# 0087_bed_occupancy_object_based

July 18, 2018

In [3]:
```
%matplotlib inline
```

```
"""
This model requires simpy to run (pip install simpy). All other modules
should are present in an Anaconda installation of Python 3. The model was
written in Python 3.6.

This model simulates the random arrival of patients at a hospital. Patients
stay for a random time (given an average length of stay). Both inter-arrival
time and length of stay are sampled from an inverse exponential distribution.
Average inter-arrival time and length of stay may be changed by changing their
values in class g (global variables). At intervals the number of beds occupied
are counted in an audit, and a chart of bed occupancy is plotted at the end of
the run.

There are four classes:

g (global variables)
--------------------

This class stores global variables. No individual object instance is used;
global variables are stored as class variables.

Hospital
--------
Hospital class (one instance created):
    1) Dictionary of patients present
    2) List of audit times
    3) List of beds occupied at each audit time
    4) Current total beds occupied
    5) Admissions to data

The Hospital class contains methods for audit of beds occupied, summarising
audit (at end of run), and plotting bed occupancy over time (at end of run).

Model
-----
```

```
The model class contains the model environment. The modelling environment is
set up, and patient arrival and audit processes initiated. Patient arrival
triggers a spell for that patient in hospital. Arrivals and audit continue
fort he duration of the model run. The audit is then summarised and bed
occupancy (with 5th, 50th and 95th percentiles) plotted.

Patient
-------
The patient class is the template for all patients generated (each new patient
arrival creates a new patietn object). The patient object contains patient id
and length of stay.

Main
----
Code entry point after: if __name__ == '__main__'
Creates model object, and runs model

"""

# Import required modules

import simpy
import random
import pandas as pd
import matplotlib.pyplot as plt


class g:
    """g holds Global variables. No individual instance is required"""

    inter_arrival_time = 1  # Average time (days) between arrivals
    los = 10  # Average length of stay in hospital (days)
    sim_duration = 500  # Duration of simulation (days)
    audit_interval = 1  # Interval between audits (days)


class Hospital:
    """
    Hospital class holds:
    1) Dictionary of patients present
    2) List of audit times
    3) List of beds occupied at each audit time
    4) Current total beds occupied
    5) Admissions to data

    Methods:

    __init__: Set up hospital instance
```

```
    audit: records number of beds occupied

    build_audit_report: builds audit report at end of run (calculate 5th, 50th
    and 95th percentile bed occupancy.

    chart: plot beds occupied over time (at end of run)
    """

    def __init__(self):
        """
        Constructor method for hospital class"
        Initialise object with attributes.
        """

        self.patients = {}  # Dictionary of patients present
        self.audit_time = []  # List of audit times
        self.audit_beds = []  # List of beds occupied at each audit time
        self.bed_count = 0  # Current total beds occupied
        self.admissions = 0  # Admissions to data
        return

    def audit(self, time):
        """
        Audit method. When called appends current simulation time to audit_time
        list, and appends current bed count to audit_beds.
        """

        self.audit_time.append(time)
        self.audit_beds.append(self.bed_count)
        return

    def build_audit_report(self):
        """
        This method is called at end of run. It creates a pandas DataFrame,
        transfers audit times and bed counts to the DataFrame, and
        calculates/stores 5th, 50th and 95th percentiles.
        """
        self.audit_report = pd.DataFrame()
        self.audit_report['Time'] = self.audit_time
        self.audit_report['Occupied_beds'] = self.audit_beds
        self.audit_report['Median_beds'] = \
            self.audit_report['Occupied_beds'].quantile(0.5)
        self.audit_report['Beds_5_percent'] = \
            self.audit_report['Occupied_beds'].quantile(0.05)
        self.audit_report['Beds_95_percent'] = \
            self.audit_report['Occupied_beds'].quantile(0.95)
        return
```

```
    def chart(self):
        """
        This method is called at end of run. It plots beds occupancy over the
        model run, with 5%, 50% and 95% percentiles.
        """

        plt.plot(self.audit_report['Time'],
                 self.audit_report['Occupied_beds'],
                 color='k',
                 marker='o',
                 linestyle='solid',
                 markevery=1,
                 label='Occupied beds')
        plt.plot(self.audit_report['Time'],
                 self.audit_report['Beds_5_percent'],
                 color='0.5',
                 linestyle='dashdot',
                 markevery=1,
                 label='5th percentile')
        plt.plot(self.audit_report['Time'],
                 self.audit_report['Median_beds'],
                 color='0.5',
                 linestyle='dashed',
                 label='Median')
        plt.plot(self.audit_report['Time'],
                 self.audit_report['Beds_95_percent'],
                 color='0.5',
                 linestyle='dashdot',
                 label='95th percentile')
        plt.xlabel('Day')
        plt.ylabel('Occupied beds')
        plt.title(
            'Occupied beds (individual days with 5th, 50th and 95th ' + \
            'percentiles)')
        # plt.legend()
        plt.show()
        return


class Model:
    """
    The main model class.

