Singapore Management University
School of Computing and Information Systems
2021/2022 Semester 2
CS606: AI Planning and Decision Making
Assignment 1 (due date: 5 Feb 2021, 12pm)

Answer BOTH Questions. Maximum marks = 30.

Submission Instructions:

You are expected to submit the following files:
(1) a Word or PDF document containing your answers to the written response to the questions (i.e. 1a, 1c and 2a);
(2) Python codes (named `A1Q1b.py`, `A1Q1d.py` and `A1Q2b.py`, and please change `Your_Name` to your real name in code skeleton files).

These files should be zipped and uploaded to the eLearn Assignment folder **Assignment 1**. Multiple submissions are permitted up to the due time, but only the last submission will be saved and graded. Pls note that the Originality Check has been enabled.

How to run python file with data file in terminal?

```
python name_of_python_file.py name_of_data_file.pk
```

Question 1 (Mathematical Programming) [20 marks]

ABC Airline operates flights between a set of cities in the USA. Management wants to build a hub-and-spoke flight network such that every city is within 1000 km from at least one of the hubs. In class, we developed an integer programming model to determine the minimum *number* of hubs needed to cover all cities. In this question, we explore this problem further.

Notationally, let c_i represent the daily cost of operating a hub in city i, a_{ij} represent whether city *i* and city *j* are within 1000km, and let the decision variables x_i indicate whether city *i* is selected as a hub.

(a) [5 marks] Construct an **IP model** to select the hubs that will minimize the *total daily operational cost*, stating clearly the objective function and constraints.

Answer:
The objective of the following IP model is to identify the optimal hubs that will minimize daily operational coast, as well as respecting the aforementioned constraints.

Hereby, we consider:

Cities: set of *n Cities*
i: A city *i* within Cities
j: Respective city *j* within 1000km of city *i*
C_i: The daily cost of operating a hub in city *i*

Decision Variables:

$$x_j = \begin{cases} 1 & \text{if a hub is located in city } j \\ 0 & \text{if no hub is located in city } j \end{cases}$$

Objective Function:

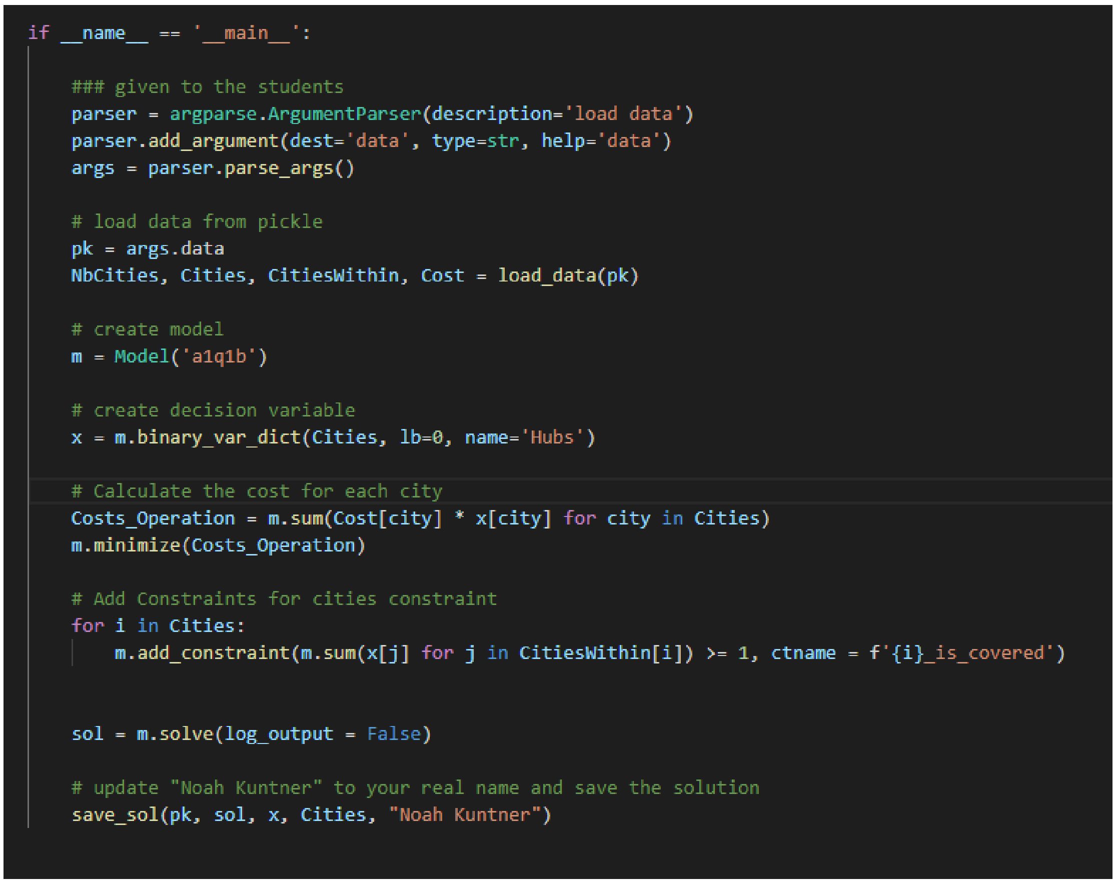
$$\text{Total Cost } = \sum_i (c_i x_i)$$

