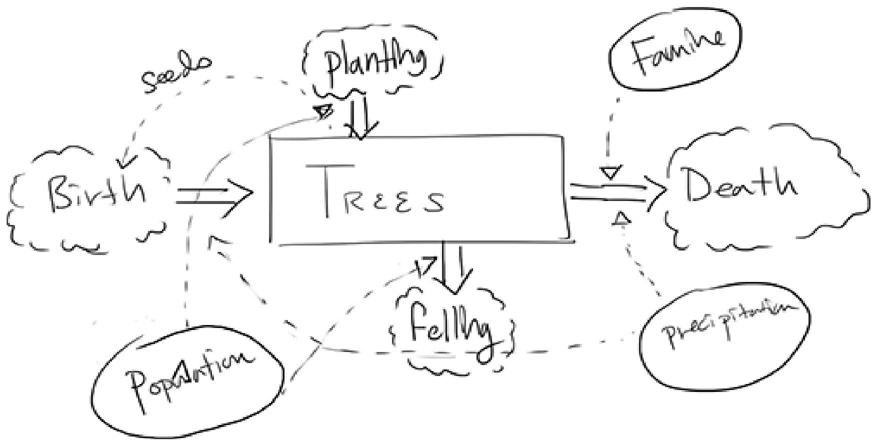
Exercise 1-1.

1. Stocks
2. Stocks
3. Stocks
4. ~~Flows~~ Neither; after I sought explanation from the solutions, I understood it as it neither represented a quantity, nor contributed to it.
5. ~~Stocks~~ Neither; I thought since there is a set cap to the battery charge, it may as well be conceptualized as a contained quantity (like water in a bowl); i.e. although it is not a physical quantity, it can be mathematically quantified, in numbers, dependent on inflow(charging) and outflow(use)
6. Flows
7. ~~Stocks & Flows~~ Neither; I thought speed is a stock since it is certainly a quantity that stays put (the law of inertia) without external interference, which is acceleration/deceleration. Likewise, I thought speed is also a flow since it impacts the distance of the runner from a fixed point. Although the solution does not regard distance as a stock, I reasoned that since distance is also a scalar quantity, it might as well be a stock – I may be in for too much abstraction.
8. Stocks
9. ~~Stocks~~ The reasoning is akin to that of #7(since it represents a rate).
10. Stocks
11. ~~Stocks~~ The reasoning is akin to that of #7.
12. ~~Flows~~ I missed the "per adult" part.
13. Flows
14. Stocks
15. Flows
16. ~~Flows~~ Again, I thought temperature might as well be regarded as a stock, but I may be in for too much abstraction.

Exercise 1-2.



- Tree = The net amount of trees in the Northeastern United States
- Birth = "Source" of tree influx, impacted by seeds which are dependent on the size of tree stock.
- Death = Destination of dead trees.

- Population = Human population
- Planting = Deliberate efforts to reforestation, commanded by eco-loving people.
- Felling = Cutting down trees as to utilize them as resources, impacted by human demand.
- Famine = Diseases which often causes trees' decimation.
- Precipitation = Water supply, to which the sustenance of trees are reliant.
- I thought I was to delineate the relationship between the various labels in the diagram; to explicitly identify them in their conceptual representation :
- Source : Birth
- Sink : Death
- Stock : # of trees in Northeastern U.S.
- Inflow : Birth & Planting
- Outflow : Death & Felling
- Links & Variables : Human Population impacts Planting & Felling; # of trees impact the rate of birth; Precipitation affects the general growth and sustenance of trees – thus birth and death. Famine accounts for (partially) the death rate of trees.
- Generally, I feel that this model is more or less adequate in describing the dynamics of tree growth in New England; I certainly did not consider other natural factors, like sunlight, nutrients in the soil, harmful substances in the air, the supply of oxygen, etc., which I feel are often constant (at least in the observable time range) and would simply complicate the model.

Exercise 1-3.

(a) $dW/dt = G(W,a,i) - E(e) - H(o)$

(b) 1. $dI/dt = P(I,R,Df) - C(I,Dc)$;

2. $dP/dt = D(R,T,Df) - P(I,R,Df)$;

3. $dU/dt = -D(R,T,Df)$;

In (a):

- W = Water in Lake Mead <Stock>
- t = Time point (…though trivial)
- G = Inflow through Glen Canyon Dam <Flow>
- a = Availability of water <Variable>
- i = Inflow Policy <Variable>
- E = Outflow through Evaporation <Flow>
- e = Evaporation rate <Variable>
- H = Outflow through Hoover Dam <Flow>
- O = Outflow Policy <Variable>

In (b) :

- I = Current Inventory <Stock>
- t = Time point (trivial)

- P = Production <Flow>
- R = Proven Reserves <Stock>
- Df = Forecasted Demand <Variable>
- C = Consumption <Flow>
- Dc = Current Demand <Variable>
- D = Discovery <Flow>
- T = Technology <Variable>
- U = Ultimately recoverable oil <Stock>

Exercise 1-4.

update_stocks.m :

```
reserve = reserve- 32; %billion barrels
inventory = inventory + 32 - 32; %ditto
```

check_stocks.m :

```
disp(reserve); %billion barrels
disp(inventory); %ditto
```

Execution:

```
>> reserve =1700;
>> inventory = 6.9;
>> check_stocks
    1700
  6.9000

>> update_stocks
>> check_stocks
    1668
  6.9000

>> update_stocks
>> check_stocks
    1636
  6.9000

>> update_stocks
>> check_stocks
    1604
  6.9000
```

- Running the scripts thereafter yielded a linear decrease of the inventory (-32 billion barrels/year).

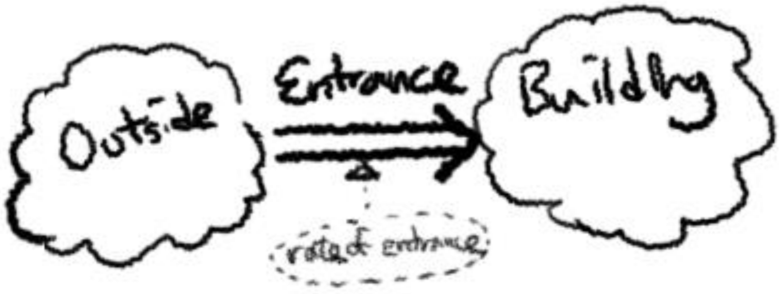
- This model is inadequate for the prediction of global oil supply, due to its naïve assumptions (constant rates). That is, it fails to take account of technological innovations which can factor into the efficiency of oil use and perhaps the demand for oil altogether: i.e. the development of alternative energy sources or – in the negative side – the emergence of a new industry which may greatly increase the demand for oil.

ModSim Exercise 2

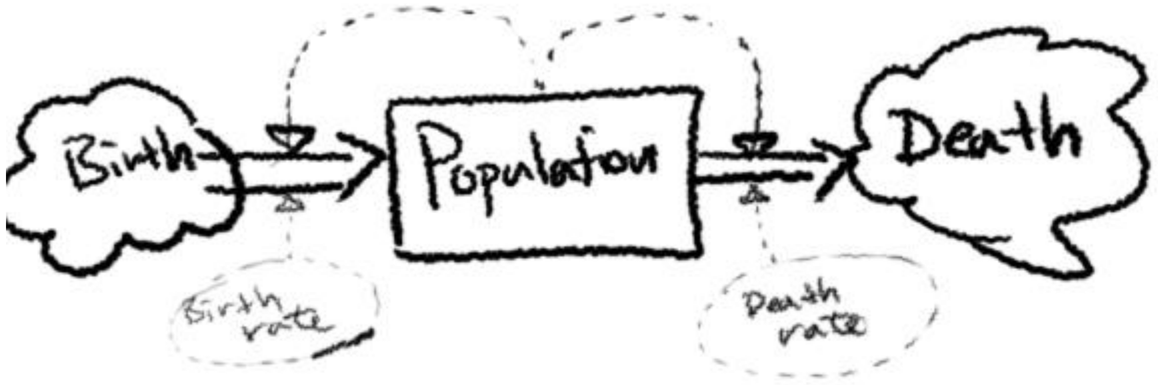
Jamie Cho

2.1)

1. Constant flows
   A. Given sufficient number of people wanting to enter a building, and finite number of known entrance, and the building spatial enough to contain every person -- in terms of modeling, this implies sources (people outside), sinks (building), and regulated rate of inflow (entrance), the inflow of people to the building will be constant.



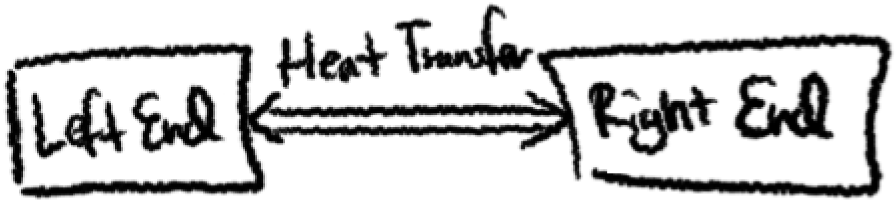
   B.
   C. $E = \frac{dP}{dt} = r_e$ (const. var.)
      - Units
        E: Entrance Flow, kg/minute
        P: Biomass of persons, kg
        t: Time, minute
        Although number of person is a discrete quantity, and so would be the flow, with adequate number of people and mathematical assumptions, a continuous model can be adopted.
2. Flows proportional to single stock
   A. The inflow (birth) and the outflow (death) of people depends on the size of human population.