    The model class contains the model environment. The modelling environment is
    set up, and patient arrival and audit processes initiated. Patient arrival
    triggers a spell for that patient in hospital. Arrivals and audit continue
    for the duration of the model run. The audit is then summarised and bed
```

```
    occupancy (with 5th, 50th and 95th percentiles) plotted.

    Methods are:

    __init__: Set up model instance

    audit_beds: call for bed audit at regular intervals (after initial delay for
    model warm-up)

    new_admission: trigger new admissions to hospital at regular intervals.
    Call for patient generation with patient id and length of stay, then call
    for patient spell in hospital.

    run: Controls the main model run. Initialises model and patient arrival and
    audit processes. Instigates the run. At end of run calls for an audit
    summary and bed occupancy plot.

    spell: increments beds occupied, stores patient in hospital patient list
    dictionary, waits for patient length of stay and then decrements beds
    occupied and removes patient from hospital patient list.
    """

    def __init__(self):

        """
        Constructor class for new model.
        """
        self.env = simpy.Environment()

        return

    def audit_beds(self, delay):
        """
        Bed audit process. Begins by applying delay, then calls for audit at
        intervals set in g.audit_interval

        :param delay: delay (days) at start of model run for model warm-up.
        """

        # Delay first audit
        yield self.env.timeout(delay)

        # Continually generate audit requests until end of model run
        while True:
            # Call audit
            self.hospital.audit(self.env.now)
            # Delay until next call
            yield self.env.timeout(g.audit_interval)
```

```
        return

    def new_admission(self, interarrival_time, los):
        """
        New admissions to hospital.

        :param interarrival_time: average time (days) between arrivals
        :param los: average length of stay (days)
        """
        while True:
            # Increment hospital admissions count
            self.hospital.admissions += 1

            # Generate new patient object (from Patient class). Give patient id
            # and set length of stay from inverse exponential distribution).
            p = Patient(patient_id=self.hospital.admissions,
                        los=random.expovariate(1 / los))

            # Add patient to hospital patient dictionary
            self.hospital.patients[p.id] = p

            # Generate a patient spell in hospital (by calling spell method).
            # This triggers a patient admission and allows the next arrival to
            # be set before the paitent spell is finished
            self.spell = self.spell_gen(p)
            self.env.process(self.spell)

            # Set and call delay before looping back to new patient admission
            next_admission = random.expovariate(1 / interarrival_time)
            yield self.env.timeout(next_admission)

        return

    def run(self):
        """
        Controls the main model run. Initialises model and patient arrival and
        audit processes. Instigates the run. At end of run calls for an audit
        summary and bed occupancy plot
        """

        # Set up hospital (calling Hospital class)
        self.hospital = Hospital()

        # Set up starting processes: new admissions and bed  audit (with delay)
        self.env.process(self.new_admission(g.inter_arrival_time, g.los))
        self.env.process(self.audit_beds(delay=20))
```

```
        # Start model run
        self.env.run(until=g.sim_duration)

        # At end of run call for bed audit summary and bed occupancy plot
        self.hospital.build_audit_report()
        self.hospital.chart()

        return

    def spell_gen(self, p):
        """
        Patient hospital stay generator. Increment bed count, wait for patient
        length of stay to complete, then decrement bed count and remove patient
        from hospital patient dictionary

        :param p: patient object (contains length of stay for patient)
        """

        # Increment bed count
        self.hospital.bed_count += 1

        # Wait for patient length of stay to complete
        yield self.env.timeout(p.los)

        # Decrement bed count and remove patient from hospital patient
        # dictionary
        self.hospital.bed_count -= 1
        del self.hospital.patients[p.id]
        return


class Patient:
    """
    Patient class. Contains patient id and length of stay (it could contain
    other info about patient, such as priority or clinical group.

    The only method is __init__ for creating a patient (with assignment of
    patient id and length of stay).
    """

    def __init__(self, patient_id, los):
        """
        Contructor for new patient.

        :param patient_id: id of patient  (set in self.new_admission)
        :param los: length of stay (days, set in self.new_admission)
        """
        self.id = patient_id
```

```
        self.los = los

        return


def main():
    """
    Code entry point after: if __name__ == '__main__'
    Creates model object, and runs model
    """

    model = Model()
    model.run()

    return


# Code entry point. Calls main method.
if __name__ == '__main__':
    main()
```