BEE 4750/5750 Homework 2

Sarah Mack (skm237)

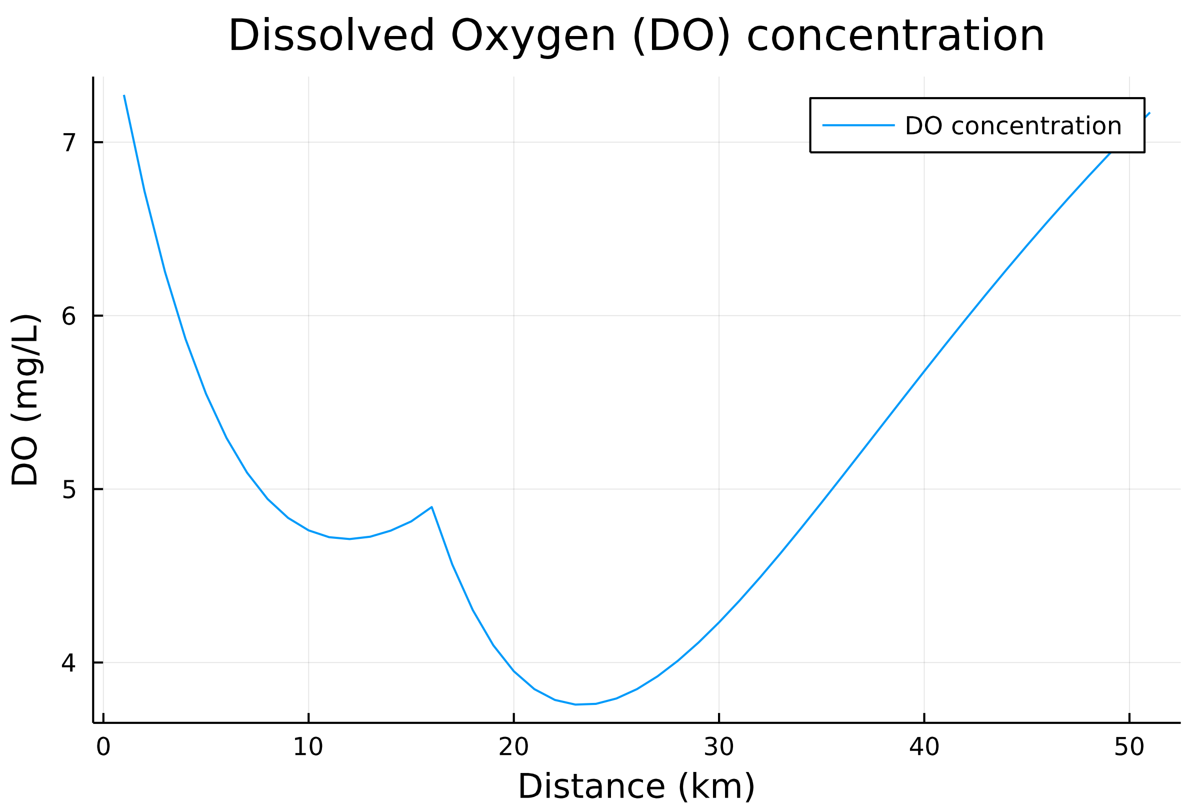
2022-09-27

Problem 1

Problem 1.1

```
julia> function calc_conc(x, U, Cs, Co, Bo, No, ka, kc, kn)
           # U is the stream velocity, Cs is the saturated oxygen saturation, Co
is the initial dissolved DO conc,
           # Bo is the inital CBOD conc, No is the initial NBOD conc, ka is the
reaeration rate, kc is the CBOD
           # CBOD decay rate, kn is the NBOD decay rate

           # terms that will end up in final equation
           a1 = exp((-ka*x)/U)
           a2 = (kc/(ka-kc))*(exp((-kc*x)/U)-exp((-ka*x/U)))
           a3 = (kn/(ka-kn))*(exp((-kn*x)/U)-exp((-ka*x/U)))

           # eq for concentration
           conc = Cs*(1-a1)+Co*a1-Bo*a2-No*a3

           # eq for CBOD
           CBOD = Bo*exp((-kc*x)/U)

           # eq for NBOD
           NBOD = No*exp((-kn*x)/U)

           return conc, CBOD, NBOD
       end
calc_conc (generic function with 1 method)

julia> # initialize for plotting
       C = zeros(51);

julia> B = zeros(51);

julia> N = zeros(51);

julia> # find vals at x=0
       init_DO = ((7.5*1000*100000)+(5*1000*10000))/(1000*100000+1000*10000)
7.2727272727272725

julia> init_CBOD = ((5*1000*100000)+(50*1000*10000))/(1000*100000+1000*10000)
```

```
9.090909090909092

julia> init_NBOD = ((5*1000*100000)+(35*1000*10000))/(1000*100000+1000*10000)
7.7272727272727275

julia> C[1] = init_DO;

julia> B[1] = init_CBOD;

julia> N[1] = init_NBOD;

julia> # from x=1 to x=15
       for i in 2:16
         C[i], B[i], N[i] = calc_conc(i-1, 6, 10, init_DO, init_CBOD,
init_NBOD, .55, .35, .25)
       end

julia> # from x=15 to x=50
       init_DO2 = ((C[16]*1000*110000)+(5*1000*15000))/(1000*110000+1000*15000)
4.896480650455226

julia> init_CBOD2 =
((B[16]*1000*110000)+(45*1000*15000))/(1000*110000+1000*15000)
8.734896157428066

julia> init_NBOD2 =
((N[16]*1000*110000)+(35*1000*15000))/(1000*110000+1000*15000)
7.839777713929134

julia> for i in 16:51
         C[i], B[i], N[i] = calc_conc(i-16, 6, 10, init_DO2, init_CBOD2,
init_NBOD2, .55, .35, .25)
       end

julia> # plot DO
       using Plots

julia> plot(C, title= "Dissolved Oxygen (DO) concentration", label= "DO
concentration", xlabel="Distance (km)", ylabel= "DO (mg/L)")
```



