03_simulation_optimisation

October 23, 2023

1 Simulation Optimsiation using DES model of homeless response system

[1]:
```
import datetime
print('Current version of this notebook updated ' + str(datetime.date.today()))
```

```
Current version of this notebook updated 2023-10-23
```

1.1 Ranking & Selection

First we employ a Ranking & Selection algorithm based on the Kim & Nelson procedure. Details of this procedure can be found in section 9.3.2 (page 247) of 'Foundation and Methods of Stochastic Simulation' - Edition 2 (2021). The Python code for this procedure is found in the ranking_and_selection.py file in the GitHub repository. Below we import this module, and some others

[2]:
```
# modules from this repository
import ranking_and_selection as rs
import simulation_model as sim

# external packages
import numpy as np
import random
```

1.1.1 Testing using Inventory System

In order to test our KN procedure, we test it on a simulation model which has already been analysed using the KN algorithm in STOR-606 module - this is an (s, S) inventory system where stock is replenished to a level of S when it reaches s.

[3]:
```
solutions = [i for i in range(1600)]
k=np.array([i for i in range(1600)])

def simulate(solution):
    # one replication of simulating the cost of the inventory policy
    out=rs.InventorySystem(solution)[0]
    return out
```

[4]:
```
random.seed(1)
opt_sols = []
for i in range(10):
    spc = rs.SolutionSpace(solutions)
    spc.optimise_rs(0.05, 50, 1, simulate, False)
    s,S = rs.get_sS_system(k[spc.active][0])
    opt_sols.append((s,S))
```

[5]:
```
print('(s,S) for the optimal solution found at each iteration of the algorithm')
print(opt_sols)
```

```
(s,S) for the optimal solution found at each iteration of the algorithm
[(19, 51), (18, 54), (16, 51), (21, 56), (16, 54), (21, 53), (13, 50), (16, 51),
(17, 49), (20, 50)]
```

The above illustrates that this KN algorithm can return different solutions when it is run at different times (i.e. with different starting seeds) - this is likely due to the difference between the true best and other good solutions being less than the 'delta' indiffference zone parameter used when running the algorithm above.

1.1.2 Developing a discrete solution space for the homeless response system

[3]:
```
build_rate_options = {'housing' : [25, 50], 'shelter' : [25,50]}
annual_budget = 75
accommodation_budgets = {'housing' : 200, 'shelter' : 200}
simulation_length = 5

sols = rs.generate_solution_space(build_rate_options, annual_budget,␣
 ↪accommodation_budgets, simulation_length)
```

Below we print ten of the 221 feasible solutions

[4]:
```
sols[0:9]
```

[4]:
```
[{'housing': [25, 25, 25, 25, 25], 'shelter': [25, 25, 25, 25, 25]},
 {'housing': [25, 25, 25, 25, 25], 'shelter': [50, 25, 25, 25, 25]},
 {'housing': [50, 25, 25, 25, 25], 'shelter': [25, 25, 25, 25, 25]},
 {'housing': [25, 25, 25, 25, 25], 'shelter': [25, 50, 25, 25, 25]},
 {'housing': [25, 25, 25, 25, 25], 'shelter': [50, 50, 25, 25, 25]},
 {'housing': [50, 25, 25, 25, 25], 'shelter': [25, 50, 25, 25, 25]},
 {'housing': [25, 50, 25, 25, 25], 'shelter': [25, 25, 25, 25, 25]},
 {'housing': [25, 50, 25, 25, 25], 'shelter': [50, 25, 25, 25, 25]},
 {'housing': [50, 50, 25, 25, 25], 'shelter': [25, 25, 25, 25, 25]}]
```

Below we initialise a solution space object with the solutions we have generated

[5]:
```
spc = rs.SolutionSpace(sols)
```

We next set a seed and then look for an optimal solution using the KN algorithm. A line of text is printed below whenever solutions are removed from the candidate list by the algorithm.

[6]:
```
random.seed(1)
spc.optimise_rs(0.05, 10, 2, sim.simulate_as_is, False)
```

The details of the optimal solution are given below, followed by the following 20 solutions in decreasing order of the iteration number at which the KN algorithm removed them from the candidate list.

[7]:
```
np.array(spc.solutions)[spc.active][0].solution
```

[7]:
```
{'housing': [50, 50, 50, 25, 25], 'shelter': [25, 25, 25, 25, 25]}
```

[8]:
```
sort_index = np.flip(np.argsort(np.array(spc.eliminate)))
n = 0 # count of solutions printed
for i in sort_index:
    print(str(spc.solutions[i].solution) + ' eliminated after ' + str(spc.
 ↪eliminate[i]))
    n+=1
    if n>20:
        break
```

```
{'housing': [25, 50, 25, 25, 50], 'shelter': [25, 25, 50, 50, 25]} eliminated
after 333
{'housing': [50, 50, 25, 25, 25], 'shelter': [25, 25, 25, 50, 50]} eliminated
after 174
{'housing': [50, 25, 25, 25, 50], 'shelter': [25, 25, 25, 25, 25]} eliminated
after 174
{'housing': [25, 25, 50, 25, 50], 'shelter': [25, 25, 25, 50, 25]} eliminated
after 173
{'housing': [50, 25, 25, 25, 50], 'shelter': [25, 25, 50, 50, 25]} eliminated
after 172
{'housing': [50, 25, 25, 25, 25], 'shelter': [25, 25, 50, 25, 25]} eliminated
after 131
{'housing': [50, 50, 25, 25, 25], 'shelter': [25, 25, 25, 50, 25]} eliminated
after 115
{'housing': [50, 25, 50, 25, 25], 'shelter': [25, 25, 25, 50, 25]} eliminated
after 109
{'housing': [50, 25, 50, 25, 25], 'shelter': [25, 25, 25, 25, 25]} eliminated
after 101
{'housing': [50, 25, 25, 50, 50], 'shelter': [25, 50, 25, 25, 25]} eliminated
after 98
{'housing': [25, 25, 50, 50, 25], 'shelter': [25, 25, 25, 25, 25]} eliminated
after 92
{'housing': [50, 25, 50, 25, 50], 'shelter': [25, 50, 25, 50, 25]} eliminated
after 90
{'housing': [50, 25, 25, 25, 25], 'shelter': [25, 25, 50, 50, 50]} eliminated
```

```
after 88
{'housing': [50, 25, 25, 50, 25], 'shelter': [25, 50, 25, 25, 25]} eliminated
after 86
{'housing': [50, 50, 25, 50, 25], 'shelter': [25, 25, 25, 25, 50]} eliminated
after 84
{'housing': [50, 25, 25, 50, 25], 'shelter': [25, 25, 25, 25, 25]} eliminated
after 84
{'housing': [50, 25, 25, 25, 50], 'shelter': [25, 25, 50, 25, 25]} eliminated
after 80
{'housing': [50, 50, 25, 25, 50], 'shelter': [25, 25, 25, 25, 25]} eliminated
after 80
{'housing': [25, 50, 25, 25, 25], 'shelter': [25, 25, 50, 25, 25]} eliminated
after 78
{'housing': [50, 50, 50, 25, 25], 'shelter': [25, 25, 25, 50, 25]} eliminated
after 77
{'housing': [25, 50, 50, 50, 25], 'shelter': [50, 25, 25, 25, 50]} eliminated
after 77
```

Next steps:

- To identify good solutions using the analytical model
- To explore (potentially using data from the simulated solutions) why certain solutions are performing better than others.