Constraints:
For each city *j* there must exist a hub within a nearby city *i*

$$\sum_i x_{i,j} \geq 1$$

(b) [5 marks] Implement your IP model using CPLEX (using `A1Q1b.py` as the filename) (Note: `A1Q1b-small.pk` and `A1Q1b-big.pk` are provided as test instances for you, along with the solutions `A1Q1b-small.txt` and `A1Q1b-big.txt;` but grading will involve running your model on a different test instance)

Answer:



```
if __name__ == '__main__':

    ### given to the students
    parser = argparse.ArgumentParser(description='load data')
    parser.add_argument(dest='data', type=str, help='data')
    args = parser.parse_args()

    # load data from pickle
    pk = args.data
    NbCities, Cities, CitiesWithin, Cost = load_data(pk)

    # create model
    m = Model('a1q1b')

    # create decision variable
    x = m.binary_var_dict(Cities, lb=0, name='Hubs')

    # Calculate the cost for each city
    Costs_Operation = m.sum(Cost[city] * x[city] for city in Cities)
    m.minimize(Costs_Operation)

    # Add Constraints for cities constraint
    for i in Cities:
        m.add_constraint(m.sum(x[j] for j in CitiesWithin[i]) >= 1, ctname = f'{i}_is_covered')


    sol = m.solve(log_output = False)

    # update "Noah Kuntner" to your real name and save the solution
    save_sol(pk, sol, x, Cities, "Noah Kuntner")
```

Due to COVID-19, the airline industry is taking a deep plunge and ABC management wants to impose cost-cutting measures. For simplicity, assume flights are to be operated daily from a hub to all other hubs as well as all cities within 1000 km from it, and all flights are two-way. Suppose we know the average daily traffic demands in terms of number of passengers traveling from city *i* to city *j*, which is denoted as d_{ij}. You are tasked to redesign the hub-and-spoke network.

(c) [5 marks] Given a total daily operating budget *B*, construct a revised IP model that now determines which hubs should be selected so as to maximize the total traffic demands that can be fulfilled. For simplicity, we assume that a passenger demand is fulfilled when there is <u>at most</u> one hop via a hub from the origin to the destination city.

Answer:

Cities: set of *n Cities*
i: A city *i* within Cities
Cities_i: A city *i* in a given set
$\text{CitiesWithin}_i \subset$ Cities: Subset of cities within the 1000km of *i*
$d_{i,j}$: Traffic demand between cities *i* and *j*
c_i: The daily cost of operating a hub in city *i*
$\text{Paths}_{i,j}$: The possible paths connecting cities *i* and *j*
b: Daily total operating budget

Decision Variables:

$$x_j = \begin{cases} 1 & \text{if a hub is located in city } j \\ 0 & \text{if there is no hub} \end{cases}$$

$$\text{Hub\_Dict}_{i,j} = \begin{cases} 1 & \text{if } i,j \text{ are both hubs} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{DirectPath}_{i,j} = \begin{cases} 1 & \text{if a direct path exists s.t. } i \in \text{CitiesWithin}_j \text{ and/or they are hubs} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Final\_Dict}_{i,j} = \begin{cases} 1 & \text{if } i,j \text{ are directly connected (either City-Hub-City, Hub-Hub-City and City-Hub-Hub)} \\ 0 & \text{otherwise} \end{cases}$$