Problem 1.2

```
julia> # initialize distance from waste stream 2
       distfrom1=16
16

julia> # find distance where DO recovers to 6 mg/L
         while C[distfrom1] < 6
           global distfrom1 += 1
         end

julia> println(distfrom1)
43

julia> println(C[distfrom1])
6.121508813045605
```

The distance at which the stream recovers to a DO of 6 mg/L if both waste streams are untreated is around 43 m after the first waste stream, or around 27 m after the second waste stream.

Problem 1.3

```
julia> function find_treatment_min(U, Cs, Co, Bo, No, C1, B1, N1, C2, B2, N2,
ka, kc, kn, E1, E2)

           # find new treated vals for waste stream 1
           CBOD_treated1 = (1-(E1/100))*B1
```

```
           NBOD_treated1 = (1-(E1/100))*N1

           # find stream vals at x=0
           initial_DO1 =
((Co*1000*100000)+(C1*1000*10000))/(1000*100000+1000*10000)
           initial_CBOD1 =
((Bo*1000*100000)+(CBOD_treated1*1000*10000))/(1000*100000+1000*10000)
           initial_NBOD1 =
((No*1000*100000)+(NBOD_treated1*1000*10000))/(1000*100000+1000*10000)

           # find stream vals at x = 15
           DO_15, CBOD_15, NBOD_15 = calc_conc(15, U, Cs, initial_DO1,
initial_CBOD1, initial_NBOD1, .55, .35, .25)

           # find new treated vals for waste stream 2
           CBOD_treated2 = (1-(E2/100))*B2
           NBOD_treated2 = (1-(E2/100))*N2

           # find stream vals after waste stream 2
           initial_DO2 =
((DO_15*1000*110000)+(C2*1000*15000))/(1000*110000+1000*15000)
           initial_CBOD2 =
((CBOD_15*1000*110000)+(CBOD_treated2*1000*15000))/(1000*110000+1000*15000)
           initial_NBOD2 =
((NBOD_15*1000*110000)+(NBOD_treated2*1000*15000))/(1000*110000+1000*15000)

           # find minimum for the treatment
           Cmin = 1000
           for i in 1:36
             newC, newB, newN = calc_conc(i-1, U, Cs, initial_DO2,
initial_CBOD2, initial_NBOD2, ka, kc, kn)
             if newC < Cmin
               Cmin = newC
             end
           end
         return Cmin

       end
find_treatment_min (generic function with 1 method)

julia> # create vector of all possible treatments
       treatment_plans = collect(0:100)
101-element Vector{Int64}:
   0
   1
   2
   3
   4
   5
   6
   7
   8
   9
   ⋮
  92
  93
  94
```

```
  95
  96
  97
  98
  99
 100

julia> # initialize vector to store minimums
       mins = zeros(101);

julia> # iterate treatments
       for i in 1:101
         mins[i] = find_treatment_min(6, 10, 7.5, 5, 5, 5, 50, 35, 5, 45, 35,
.55, .35, .25, 0, treatment_plans[i])
       end

julia> # minimum %removal that has DO always less that 4 mg/L
       tr = 1
1

julia> while mins[tr] < 4
         global tr +=1
       end

julia> println(tr-1)
12

julia> println(mins[tr])
4.007015974604083
```

The minimum treatment plan that guarantees all DO values are above 4 mg/L is waste stream 2 having a treatment of 12%.