   B.
   C. $\frac{dP}{dt} = B(P) - D(P) = k_b P - \ k_d P$
      - Units
        P: Biomass of the population, Metric Tons
        t: Time, year

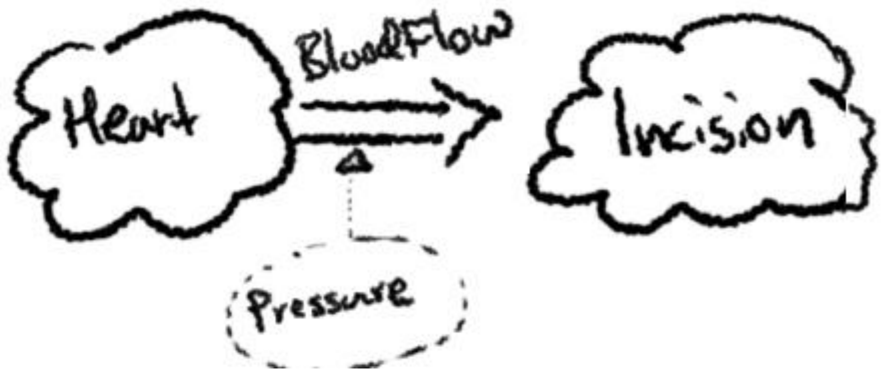
B: Birth, Metric Tons/year
D: Death, Metric Tons/year
k_b: Birth rate const., unitless scalar
k_d: ,Death rate const., unitless scalar

3. Flows proportional to the difference between two stocks
   A. The heat flow in a conductible material is dependent on the magnitude of enthalpy(amount of thermal energy) on either ends.



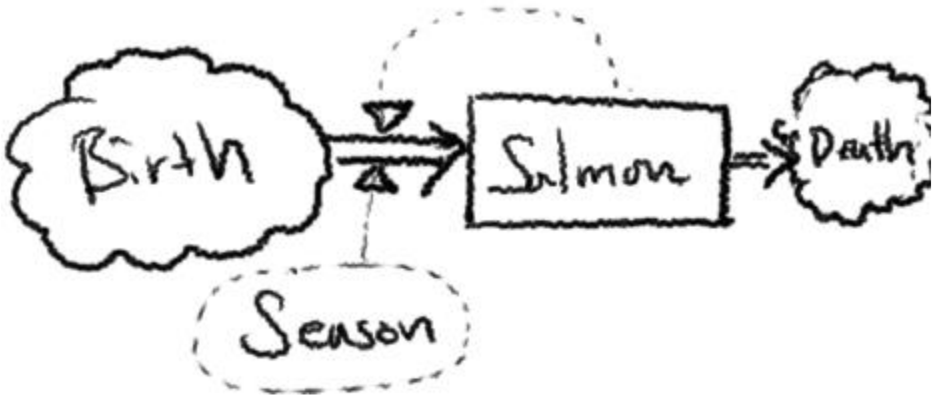
   B.
   C. $Q = \frac{dE}{dt} = k_T \Delta E = k_T(E_L - E_R)$
      - Units
        Q: Heat, Joules/second
        E: Enthalpy, Joules
        t: Time, seconds
        k_T: Heat transfer constant, unitless scalar (for the purposes of modeling)
        E_L: Enthalpy of the Left end, Joules
        E_R: Enthalpy of the Right end, Joules

4. Non-monotonic flows
   A. The flow of blood from an incision in the artery, which would depend on the pressure of blood outflow as dictated by the heart.



   B.
   C. $\frac{dB}{dt} = F_B(P)$
      - Units
        B = Blood, Liters
        t = Time, seconds
        F_B = Flow of Blood, Liters/second
        P = Pressure, external variable dependent on factors such as time.

5. Flows that depend on time
   A. The inflow of the biomass of salmon stock depends on time, as its reproduction occurs seasonally – the “salmon run” occurs during the fall, September through November.

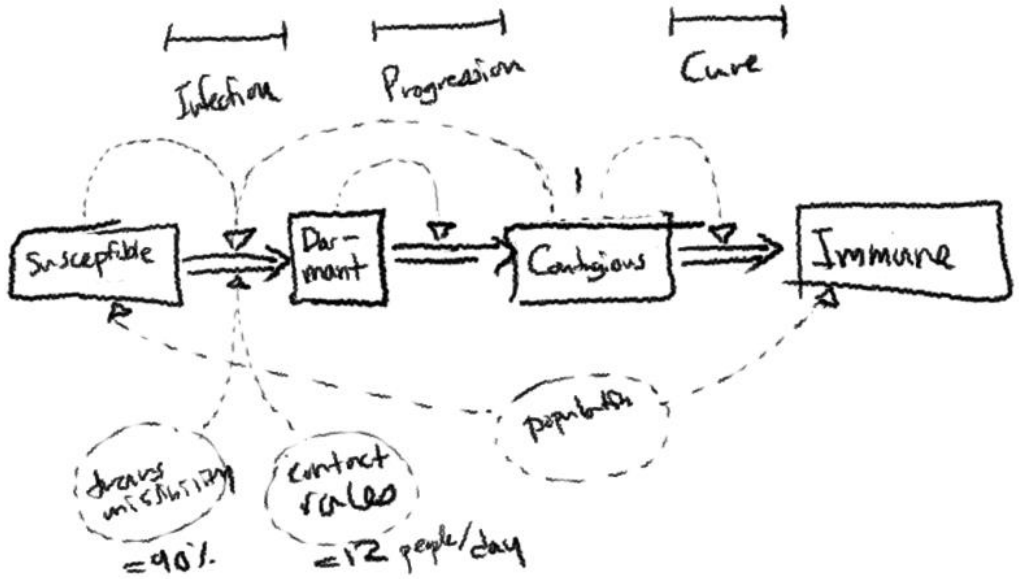


   B.
   C. $\frac{dP_S}{dt} = B(s(t), P_S) - D(P_S)$

      Units
      P_S = Biomass of Salmon, Metric Tons
      t = Time, month
      B = Birth flow, Metric Tons/month
      s = season (arbitrary variable)
      D = Death (or Biomass dissipation due to death) flow, Metric Tons/month

2.2)

1.



2.

$$F_I = \frac{dS}{dt} = C * k_r * k_t * \frac{S}{P}$$

$$F_P = \frac{\mathrm{dD}}{\mathrm{dt}} = \mathrm{D}(13), \mathrm{F}_C = \frac{dC}{dt} = C(9)$$

Which is equivalent to:

Infection flow = Contagious *Contact Rates*(Transmissibility*Susceptible/Population)

Progression flow = Population at 13th day of Dormant Population

Cure flow = Population at 9th day of Contagious Population

Data Plots:

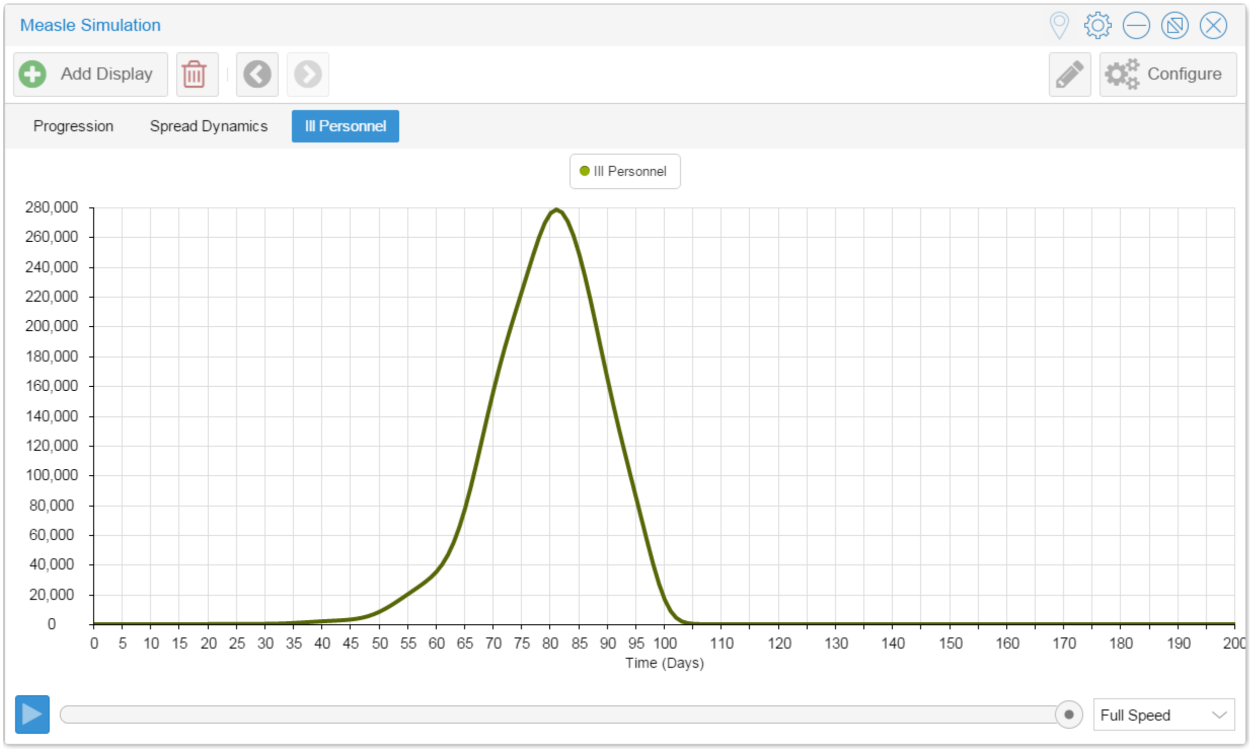


Figure 1. The time series of the number of Ill Personnel.

3.