Objective Function:

$$\sum_{i\in \text{Cities}} \sum_{j\in \text{Cities}} d_{i,j} \cdot \text{Final\_Dict}_{i,j}$$

Constraints:

$$\sum_{i\in \text{Cities}} c_i x_i \leq b$$

$$\text{Hub\_Dict}_{i,j} \leq x_i \qquad \forall i,j \in \text{Paths}$$

$$\text{Hub\_Dict}_{i,j} \leq x_j \qquad \forall i,j \in \text{Paths}$$

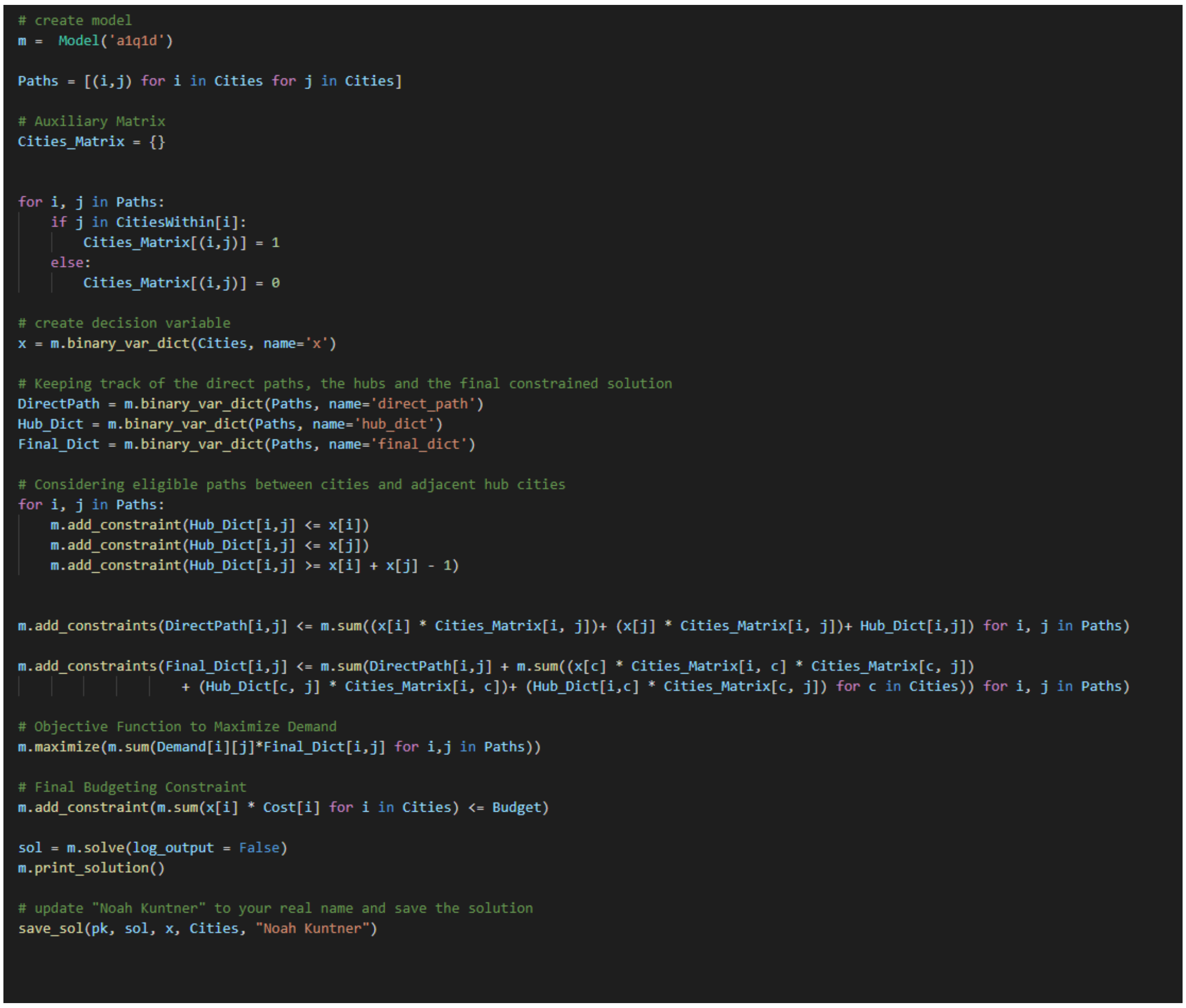
$$\text{Hub\_Dict}_{i,j} \geq x_i + x_j - 1 \qquad \forall i,j \in \text{Paths}$$

