Lydia Yu

15.076 HW2

Problem 1

- 10 potential sites, build 4 max
- each site has fixed capacity
- each resident needs access to one site
- residential areas $i \in \{1 \dots 50\}$, each w/ r_i residents
- sites $j \in \{1 \dots 10\}$, each w/ capacity C_j
- d_{ij} = distance from i to j $= |x_i - x_j| + |y_i - y_j|$

a) Variables: $S_j = \begin{cases} 1 & \text{if site } j \text{ is open} \\ 0 & \text{ow} \end{cases}$ $\forall j \in [10]$ (indicates which sites to open)

V_{ij} = number of residents from area i served by site j $\quad \forall i \in [50], j \in [10]$

Constraints: build 4 sites max: $\sum_{j=1}^{10} S_j \le 4$

total # residents coming from residential area i must equal # residents in i (every resident assigned to exactly 1 site) $\quad \sum_{j=1}^{10} V_{ij} = r_i \quad \forall i \in [50]$

total # residents served by site j must be within site j's capacity: $\quad \sum_{i=1}^{50} V_{ij} \le C_j \quad \forall j \in [10]$

define distance (not nonlinear because dij not a variable - data is given) $\quad d_{ij} = |x_i - x_j| + |y_i - y_j| \quad \forall i \in [50], j \in [10]$
(this is not an actual constraint, it's a matrix of constants that can be defined when the data is loaded)

if residents from i are served by site j, then j must be built: $\quad V_{ij} \le S_j \cdot C_j \quad \forall i \in [50], j \in [10]$

binary variable: $S_j \in \{0,1\} \quad \forall j \in [10]$

positive integer: $V_{ij} \in \mathbb{Z} \ge 0 \quad \forall i \in [50], j \in [10]$

Objective: minimize total distance residents must travel to vaccination clinics

$$\min_{S,V} \sum_{i=1}^{50} \sum_{j=1}^{10} d_{ij} V_{ij}$$

b) average distance : 1.858
largest distance : 3.731

load in data

```julia
areas = CSV.read("P1_areas.csv", DataFrame, header=["x", "y", "num_res"], skipto=2)
clinics = CSV.read("P1_clinics.csv", DataFrame, header=["x", "y", "capacity"], skipto=2)
```

...

```julia
r = areas.num_res;
C = clinics.capacity;
x_res = areas.x;
y_res = areas.y;
x_clin = clinics.x;
y_clin = clinics.y;
```

```julia
# distance matrix: manhattan distance from res area i to clinic j
d = [abs(x_res[i]-x_clin[j]) + abs(y_res[i]-y_clin[j]) for i=1:nrow(areas), j=1:nrow(clinics)];
```

define the model

```julia
m1 = Model(Gurobi.Optimizer);
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2022-09-06
```

```julia
# variables
# s[j] indicates whether or not site j is opened
@variable(m1, s[1:nrow(clinics)], Bin)  # nrow(clinics) = number of clinics
# v[i,j] represents how many residents from area i go to site j
@variable(m1, v[1:nrow(areas), 1:nrow(clinics)] >= 0, Int)  # nrow(areas) = number of residential areas
```

...

```julia
# constraints
# build 4 sites max
@constraint(m1, sum(s[j] for j=1:nrow(clinics)) <= 4)
# total num residents coming from res area i must be equal to the number of residents
# (equivalent to every resident must be assigned to exactly 1 site)
@constraint(m1, [i=1:nrow(areas)], sum(v[i,j] for j=1:nrow(clinics)) == r[i])
# total num residents served by site j must be within site j's capacity
@constraint(m1, [j=1:nrow(clinics)], sum(v[i,j] for i=1:nrow(areas)) <= C[j])
# if residents from i are served by site j, then j must be built
@constraint(m1, [i=1:nrow(areas), j=1:nrow(clinics)], v[i,j] <= C[j]*s[j])
```