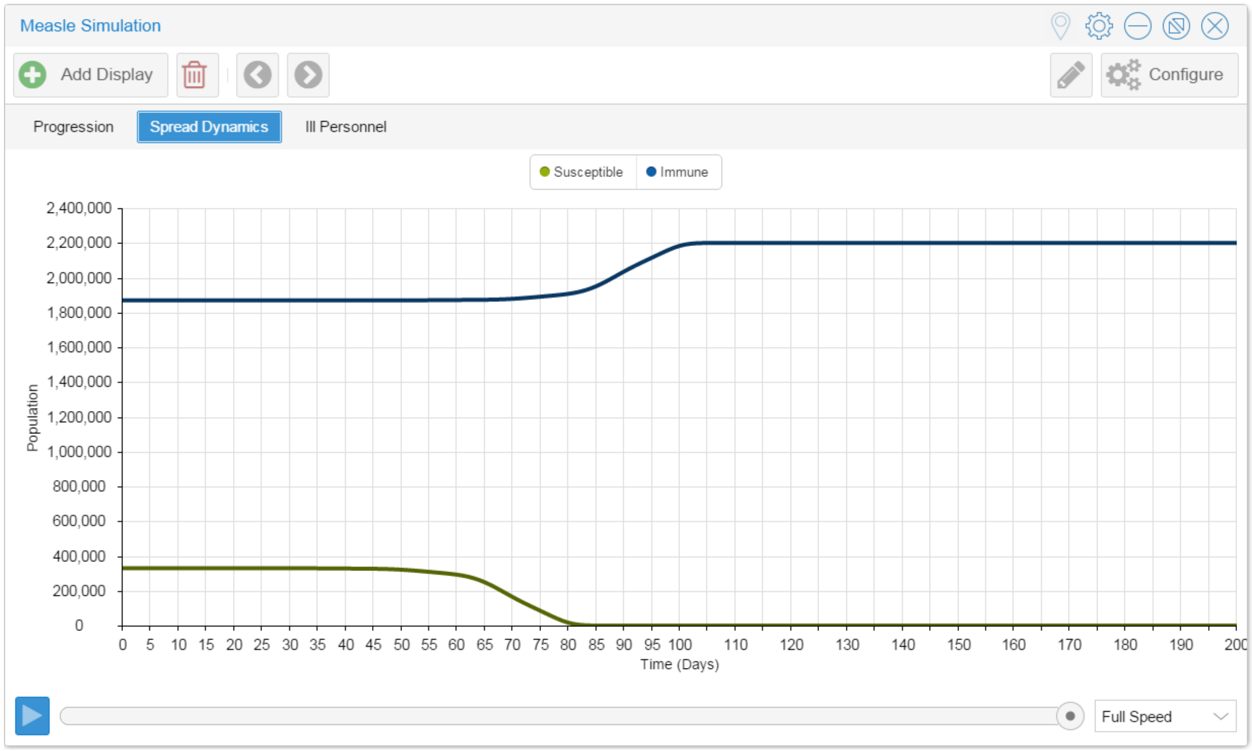


Figure 2. The Times series that illustrates the transition in population dynamics with respect to measles.

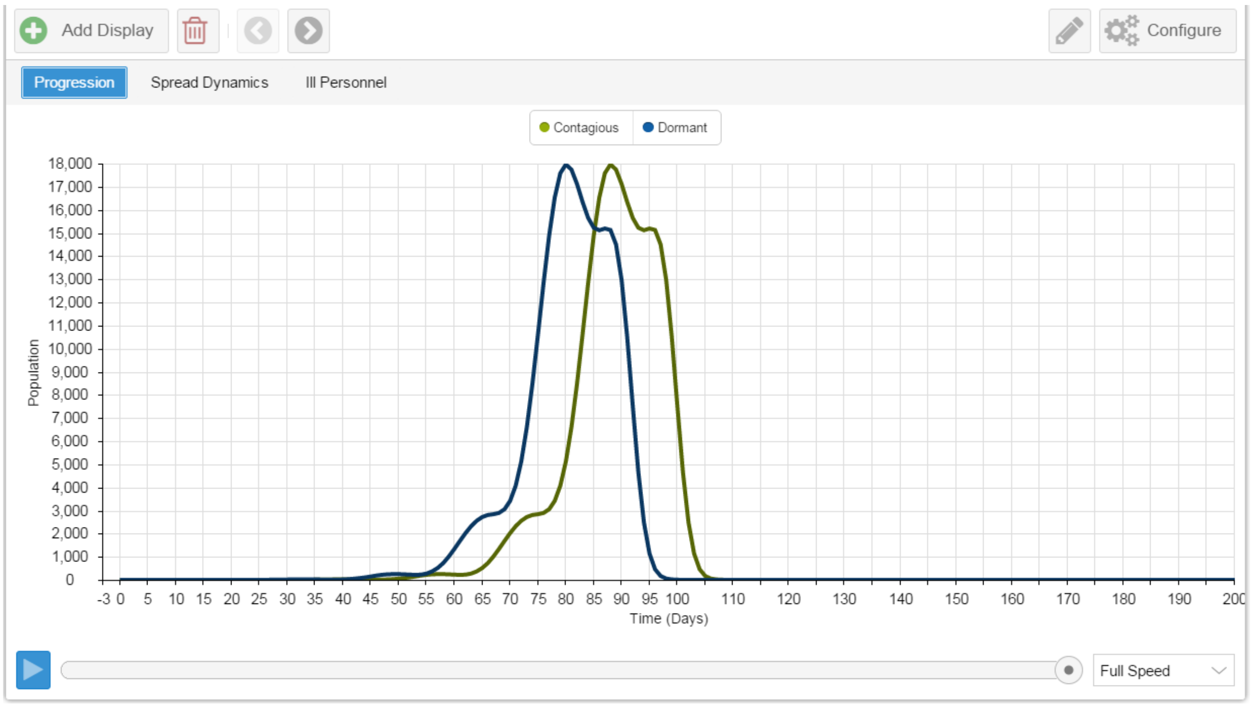


Figure 3. The Time series that depicts the intimate relationship between contagious and dormant persons.

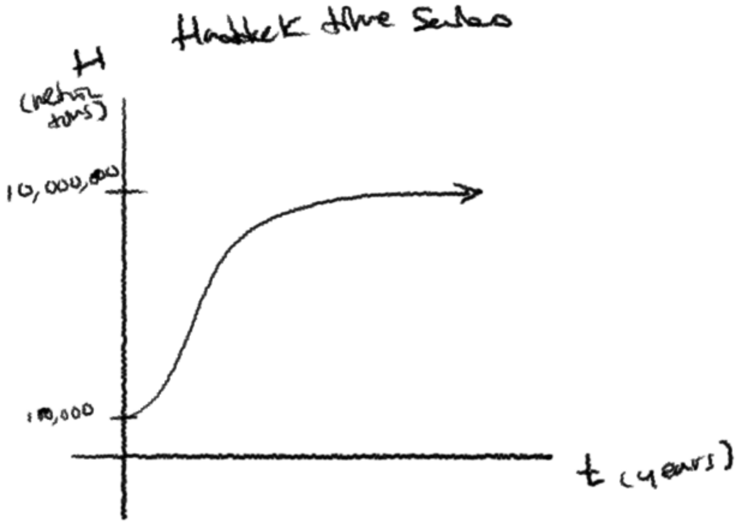
3.

In my implementation, the epidemic peaks the 81st day with 278,518 persons affected. After the 111th day, the number of ill personnel was zero; after the 90th day, no more spread of the disease was observed, and all but 1 of the population had undergone the process of the disease. Although this does not match practical data from experience, this model is based on the assumption that afflicted persons freely interact with any of the population – and fails to take into account the societal efforts to thwart the disease, which serves as an adequate explanation of why this model might fail to reflect on the reality.

When I altered the immunization percentage from 85% to 90%, 163,611 persons were ill on the 96th day (Peak). This is not only a display of delaying the spread of the disease – which provides longer time to prepare and counteract the epidemic – but also the evidence of greatly reduced severity. On the other hand, when I limited the contact rates from 12 persons a day to 6 persons a day, greater delay in time – peaking at the 113th day – was observed, with 214939 persons ill. Thus, it seems that the contact rates have greater effect in delaying the propagation of the disease than reducing its severity, which is logically coherent with intuition.

2.3)

1.



2. The time series looked congruent, with minor discrepancies in curvature magnitude.

3.

update_haddock.m

```
%precondition : varialbe H represents Haddock Stock
% g is the growth rate constant,
% and K is the carrying capacity. (same conventions from exercise)
% the units are in metric tons, per year.
```

```
H = H + g*H*(1-H/K);
%postcondition : H now contains the changed Haddock Stock after an year.
```

plot_haddock.m

```
H = 100000; %metric tons
K = 10000000; %metric tons
g = 0.10; % years^-1
hold on
for i = 0:100
   plot(i,H,'*');
   update_haddock;
end
```

The two plots looked nearly identical.