$$\text{DirectPath}_{i,j} \leq \sum_{i\in \text{Paths} j\in \text{Paths}} \left(x_i \text{ CitiesWithin}_{i,j} + x_j \text{ CitiesWithin}_{i,j} + \text{Hub\_Dict}_{i,j}\right)$$

$$\text{Final\_Dict}_{i,j} \leq \sum_{i\in \text{Paths} j\in \text{Paths}} \Big[\text{DirectPath}_{i,j} + \sum_{c\,\in \text{Cities}} \big(x_c \text{ CitiesWithin}_{i,c} \text{ Cities Within}_{c,j} + \text{DirectPath}_{c,j} \text{ CitiesWithin}_{i,c} + \text{DirectPath}_{i,c} \text{ CitiesWithin}_{c,j}\big)\Big]$$

(d) [5 marks] Implement your IP model using **CPLEX** (using `A1Q1d.py` as the filename) (Note: Like 1(b), two test instances `A1Q1d-small.pk` and `A1Q1d-big.pk` and corresponding solutions `A1Q1d-small.txt` and `A1Q1d-big.txt` are provided; but grading will involve running your model on a different test instance)

Answer:



```python
# create model
m =  Model('a1q1d')

Paths = [(i,j) for i in Cities for j in Cities]

# Auxiliary Matrix
Cities_Matrix = {}


for i, j in Paths:
    if j in CitiesWithin[i]:
        Cities_Matrix[(i,j)] = 1
    else:
        Cities_Matrix[(i,j)] = 0

# create decision variable
x = m.binary_var_dict(Cities, name='x')

# Keeping track of the direct paths, the hubs and the final constrained solution
DirectPath = m.binary_var_dict(Paths, name='direct_path')
Hub_Dict = m.binary_var_dict(Paths, name='hub_dict')
Final_Dict = m.binary_var_dict(Paths, name='final_dict')

# Considering eligible paths between cities and adjacent hub cities
for i, j in Paths:
    m.add_constraint(Hub_Dict[i,j] <= x[i])
    m.add_constraint(Hub_Dict[i,j] <= x[j])
    m.add_constraint(Hub_Dict[i,j] >= x[i] + x[j] - 1)


m.add_constraints(DirectPath[i,j] <= m.sum((x[i] * Cities_Matrix[i, j])+ (x[j] * Cities_Matrix[i, j])+ Hub_Dict[i,j]) for i, j in Paths)

m.add_constraints(Final_Dict[i,j] <= m.sum(DirectPath[i,j] + m.sum((x[c] * Cities_Matrix[i, c] * Cities_Matrix[c, j])
                    + (Hub_Dict[c, j] * Cities_Matrix[i, c])+ (Hub_Dict[i,c] * Cities_Matrix[c, j]) for c in Cities)) for i, j in Paths)

# Objective Function to Maximize Demand
m.maximize(m.sum(Demand[i][j]*Final_Dict[i,j] for i,j in Paths))

# Final Budgeting Constraint
m.add_constraint(m.sum(x[i] * Cost[i] for i in Cities) <= Budget)

sol = m.solve(log_output = False)
m.print_solution()

# update "Noah Kuntner" to your real name and save the solution
save_sol(pk, sol, x, Cities, "Noah Kuntner")
```

Question 2 (Constraint Programming) [10 marks]

In the **SMU Professor Timetabling problem**, we are given a list of courses to be offered, a list of possible time slots, a list of rooms, and a list of professors to teach those courses. Each course needs to be assigned to one professor, in a room, at a timeslot.

For simplicity, assume that every course meets once a week at one of the 15 time slots (that is, Mon to Fri 8:30-11:45am, 12:00-3:15pm, and 3:30-6:45pm), and all rooms are identical.