...

```julia
# objective: min total distance residents travel to clinics
@objective(m1, Min, sum(sum(d[i,j]*v[i,j] for j=1:nrow(clinics)) for i=1:nrow(areas)));
```

optimize the model and evaluate solution

```julia
optimize!(m1)
```

...

```julia
# see which clinics got built and min total distance
@show [value(s[j]) for j=1:nrow(clinics)]
@show objective_value(m1)
```

```
[value(s[j]) for j = 1:nrow(clinics)] = [-0.0, 1.0, -0.0, 0.0, 1.0, 1.0, -0.0, 0.0, -0.0, 1.0]
objective_value(m1) = 36763.43319948599
```

```
36763.43319948599
```

```julia
# v_opt is a matrix indicating how many people from each res area go to each site
v_opt = [value(v[i,j]) for i=1:nrow(areas), j=1:nrow(clinics)]

# find which residents i go to which clinics j
# (the indices for values >0 in v_opt)
# this is a vector of i,j pairs indicating res->clin assignments
res_to_clin = findall(i->(i>0), v_opt)
```

...

```julia
# index matrix d by res_to_clin to get a vector of distances that
# each res area must travel to get to its assigned clinic
distances = d[res_to_clin]

# v_opt[res_to_clin] is vector of num people going from each
# res area to each clinic that was built
# the dot prod of distances and v_opt[res_to_clin] is the total distance traveled
total_distance = sum(v_opt[res_to_clin][i]*distances[i] for i=1:length(distances))
# this is also equal to the objective value
```

```
36763.43319948599
```

```julia
# find max distance
print("max distance: ", maximum(distances))
# find average distance
print("\navg distance: ", total_distance/sum(r))
```

```
max distance: 3.7314384460000003
avg distance: 1.8578650292847174
```

plot of locations of residents & chosen clinics:

plot locations of clinics and residents

```
s_opt = [value(s[j]) for j=1:nrow(clinics)]
# find which clinics were built (the indices for values >0 in s_opt)
clinics_built_idx = findall(i->(i>0), s_opt)
```

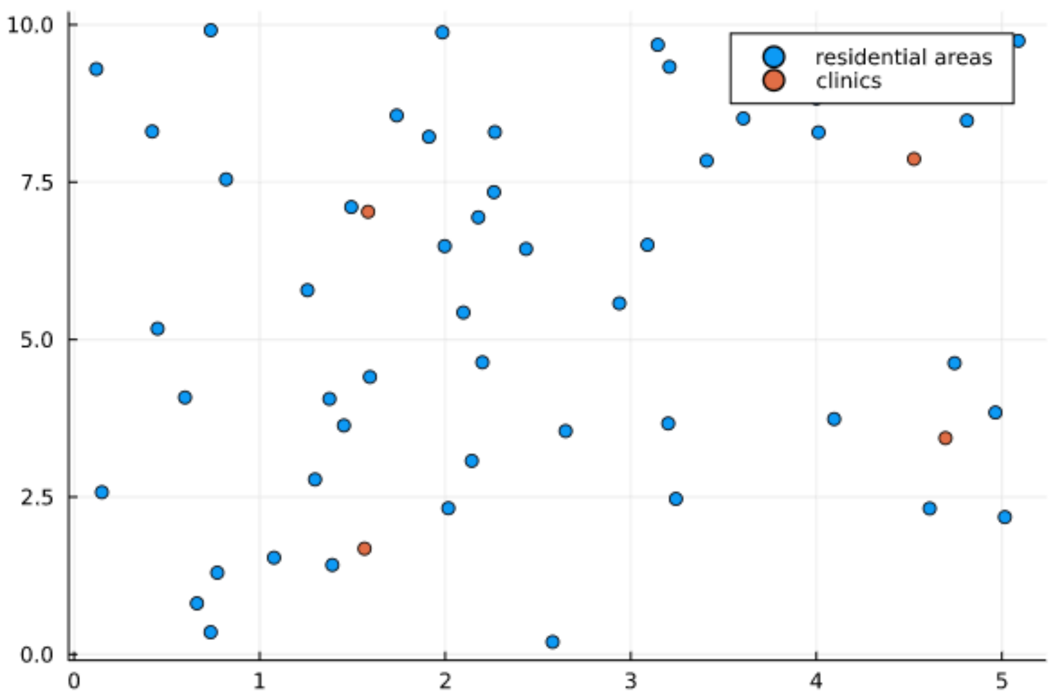
```
4-element Vector{Int64}:
  2
  5
  6
 10
```

```
# subset the clinics df to just data on the ones that were built
clinics_built = clinics[clinics_built_idx, :]
```