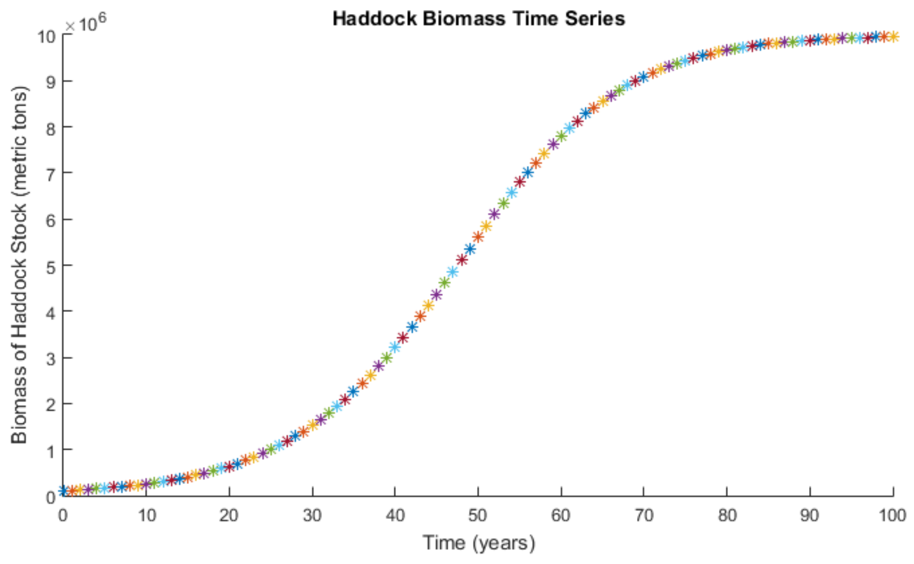


Figure 2. Matlab Time Series

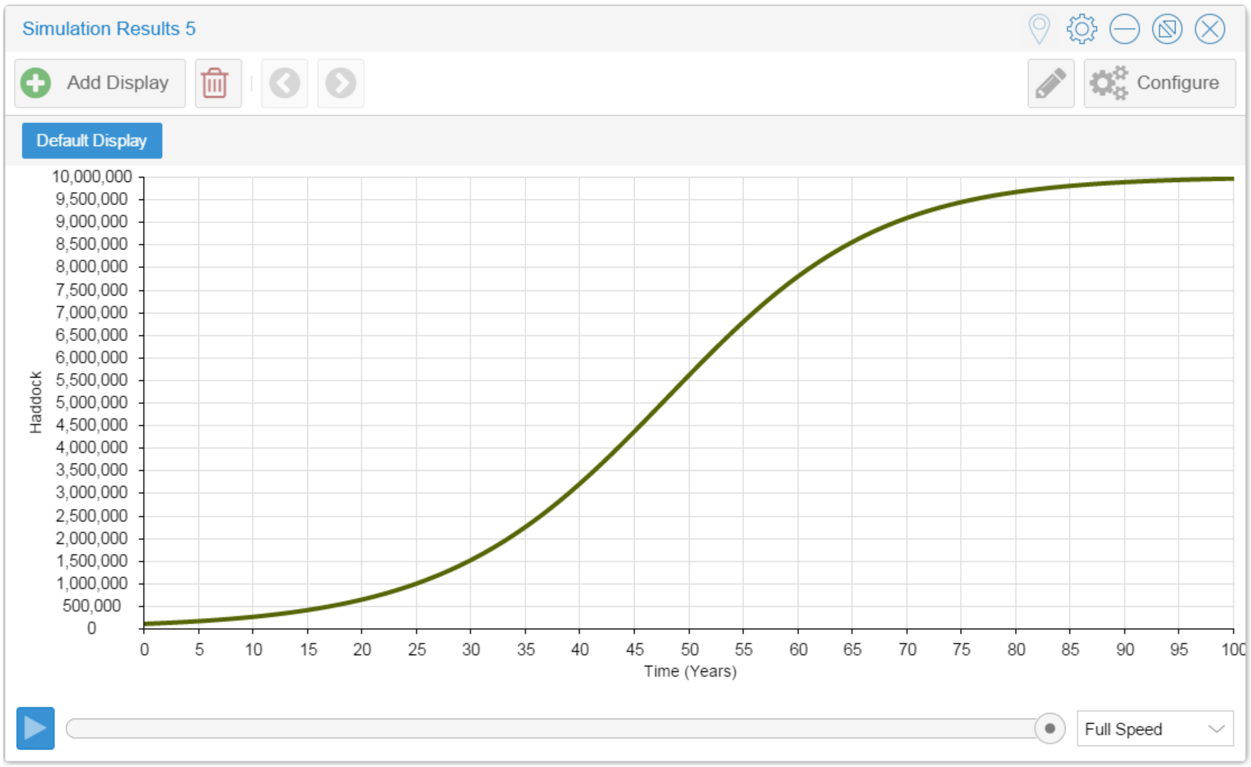


Figure 3. Insight Maker Time Series

ModSim Exercise 3

Jamie Cho

1.

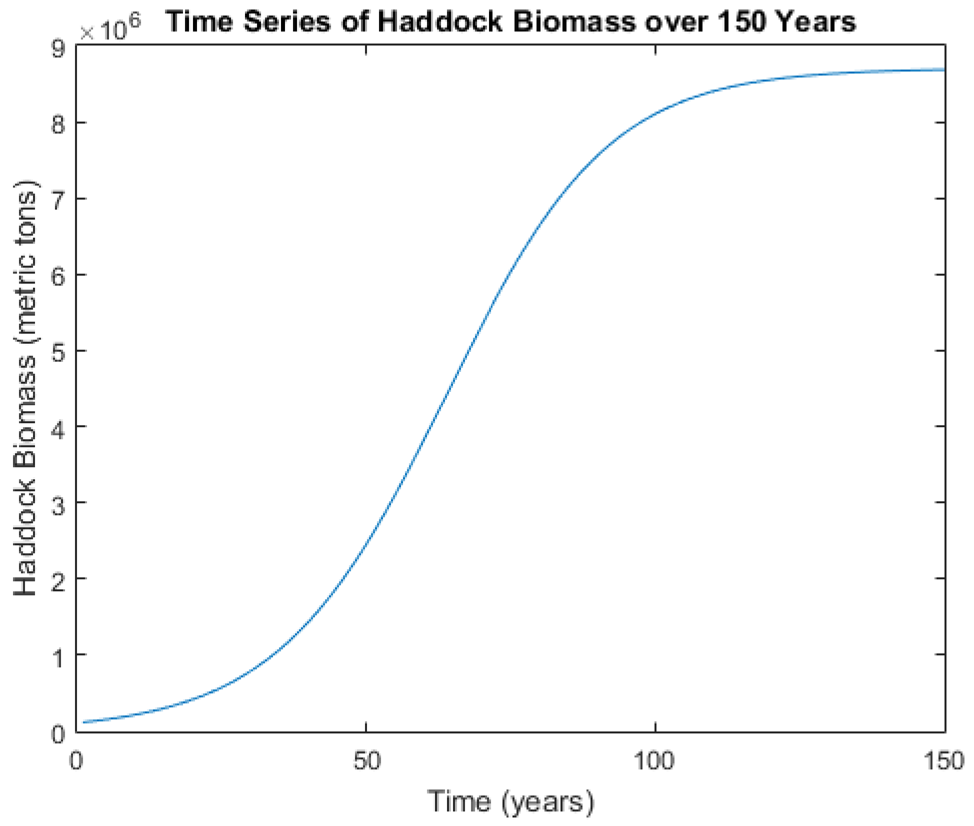
1) haddock_flow.m:

```
%res = haddock_flow(H,K,g)
%computes the flow level from
%current value of Stock, Carrying Capacity, and
%growth rate.
function res = haddock_flow(H,K,g)
    res = g*H*(1 - H/K);
end
```

2) simulate_haddock3.m:

```
%simulate_haddock3()
%simulates haddock stock population
%over the course of 150 years.
function simulate_haddock3()
    haddock = 120000;%metric tons
    cap = 8.7*1000*1000;%metric tons
    growth = 0.07; %per year

    hVec = zeros(150,1);
    for i = 1:150
        hVec(i) = haddock;
        haddock = haddock + haddock_flow(haddock,cap,growth);
    end

    disp(haddock);
    plot(hVec);
    xlabel('Time (years)');
    ylabel('Haddock Biomass (metric tons)');
    title('Time Series of Haddock Biomass over 150 Years');
end
```

3) resultant plot:



2.

1) infection_flow.m:

```
%res = infection_flow(S,P,C,k_c,k_t)
%returns infection flow amount with values:
%S = Susceptible Personnel
%P = Total Population
%C = Contagious Personnel
%k_c = contact rate constant
%k_t = transmissivity constant
function res = infection_flow(S,P,C,k_c,k_t)
    res = round(k_c*k_t*S/P*C);
end
```

2) progression_flow.m:

```
%res = progression_flow(D)
%D = Last Element of Dormant Population
function res = progression_flow(D)
    res = D;
end
```

3) cure_flow_m:

```
%res = cure_flow(C)
```

```
%C = Last Element of Contagious Population
function res = cure_flow(C)
    res = C;
end
```

4) simulate_measles_m:

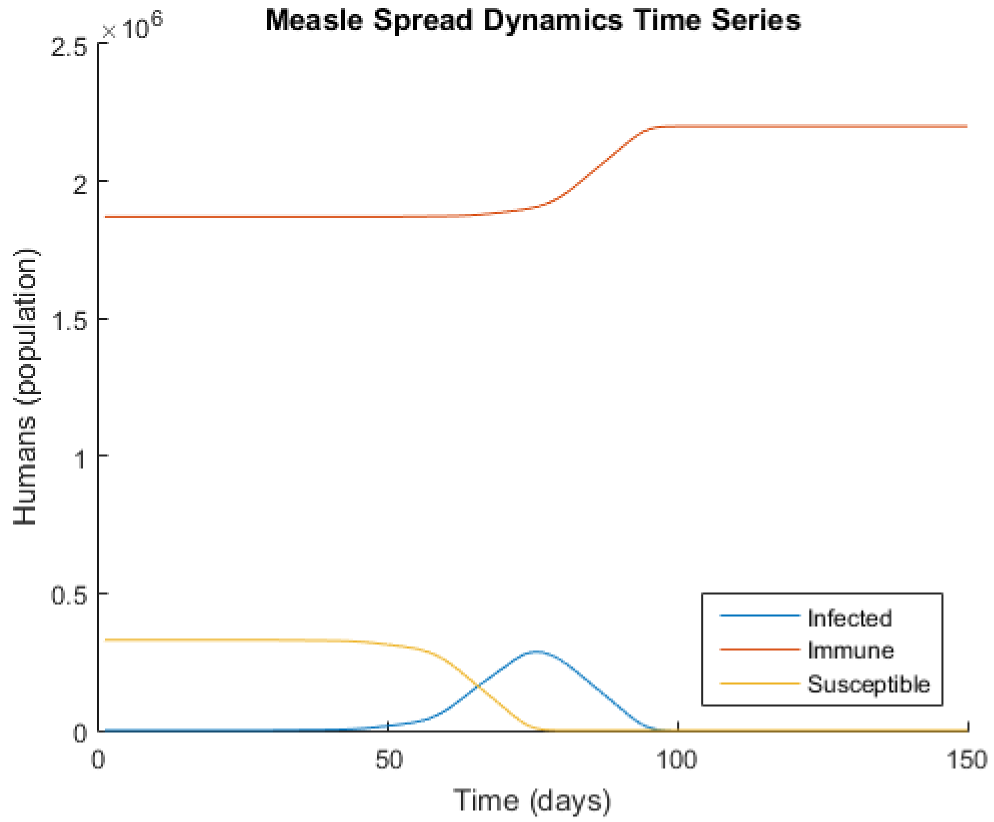
```
%simulate_measles(vaccination, population, contact_rates,
transmissibility)
%simulation of measle spread
%with internal given constants
%and control variables.
%Units:
%vaccination = decimal
%population = 1 person
%contact_rates = person / day
%transmissibility = person / contact

function simulate_measles(vaccination, population, contact_rates, ...
    transmissibility)
    %controls
    %vaccination = 0.85;
    %population = 2200000;
    %contact_rates = 12;
    %transmissibility = 0.9;

    %initialization
    contagious = zeros(8,1);
    dormant = zeros(12,1);
    immune = vaccination * population;
    susceptible = (1 - vaccination) * population;
    contagious(1) = 1;

    %plotting vectors
    vInfected = zeros(150,1);
    vImmune = zeros(150,1);
    vSusceptible = zeros(150,1);

    %simulation
    for time = 1:150
        immune = immune + cure_flow(contagious(end));

        for i = length(contagious)-1:-1:1
            contagious(i+1) = contagious(i);
        end
        contagious(1) = progression_flow(dormant(end));
        for i = length(dormant)-1:-1:1
            dormant(i+1) = dormant(i);
        end
        infection =
infection_flow(susceptible,population,sum(contagious),...
            contact_rates,transmissibility);
        dormant(1) = infection;
        susceptible = susceptible - infection;
```

```
            vInfected(time) = sum(dormant) + sum(contagious);
            vImmune(time) = immune;
            vSusceptible(time) = susceptible;
        end
        hold on
        plot(vInfected);
        plot(vImmune);
        plot(vSusceptible);
        title('Measle Spread Dynamics Time Series');
        xlabel('Time (days)');
        ylabel('Humans (population)');
        legend('show')
        legend('Infected','Immune','Susceptible','Location','southeast');
        hold off
    end
```

5) command line:

```
>> simulate_measles(0.85,2200000,12,0.9);
```

6) resultant plot:



In both scenarios, the plots were equivalent to that of InsightMaker.

- Note: as the implementation of the model was strictly based on the daily propagation from one state to the next – i.e., the “contagious” group will be cured in 8 days – the arrangement in time steps could not be accommodated.

The major difference in the approach of the Solution and my solution was due to the different method by which the implementation of the scenario occurred: I tracked the state-transitions of each group in the population: on the 14th day, the dormant period would end and the infected person would become contagious. This is a different approach from the solution that applied simpler probabilistic mechanism to the flow.

ModSim Exercise 4

Jamie Cho

1.

1) alcohol_net_flows.m

```
%params(1) = d
%params(2) = k
%params(3) = Vm
%params(4) = Km
function res = alcohol_net_flows(stock, params)
res = [params(1) - params(2)*stock(1), ...
    params(2)*stock(1) - params(3)*stock(2)/(params(4)+stock(2))];
end
```

2) simulate_alcohol.m

```
function simulate_alcohol()

stocks = [.9,0];
params = [0,0.24/60,1.31/60,3.67];
%params = [0,1/.7*(1/3600),.4/3600,.2];
lim = 5*60*60;

V_s1 = zeros(lim,1);
V_s2 = zeros(lim,1);
for i=1:lim
    stocks = stocks + alcohol_net_flows(stocks,params);
    V_s1(i) = stocks(1);
    V_s2(i) = stocks(2);
end
hold on
plot(V_s1);
plot(V_s2);
end
```

3) simulate_alcohol2.m

```
function simulate_alcohol2()

stocks = [.9,0];
params = [0,0.24/60,1.31/60,3.67];
%params = [0,1/.7*(1/3600),.4/3600,.2];
lim = 5*60*60;
stepsize = 100;

V_s1 = zeros(lim/stepsize,1);
V_s2 = zeros(lim/stepsize,1);

for i=1:stepsize:lim
    stocks = stocks + stepsize*alcohol_net_flows(stocks,params);
    V_s1(1+floor(i/stepsize)) = stocks(1);
    V_s2(1+floor(i/stepsize)) = stocks(2);
end
hold on
```

```
plot(V_s1);
plot(V_s2);
end
```

Graph (Unitless concentration):

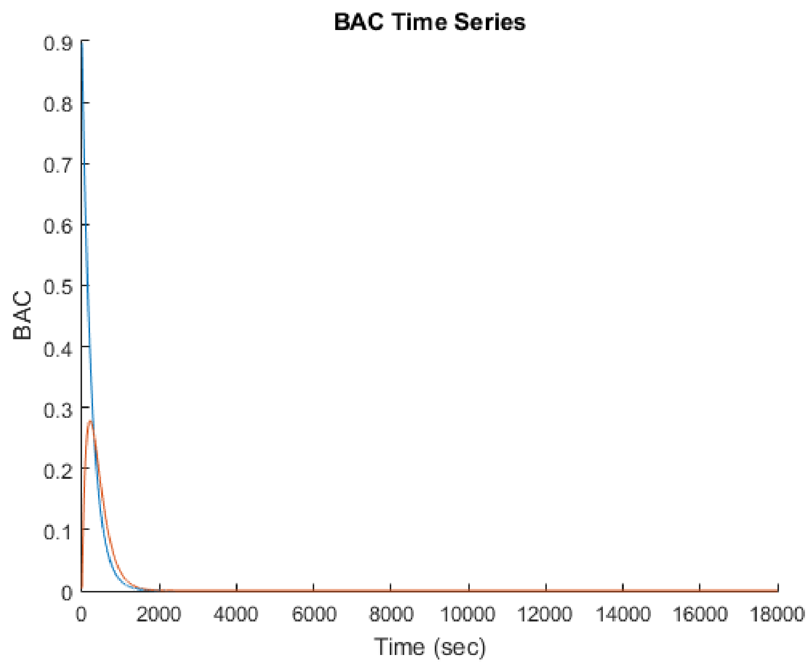


Figure 1 BAC Time Series, simulated per every second

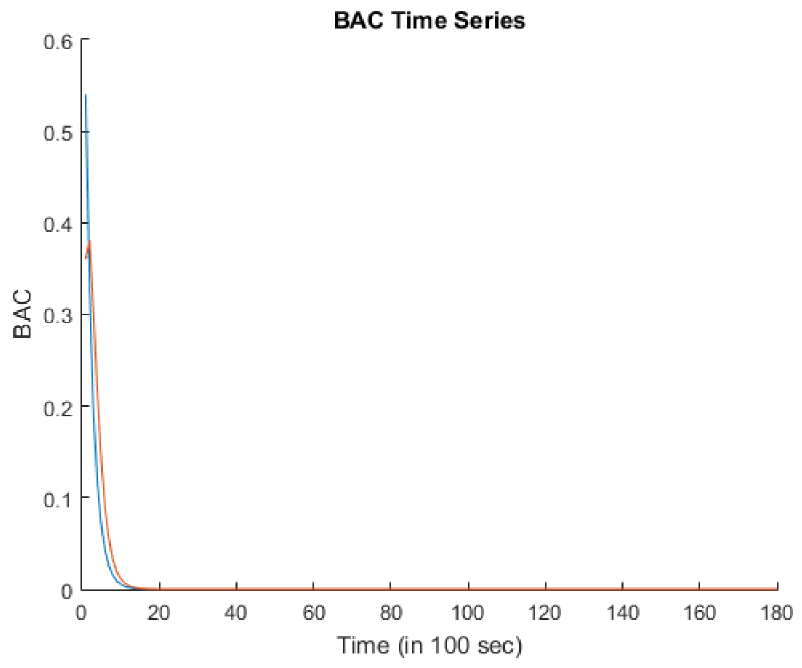


Figure 2. BAC Time Series, simulated per every 100 seconds.