Problem 1.4

```
julia> # create vector of all possible treatments
       treatment_plans = collect(0:100)
101-element Vector{Int64}:
   0
   1
   2
   3
   4
   5
   6
   7
   8
   9
   ⋮
  92
  93
  94
  95
  96
  97
```

```
 98
 99
100

julia> # initialize vector to store minimums
       mins2 = zeros(101);

julia> # iterate treatments, each stream is treated equally
       for i in 1:101
         mins2[i] = find_treatment_min(6, 10, 7.5, 5, 5, 5, 50, 35, 5, 45, 35,
.55, .35, .25, treatment_plans[i], treatment_plans[i])
       end

julia> # minimum %removal that has DO always less that 4 mg/L
       tr2 = 1
1

julia> while mins2[tr2] < 4
         global tr2 +=1
       end

julia> println(tr2-1)
7

julia> println(mins2[tr2])
4.017179459831182
```

If the streams are treated equally, the treatment value that keeps the DO above 4 mg/L is 7% for each stream.

Problem 1.5

In order to make a decision about the two treatment plans above, cost should be considered, as well as the plants' willingness to participate in a treatment plan. The first plan, only requiring waste stream 2 to be treated, is not fair to the second plant as the first plant does not need to treat their waste. However, treating both waste streams would likely be more expensive, or it would be tough getting both plants to agree to the treatment plan. Realistically, it would be best to choose the cheaper option despite the unfairness of the situation. In a true setting this choice would not be that simple, and other factors would be necessary to consider. However with the information we are given and my general knowledge of the engineer/client relationship, the cheapest option, where only waste stream 2 is treated, would be my choice.

Problem 1.6

```
julia> using Distributions

julia> # initialize vector to store probabilities
       probs = zeros(100);

julia> # iterate to find probabilities
       for j in 1:100
```

```
  # initialize
  global mins3 = zeros(100);
  global fails_regulation = zeros(0);
  global CBODs = zeros(100);
  global NBODs = zeros(100);

  # create vectors for CBOD, NBOD that have uniform dist
  for i in 1:100
    CBODs[i] = rand(Uniform(4, 7));
    NBODs[i] = rand(Uniform(3, 8));
  end

  # find the treatment min for each scenario
  for i in 1:100
    mins3[i] = find_treatment_min(6, 10, 7.5, CBODs[i], NBODs[i], 5,
50, 35, 5, 45, 35, .55, .35, .25, 0, 12);
  end

  # find how many fail the regulation
  for i in 1:100
   if mins3[i] < 4
     append!(fails_regulation, mins3[i]);
   end
  end

  # calc probability
  probs[j] = (length(fails_regulation))/(length(mins3))

end

julia> # take average probability
       avg_prob = (sum(probs))/(length(probs))
0.7031000000000001

julia> println(avg_prob)
0.7031000000000001
```

The average probability that the treatment does not meet the requirement of DO above 4 mg/L is around 70%.

Problem 1.7

```
julia> # initialize vector to store probabilities
       probs = zeros(100);

julia> # iterate to find probabilities
       for j in 1:100

         # initialize
         global mins4 = zeros(100);
         global fails_regulation = zeros(0);
         global CBODs = zeros(100);
         global NBODs = zeros(100);

         # create sample vector from function
```

```
        sample_CBOD_NBOD = sample_correlated_uniform(100, [4,7], [3,8]);

        # create vectors for CBOD, NBOD that have uniform dist
        for i in 1:100
          CBODs[i] = sample_CBOD_NBOD[i,1];
          NBODs[i] = sample_CBOD_NBOD[i,2];
        end

        # find the treatment min for each scenario
        for i in 1:100
          mins4[i] = find_treatment_min(6, 10, 7.5, CBODs[i], NBODs[i], 5,
50, 35, 5, 45, 35, .55, .35, .25, 0, 12);
        end

        # find how many fail the regulation
        for i in 1:100
         if mins4[i] < 4
           append!(fails_regulation, mins4[i]);
         end
        end

        # calc probability
        probs[j] = (length(fails_regulation))/(length(mins3))

      end

julia> # take average probability
       avg_prob = (sum(probs))/(length(probs))
0.6478

julia> println(avg_prob)
0.6478
```

The average probability that the treatment does not meet the requirement of DO above 4 mg/L is around 64%.

Problem 1.8

The uncertainty here must be taken into account. The uncorrelated values while following the previously discussed treatment plan in 1.5 resulted in an uncertainty of 70%. This was slightly lower at 64% when CBOD and NBOD were related. Both of these values have an unacceptable percent of scenarios that fail to reach the regulation. Cost should be considered as a guide for increasing the treatment efficiencies to a certain, feasible point. Additionally, even if reasonable treatment plan is found that minimizes the probability of failure as well as minimizes cost, other environmental factors could change these predictions and could be added to refine the model.
'''

References

Julia: append to an empty vector. Stack Overflow. https://stackoverflow.com/questions/28524105/
append-to-an-empty-vector

Julia: short syntax for creating (1,n) array. Stack Overflow. https://stackoverflow.com/questions/45
short-syntax-for-creating-1-n-array

Repeated Evaluation: Loops. Control Flow. Julia Documentation. https://docs.julialang.org/en/v1
flow/#man-loops

Univariate Distributions. Distributions.jl. Julia Documentation. https://docs.juliahub.com/Distribu