_inflows[t-1]
                    average_daily_P[t] += X[t]/float(nsamples)

                reliability += np.sum(X < Pcrit)/float(nsamples*nvars)

            max_P = np.max(average_daily_P)
            utility = np.sum(alpha*decisions*np.power(delta,np.arange(nvars)))
            inertia = np.sum(np.abs(np.diff(decisions)) > 0.02)/float(nvars-1)

            return max_P, utility, inertia, reliability
```

```
In [3]: from ema_workbench import (Model, RealParameter, ScalarOutcome, Constant)

        #instantiate the model
        lake_model = Model('lakeproblem', function=lake_problem)
        lake_model.time_horizon = 100 # used to specify the number of timesteps

        #specify uncertainties
        lake_model.uncertainties = [RealParameter('mean', 0.01, 0.05),
                                    RealParameter('stdev', 0.001, 0.005),
                                    RealParameter('b', 0.1, 0.45),
                                    RealParameter('q', 2.0, 4.5),
                                    RealParameter('delta', 0.93, 0.99)]

        # set levers, one for each time step
        lake_model.levers = [RealParameter(str(i), 0, 0.1) for i in
                             range(lake_model.time_horizon)] # we use time_horizon here

        #specify outcomes
        lake_model.outcomes = [ScalarOutcome('max_P'),
                               ScalarOutcome('utility'),
                               ScalarOutcome('inertia'),
                               ScalarOutcome('reliability')]
```

```
In [10]: from ema_workbench import ema_logging, Policy, perform_experiments

         ema_logging.log_to_stderr(ema_logging.INFO)

         np.random.seed(123456)

         policy = Policy("no release", **{l.name:0 for l in lake_model.levers})
         n_scenarios = 100
         results1 = perform_experiments(lake_model, n_scenarios, policy)
```

```
[MainProcess/INFO] performing 100 scenarios * 1 policies * 1 model(s) = 100 experiments
[MainProcess/INFO] performing experiments sequentially
```

```
[MainProcess/INFO] 10 cases completed
[MainProcess/INFO] 20 cases completed
[MainProcess/INFO] 30 cases completed
[MainProcess/INFO] 40 cases completed
[MainProcess/INFO] 50 cases completed
[MainProcess/INFO] 60 cases completed
[MainProcess/INFO] 70 cases completed
[MainProcess/INFO] 80 cases completed
[MainProcess/INFO] 90 cases completed
[MainProcess/INFO] 100 cases completed
[MainProcess/INFO] experiments finished
```

```
In [11]: np.random.seed(123456)

         policy = Policy("no release", **{l.name:0 for l in lake_model.levers})
         n_scenarios = 100
         results2 = perform_experiments(lake_model, n_scenarios, policy)
```

```
[MainProcess/INFO] performing 100 scenarios * 1 policies * 1 model(s) = 100 experiments
[MainProcess/INFO] performing experiments sequentially
[MainProcess/INFO] 10 cases completed
[MainProcess/INFO] 20 cases completed
[MainProcess/INFO] 30 cases completed
[MainProcess/INFO] 40 cases completed
[MainProcess/INFO] 50 cases completed
[MainProcess/INFO] 60 cases completed
[MainProcess/INFO] 70 cases completed
[MainProcess/INFO] 80 cases completed
[MainProcess/INFO] 90 cases completed
[MainProcess/INFO] 100 cases completed
[MainProcess/INFO] experiments finished
```

```
In [12]: results1[0].head()
```

```
Out[12]:          b     delta      mean         q     stdev    0    1    2    3    4  \
         0  0.179137  0.984665  0.035851  4.349994  0.001089  0.0  0.0  0.0  0.0  0.0
         1  0.140257  0.970474  0.041461  2.549141  0.001369  0.0  0.0  0.0  0.0  0.0
         2  0.309331  0.932648  0.037357  4.413641  0.001235  0.0  0.0  0.0  0.0  0.0
         3  0.318668  0.974674  0.044841  4.469080  0.004531  0.0  0.0  0.0  0.0  0.0
         4  0.218840  0.944542  0.024423  2.715838  0.001919  0.0  0.0  0.0  0.0  0.0

            ...   93   94   95   96   97   98   99  scenario      policy        model
         0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1200  no release  lakeproblem
         1  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1201  no release  lakeproblem
         2  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1202  no release  lakeproblem
         3  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1203  no release  lakeproblem
         4  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1204  no release  lakeproblem
         [5 rows x 108 columns]

In [13]: results2[0].head()

Out[13]:           b     delta      mean         q     stdev    0    1    2    3    4  \
         0  0.179137  0.984665  0.035851  4.349994  0.001089  0.0  0.0  0.0  0.0  0.0
         1  0.140257  0.970474  0.041461  2.549141  0.001369  0.0  0.0  0.0  0.0  0.0
         2  0.309331  0.932648  0.037357  4.413641  0.001235  0.0  0.0  0.0  0.0  0.0
         3  0.318668  0.974674  0.044841  4.469080  0.004531  0.0  0.0  0.0  0.0  0.0
         4  0.218840  0.944542  0.024423  2.715838  0.001919  0.0  0.0  0.0  0.0  0.0

            ...   93   94   95   96   97   98   99  scenario      policy        model
         0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1300  no release  lakeproblem
         1  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1301  no release  lakeproblem
         2  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1302  no release  lakeproblem
         3  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1303  no release  lakeproblem
         4  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0      1304  no release  lakeproblem

         [5 rows x 108 columns]

In [14]: results1[1].keys()

Out[14]: dict_keys(['max_P', 'utility', 'inertia', 'reliability'])

In [15]: results1[1]['max_P'] - results2[1]['max_P']

Out[15]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

In [ ]:
```