2. duck.m:

```
function duck()
    figure;
    rho = 0.3; %g/cm^3
    rad = 10; %cm
    V_tot = @(r) 4*pi*r^3/3;
    V_sub = @(r,d) pi/3*(3*r*d^2-d^3);
    eqn = @(d) V_sub(rad,d)*1 - V_tot(rad)*rho;
    ezplot(eqn,[-rad*2,rad*2]);
    disp(fzero(eqn,rad));


    figure;
    plotVec = zeros(100,100);
    eqn2 = @(rh,rd) V_tot(rd)*rh - V_sub(rad,10)*1;
    for i = 1:100
        for j = 1:1000
            plotVec(i,j) = eqn2(i/10,j);
        end
    end
    mesh(plotVec);
    xlabel('radius (mm)');
    ylabel('density (g/cm^3)')
    %for i = 1 : 20
     %  for j = 1 : 20
     %      plotVec(i,j) = eqn
     %  end
    %end
    %figure;

end
```

fzero result : 7.2651 cm

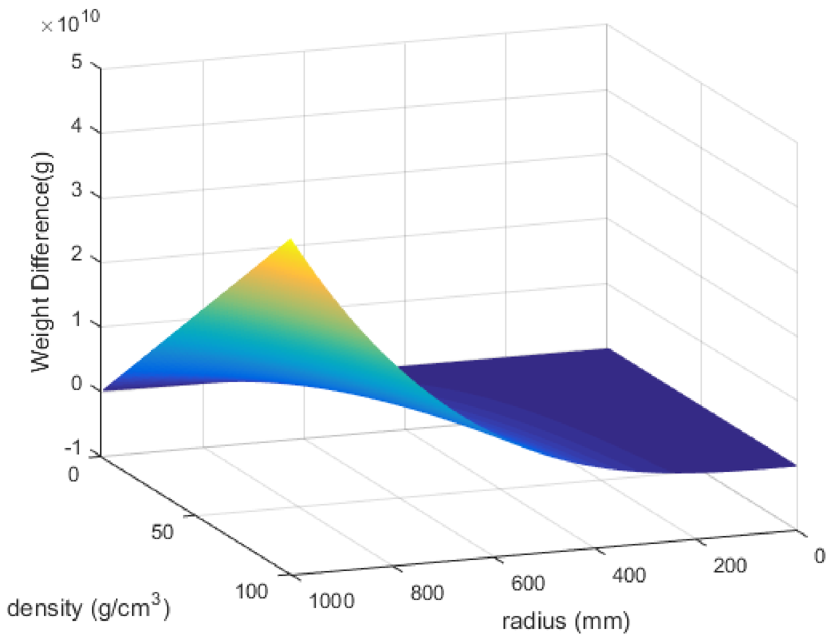


Figure 3 3d plot of weight difference with respect to density and the radius of the duck, given the displacement of 10 cm.

ModSim Exercise 5

Yoonyoung Cho

October 11 2015

1 Simple Conduction Model

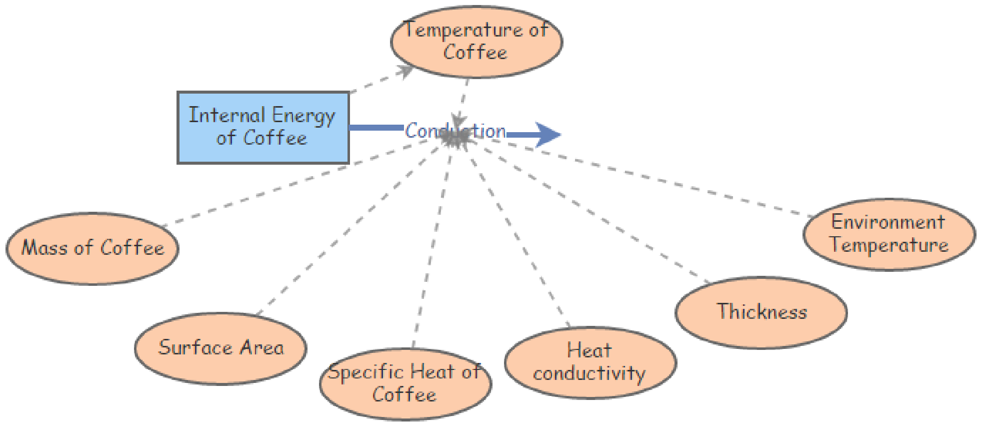


Figure 1: Rudimentary representation of the coffee-cup system

$$\begin{aligned}
\Delta U &= mc\Delta T \\
\Delta U &= Q - (W = 0) \\
Q &= -\frac{kA(T_{self} - T_{env})}{d} \\
mc\Delta T &= -\frac{kA(T_{self} - T_{env})}{d} \\
\therefore \Delta T &= -\frac{kA(T_{self} - T_{env})}{dmc}
\end{aligned}$$

In its MATLAB implementation, this flow was modeled as follows:

```
%Part I

r = 8/2 / 100; %m
h = 10 / 100; %m
A = pi*r^2 + 2*pi*r*h; %total surface area, m^2

d = 0.7 / 100;%m
c = 4186; %J/kg*K
rho = 1000; %kg/m^3
```

```
k = 1.5; %W/(m*K)

m = rho*pi*r^2*h; %kg

T_s = 370; %K
T_e = 290; %K

conduction = @(k,A,T_s,T_e,d,m,c) -k*A*(T_s-T_e)/(d*m*c);

range = 30;
vCoffee = zeros(1,range);
intv = 1:range;

for t = intv
    vCoffee(t) = T_s;
    T_s = T_s + conduction(k*60,A,T_s,T_e,d,m,c);%k to minutes
end
plot(intv-1,vCoffee);
title('The Time Series of the Temperature of Coffee')
xlabel('Time (minutes)');
ylabel('Temperature of Coffee(K)');
```

The resultant plot:

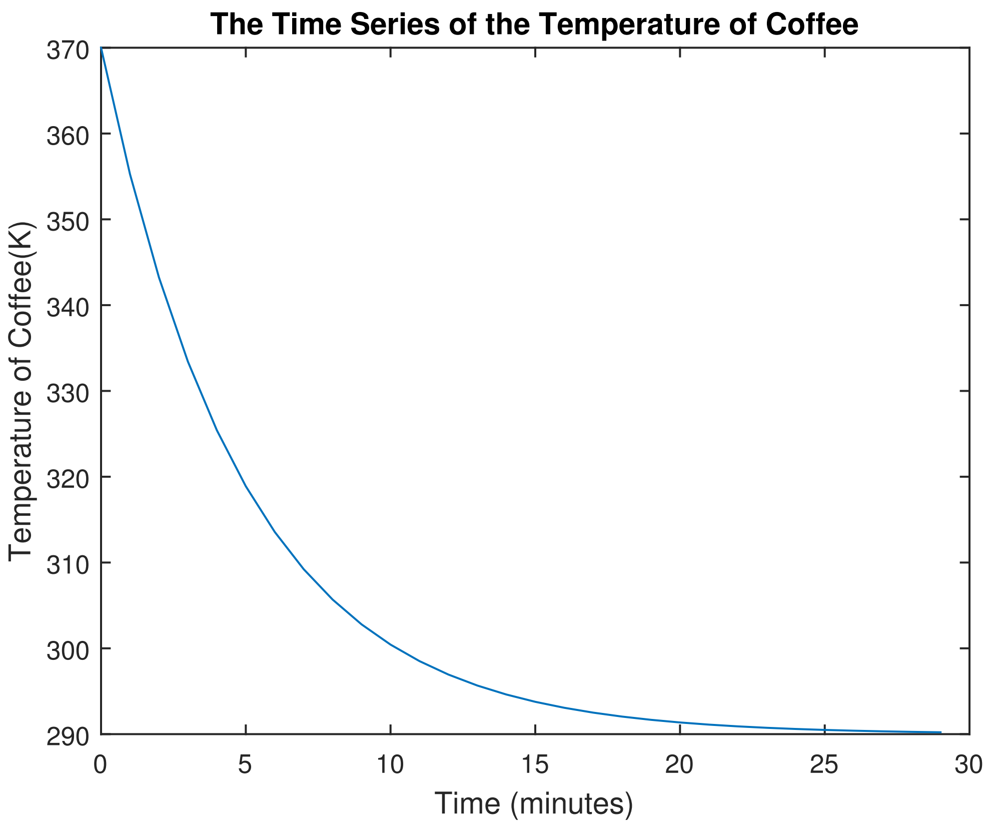


Figure 2: The cooling of 370K coffee over 30 minutes.

This graph is consistent to intuition; a hot coffee would not take more than 30 minutes to cool in room temperature; of course, with the addition of other factors, the coffee should cool down at a greater rate (unless given an external heat source).

2 Adding a Cream

In adding a cream, I would assume that the internal energy of the mixture after the addition is simply the sum of the internal energy of the two systems, and I would also set the internal energy to be proportional to the specific heat, mass, and the temperature of the respective substances. If I only knew the volume, I would further assume that the density of the cream similar to that of the coffee in order to deduce the mass. I will denote the coffee as system 1 and the cream as system 2 in the following qualitative analysis(as it is by no means numeric). Accordingly,

$$\begin{aligned}
E_{mix} &= E_1 + E_2 \\
E_1 &= k * c_1 * m_1 * T_1 \\
E_2 &= k * c_2 * m_2 * T_2 \\
E_{mix} &= k * c_1 * m_1 * T_1 + k * c_2 * m_2 * T_2 \\
&= k * c_1 * m_1 * T_{mix} + k * c_2 * m_2 * T_{mix} \\
T_{mix} &= \frac{k * c_1 * m_1 * T_1 + k * c_2 * m_2 * T_2}{k * c_1 * m_1 + k * c_2 * m_2} \\
T_{mix} &= \frac{c_1 * m_1 * T_1 + c_2 * m_2 * T_2}{c_1 * m_1 + c_2 * m_2}
\end{aligned}$$

Here, k is an arbitrary proportionality constant. Now, this is obviously a very rough sketch without knowing about how the physical system actually interacts. Upon this model, my next iteration would be to apply this to the simulation. The factors that impact the conduction flow are surface area, the temperature of the system and the environment, the thickness of the cup, the mass of the system, and the specific heat of the system. Of them, the surface area, the temperature of the system, the mass of the system, and the specific heat of the system undergoes a change when cream is added.

Making realistic assumptions, the surface area would change by $\frac{V_2}{\pi r^2} * 2\pi r$; the new temperature was determined above; the mass of the system is simply $m_1 + m_2$, and it is possible to deduce the new "specific heat" of the material under how I modeled the internal energy; since $E_{mix} = k * (c_1 * m_1 + c_2 * m_2) * T_{mix}$ and the internal energy is represented as $E = k * c * m * T$, it is reasonable to claim that the specific heat term would now correspond to $(c_1 * m_1 + c_2 * m_2)/(m_1 + m_2)$.