The goal is to generate a weekly timetable showing which course is held at which timeslot and room, and taught by which professor. The timetabling constraints are as follows:

- Each professor has a set of courses that he or she is eligible to teach (e.g. Prof Amy can teach CS601, CS602, CS603, CS605, …) and a teaching load (i.e. the number of courses he/she is allocated to).
- Each professor should teach at most 1 course per day.
- Each professor should be assigned no more than and at least 1 less than the teaching load (e.g. if Amy is allocated to 5 courses, the schedule must assign at most 5 but at least 4 courses to her).
- Each room may be assigned to at most one course at a time.

Notationally, let the decision variables be the timeslot, room and professor assigned to each course:

- T (i.e. T[x] denotes the time slot assigned to course x, for each course x)
- R (i.e. R[x] denotes the room assigned to course x, for each course x)
- P (i.e. P[x] denotes the professor assigned to course x, for each course x)

(a) [6 marks] Construct a **CP model** for this problem that will find a feasible solution if one exists. (You should make use of CP constraints such as logical/conditional constraints, the `count` global constraint discussed in class.)

Answer:

Decision Variables:

P_x = Professor assigned to course *x*
R_x = Room assigned to course x
T_x = Time slot assigned to course x

Domains:

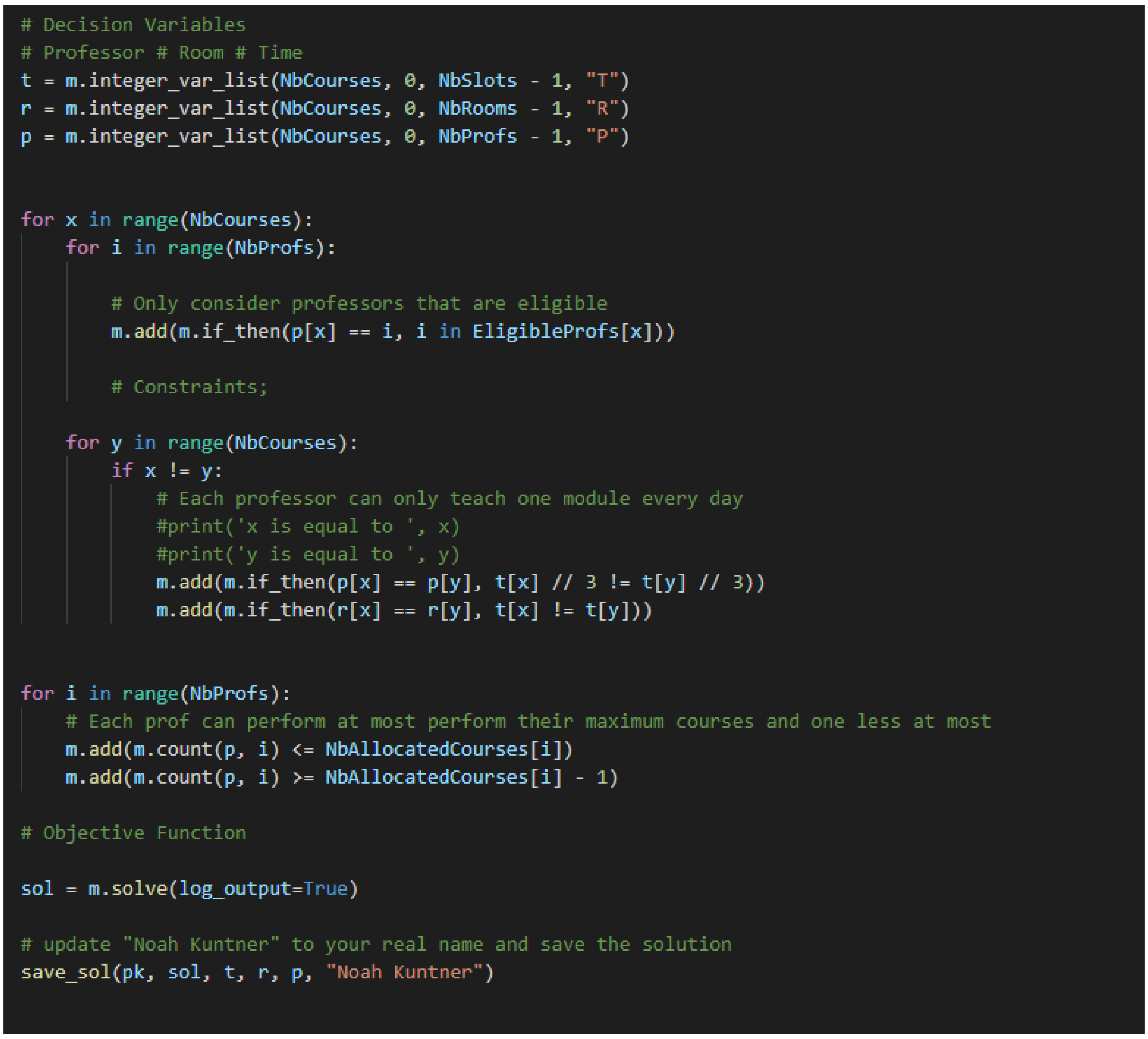
$$x \in C = \{CS601, CS602, \ldots, CS410\} \rightarrow \text{ Set of courses available}$$
$$R_x \in R = \{SR2-1, \ldots, SR2-4\} \rightarrow \text{ Set of rooms}$$
$$p \in P = \{\text{ Amy, …, … Ellie }\} \rightarrow \text{ Set of professors}$$
$$T_x \in T = \{0, \ldots 15\} \rightarrow \text{ There are 15 timeslots available per week}$$
$$E_x \subset P = \text{ Subset of eligible professors for course } x$$
$$P_x \in E_x = \text{ Professors eligible to teach course } x \text{ should be allocated to it}$$
$$n_p \in N = \mathbb{Z}^+ \rightarrow \text{ Each professor is allocated integer values of course loads}$$