4 rows × 3 columns

	x	y	capacity
	Float64	Float64	Int64
1	1.56519	1.67896	7867
2	4.52658	7.86786	13338
3	1.5845	7.02909	8664
4	4.69616	3.43461	12670

```
scatter(areas.x, areas.y, label="residential areas")
scatter!(clinics_built.x, clinics_built.y, label="clinics")
```



c) add constraint $\sum_{j=1}^{10} F_j s_j \le B$ (total cost of opening clinics must be within B)

Problem 2

- $n = 17$ patients, each needs 15 fractions
- can be proton and/or photon : photon $= 15 - p$ ($p = \#$ proton fractions)
- score for each patient $i \in [n]$: $BED_i(p, 15-p)$ given for each $0 \le p \le 15$
- max of C proton fractions (sum of p across all patients $\le C$)

a) Variables: $p_{ij} = \begin{cases} 1 & \text{if patient } i \text{ receives } j-1 \text{ proton fractions} \\ 0 & \text{o/w} \end{cases} \quad \forall i \in [n], j \in [16]$ $\{1, \dots 16\}$

(proton fractions range from 0 to 15. add 1 to get index of matrix)

Constraints: total proton fractions given across all patients can't exceed C:

$$\sum_{i=1}^{17} \sum_{j=1}^{16} \underbrace{p_{ij} \cdot (j-1)}_{\substack{\text{if } p_{ij}=1, \text{ add } (j-1) \text{ proton fractions} \\ \text{to the sum} - (j-1) = \# \text{ proton fractions}}} \le C$$

each patient should be assigned to only one proton fraction amount:

$$\sum_{j=1}^{16} p_{ij} = 1 \quad \forall i \in [17]$$

p_{ij} binary: $p_{ij} \in \{0,1\} \quad \forall i \in [n], j \in \{0 \dots 15\}$

Objective: maximize total BED scores across all patients:

$$\max_{p} \sum_{i=1}^{17} \sum_{j=0}^{15} p_{ij} \cdot BED_i(j, 15-j)$$

b)

load in data

```
# 17x16 df
# rows are patient i and cols are j-1 proton fractions (idx from 1:16)
bed = CSV.read("outcome_data.csv", DataFrame, header=false)
```

...

solve for C = [20, 40, 50]

```
C = [20, 40, 50]
max_scores = []

for x in C
    m2 = Model(Gurobi.Optimizer);

    # variables
    # p[i,j] indicates whether patient i receives j proton fractions
    @variable(m2, p[1:17, 1:16], Bin);

    # constraints
    # total can't exceed max capacity of x proton fractions
    @constraint(m2, sum(sum(p[i,j]*(j-1) for i=1:17) for j=1:16) <= x);
    # each patient is assigned exactly 1 proton fraction amount
    @constraint(m2, [i=1:17], sum(p[i,j] for j=1:16) == 1);

    # objective: maximize total BED score
    # bed[i,j]: BED score for patient i receiving j-1 proton fractions
    @objective(m2, Max, sum(sum(bed[i,j]*p[i,j] for i=1:17) for j=1:16));

    optimize!(m2);

    append!(max_scores, objective_value(m2))

    p_opt = [value(p[i,j]) for i=1:17, j=1:16]
    # get i,j pairs of patients i who received j-1 proton fractions
    p_assigned = findall(i->(i>0), p_opt)
    # df displaying number of proton fractions assigned to each patient
    p_df = DataFrame(Patient=[p_assigned[i][1] for i=1:17],
                     Proton_Fractions=[p_assigned[i][2]-1 for i=1:17])
    sort!(p_df, [:Patient])
    @show(p_df)
end
```

...

```
for i=1:3
    println("Max BED score for C = ", C[i], ": ", max_scores[i])
end
```

```
Max BED score for C = 20: 1437.215
Max BED score for C = 40: 1580.373
Max BED score for C = 50: 1619.601
```

C = 20

Patient Int64	Proton_Fractions Int64
1	2
2	0
3	0
4	0
5	0
6	0
7	1
8	2
9	9
10	0
11	4
12	0
13	0
14	0
15	2
16	0
17	0

C = 40

Patient Int64	Proton_Fractions Int64
1	2
2	0
3	1
4	0
5	0
6	0
7	3
8	5
9	9
10	1
11	5
12	3
13	0
14	0
15	7
16	4
17	0

C = 50

Patient Int64	Proton_Fractions Int64
1	2
2	3
3	1
4	0
5	0
6	0
7	6
8	5
9	9
10	4
11	5
12	4
13	0
14	0
15	7
16	4
17	0