Therefore, in my next iteration, I will update the associated variables as described above at the moment I put in the cream. For the purposes of the model, I will assume that the temperature of the system would immediately settle down to the intermediate temperature, because it would be beyond the scope of this model to take into account the complex convection mechanism within the mixture itself. In determining the optimal moment at which to put the cream in the coffee, I would run the simulation 60 times, over an hour, and have the simulation return the moment at which the coffee temperature fell below the drinkable temperature. I would then take the resultant values and plot them against the moment at which the cream was put in. The punchline graph would roughly look like:

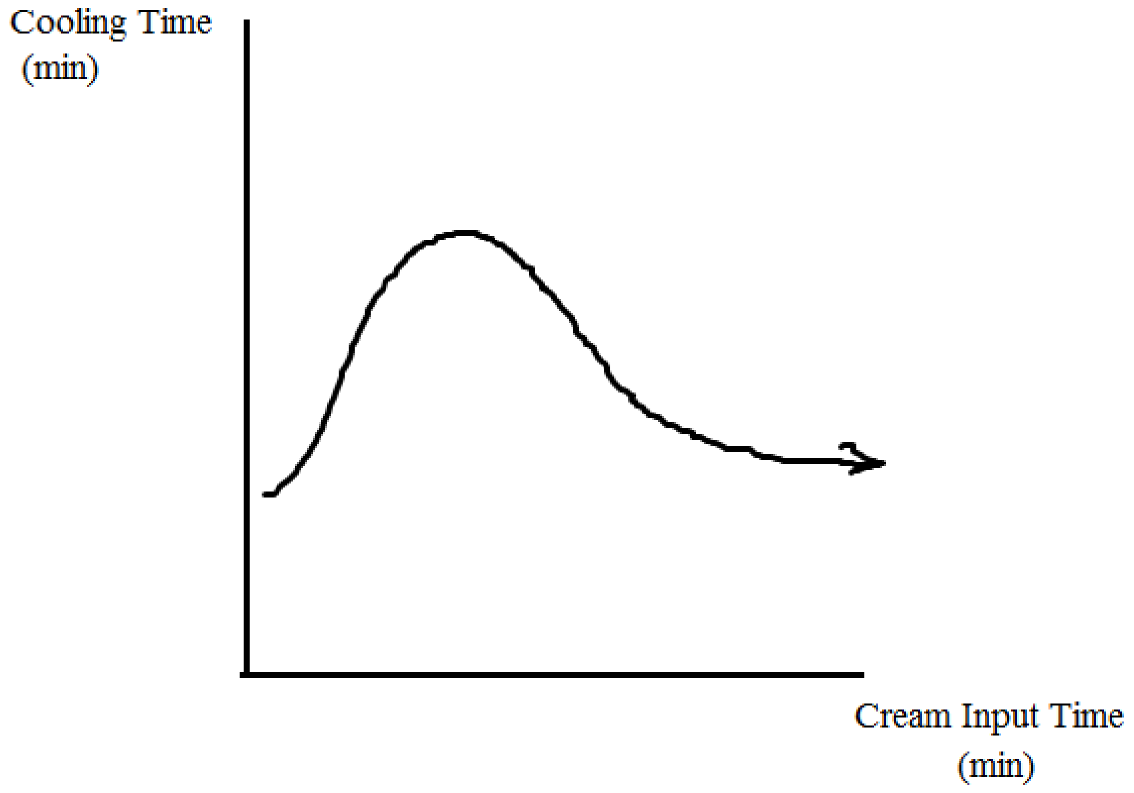


Figure 3: The graph of cream insertion time vs. time it took to cool down to a certain temperature (perhaps 60 degrees Celcius)

ModSim Exercise 6

yoonyoung.cho

October 2015

1 Implementing with ode45

The following is the same model as the previous exercise, now transcribed to an ode45-compatible code;

```
function simulate()



r = 8/2/100; %radius, m
h = 10/100; %height, m
A = pi*r^2+2*pi*r*h; %surface area, m^2
d = 0.7/100;%thickness, m
c = 4186; %specific heat, J/(kg*K)
rho = 1000; %density, kg/m^3
k = 1.5; %heat conductivity, W/(m*K) => (in seconds)
V = pi*r^2*h; %volume, m^3
m = rho*V; %mass, kg

T_s = 370;
T_e = 290;

TbyE = @(E,m,c) E/(m*c);
EbyT = @(T,m,c) T*m*c;

E_s_0 = EbyT(T_s,m,c);

[T,E_s] = ode45(@Q,[0 30],E_s_0);

plot(T,TbyE(E_s,m,c));


function cond = Q(t,E)
cond = -k*60*A*(TbyE(E,m,c)-T_e)/d;
end

end
```

And the resultant plot is shown below:

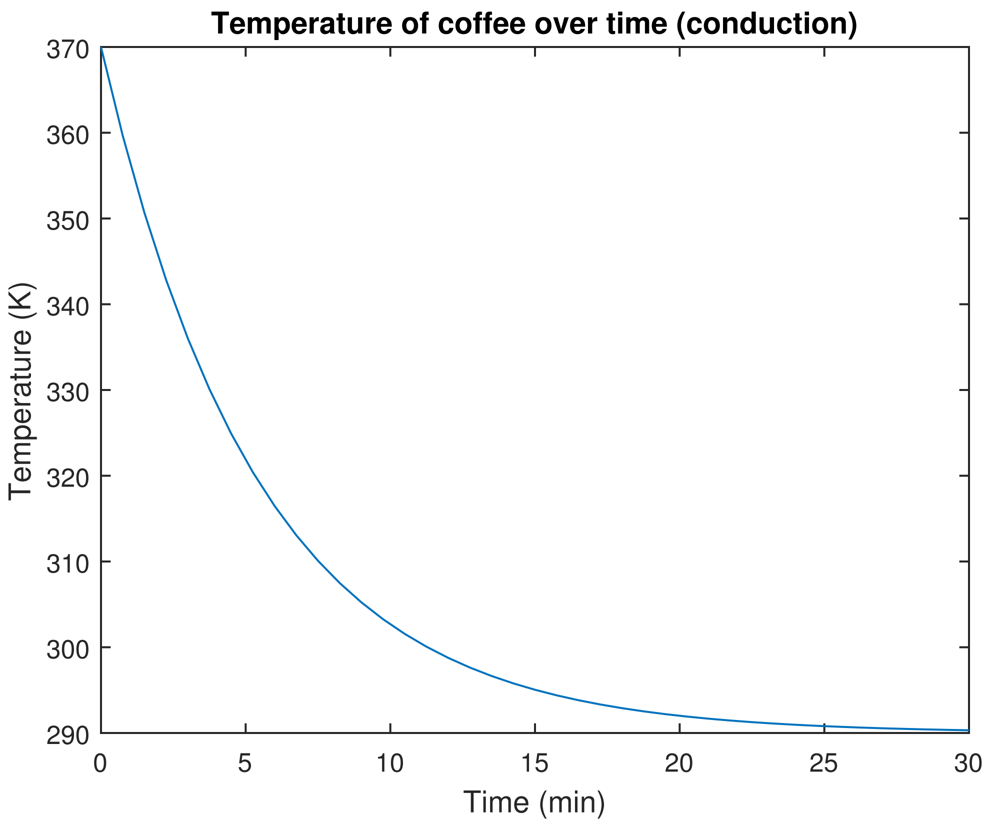


Figure 1: The temperature of coffee over time. In this elementary model, only conduction was taken into account in terms of the cooling of the coffee.

This graph is coherent the previous one, which did not utilize ode45.

2 Adding 100mL cream

The following MATLAB code is a function that takes the time to insert the cream as an argument to produce a plot accordingly.

```
function simulate(addCreamTime)
figure;

r = 8/2/100; %radius, m
h = 10/100; %height, m
A = pi*r^2+2*pi*r*h; %surface area, m^2
d = 0.7/100;%thickness, m
c = 4186; %specific heat, J/(kg*K)
rho = 1000; %density, kg/m^3
V = pi*r^2*h; %volume, m^3
m = rho*V; %mass, kg
k = 1.5; %heat conductivity, W/(m*K) => (in seconds)
T = 370;
T_e = 290;
E_0 = EbyT(T,m,c);

%Q = @(t,E) -k*60*A*(TbyE(E,m,c)-T_e)/d;
```

```
if(addCreamTime>0)
    [t,E] = ode45(@conduction,[0 addCreamTime],E_0);
    plot(t,TbyE(E,m,c));
    [h,A,V,T,m,c,E_0] = addCream(h,r,V,m,c,E(end));
else
    plot(0,TbyE(E_0,m,c),'o');
    [h,A,V,T,m,c,E_0] = addCream(h,r,V,m,c,E_0);
end

hold on;

if(addCreamTime<30*60)
    [t,E] = ode45(@conduction,[addCreamTime 30*60],E_0);
    plot(t,TbyE(E,m,c));
else
    plot(30*60,T,'o');
end
hold on;

line(xlim, [273+60 273+60]);

    function Q = conduction(~,E)
        Q = -k*A*(TbyE(E,m,c)-T_e)/d;
    end

    function [h_new,A_new,V_new,T_new,m_new,c_new,E_new] = addCream(h,r,V,m,c,
       E)
        T = TbyE(E,m,c);

        %CREAM STUFF
        T_c = 275; %K
        V_c = 100/1e6;% m^3
        c_c = 4186; %specific heat, J/(kg*k)
        rho_c = 1000; %density, kg/m^3
        m_c = rho_c*V_c; %kg
        E_c = EbyT(T_c,m_c,c_c); %J

        %END OF CREAM STUFF
        %update constants
        h_new = h + V_c/(pi*r^2);
        A_new = pi*r^2+2*pi*r*h;
        V_new = V + V_c;
        T_new = (E+E_c)/(m*c+m_c*c_c);
        m_new = m + m_c;
        c_new = (c*m+m_c*c_c)/(m_new);
        E_new = E_c+E;
    end
end


function res = TbyE(E,m,c)
res = E/(m*c);
end
```

```

function res = EbyT(T,m,c)
res = T*m*c;
end
```