Constraints:

$$P_x = p \rightarrow p \in E_x \quad \forall p \in P$$
$$P_x = P_y \rightarrow \text{ Remainder } (T_x, 3) \neq Remainder\left(T_y, 3\right) \forall x, y \in C, \quad x \neq y$$
$$N_p - 1 \leq \text{ Count } (P_x = p) \leq N_p \quad \forall x \in C$$
$$R_x = R_y \rightarrow T_x \neq T_y \forall x, y \in C, \quad x \neq y$$

(b) [4 marks] Implement your CP model using **CP Optimizer** that will generate a feasible schedule for a given dataset, if one exists (using `A1Q2b.py` as the filename).
(Note: `A1Q2b-sample.pk` is provided as a test instance for you, as well as a possible solution `A1Q2b-sample.txt`, but grading will involve running your model on a different test instance)

Answer:



```python
# Decision Variables
# Professor # Room # Time
t = m.integer_var_list(NbCourses, 0, NbSlots - 1, "T")
r = m.integer_var_list(NbCourses, 0, NbRooms - 1, "R")
p = m.integer_var_list(NbCourses, 0, NbProfs - 1, "P")


for x in range(NbCourses):
    for i in range(NbProfs):

        # Only consider professors that are eligible
        m.add(m.if_then(p[x] == i, i in EligibleProfs[x]))

        # Constraints;

    for y in range(NbCourses):
        if x != y:
            # Each professor can only teach one module every day
            #print('x is equal to ', x)
            #print('y is equal to ', y)
            m.add(m.if_then(p[x] == p[y], t[x] // 3 != t[y] // 3))
            m.add(m.if_then(r[x] == r[y], t[x] != t[y]))


for i in range(NbProfs):
    # Each prof can perform at most perform their maximum courses and one less at most
    m.add(m.count(p, i) <= NbAllocatedCourses[i])
    m.add(m.count(p, i) >= NbAllocatedCourses[i] - 1)

# Objective Function

sol = m.solve(log_output=True)

# update "Noah Kuntner" to your real name and save the solution
save_sol(pk, sol, t, r, p, "Noah Kuntner")
```

Marking criteria for both questions:

The written models will be graded according to correctness and completeness.

The implementation codes will be graded according to correctness and efficiency. Where there are multiple correct solutions any solution will be accepted, and full marks will be awarded as long as your model runs in "reasonable" time (check with your classmates how long theirs take!). If your model runs with exceptionally long time or returns a wrong solution, it means it can be improved, and you will get partial credits accordingly.