In the case of adding the cream at the two endpoints (special cases), since ode45 needs to be executed for an interval of which the end must not be the same as the beginning, only one simulation was conducted by adjusting the control flow. In this case, the transition point was marked as a circle, as it couldn't be represented by a line. The resultant plot is shown below, for adding the cream at the beginning, middle, and the end, respectively:

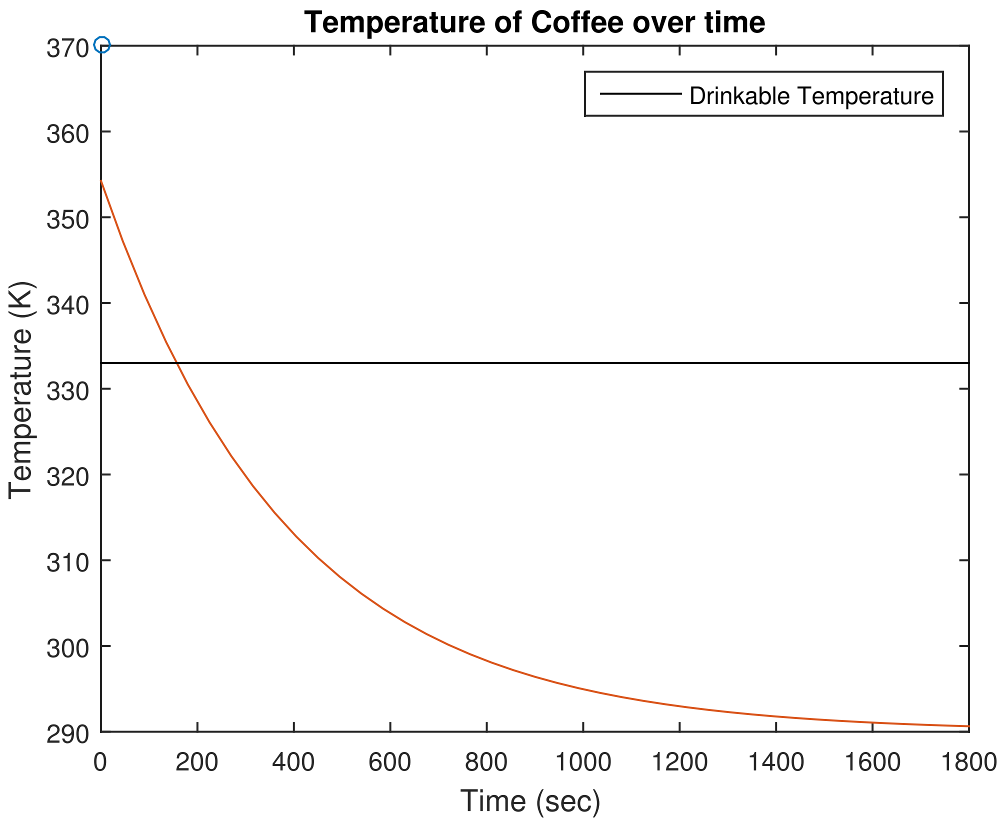


Figure 2: Cream added at t=0

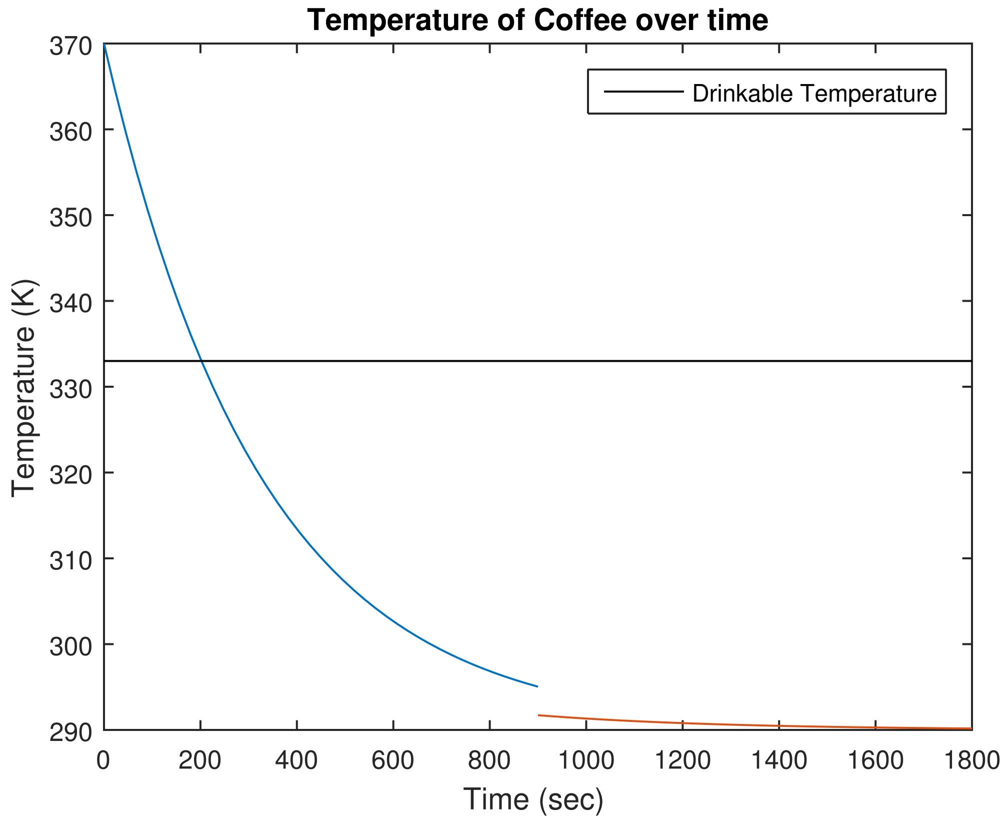


Figure 3: Cream added at t = 15 min

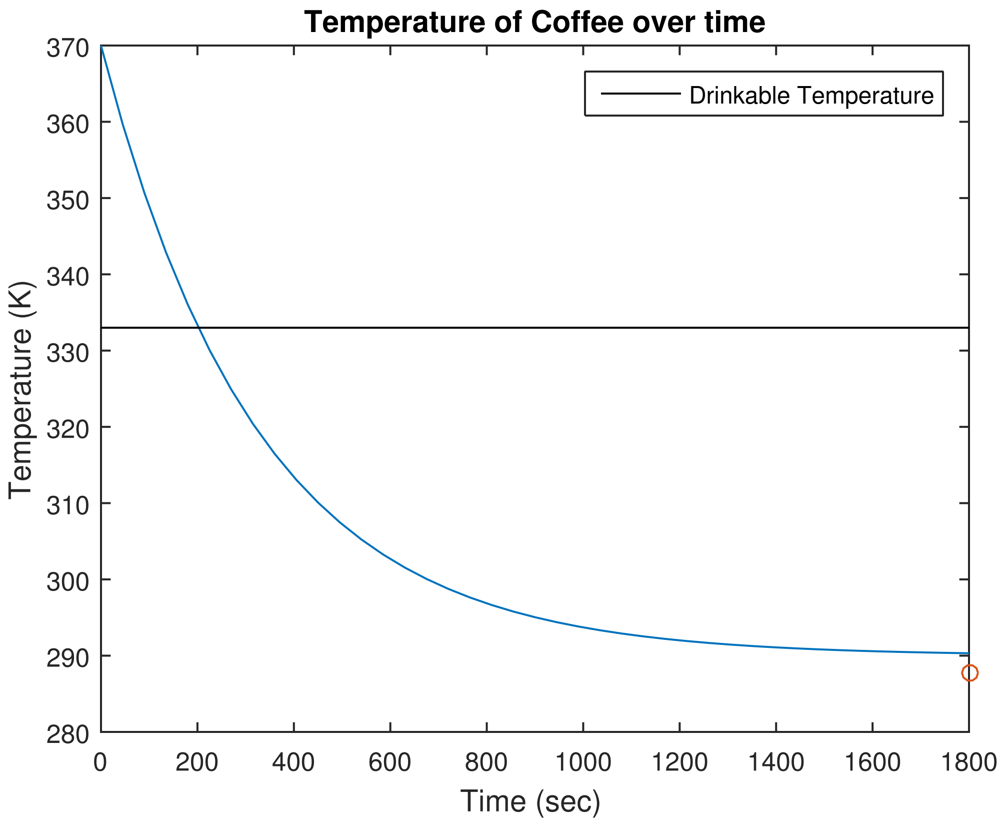


Figure 4: Cream added at t = 30 min

As seen, the falling slope is steepest at the beginning – which is the predicted behavior, as conduction is proportional to the differnce in temperature between the system and the environment. Therefore, it seems most advantageous to procrastinate the adding of the cream until the point at which adding the cream would immediately make the coffee drinkable.

ModSim Exercise 7

Yoonyoung Cho

October 27 2015

Overview

The following MATLAB code oversees the entire assignment:

```
helper.m

%Part I
disp(simulate_4(1*60));
disp(simulate_4(5*60));
disp(simulate_4(10*60));

%Part II
for i = 0:30*60
    ans_vec(i+1) = simulate_4(i);
end
figure;
plot((0:30*60) / 60, ans_vec);
xlabel('Cream Addition Time(min)');
ylabel('Drinkable Time(sec)');
title('Drinkable Time per Cream Addition');

[val,time] = min(ans_vec);
disp(time);
%Part III
disp(fminsearch(@simulate_4,5.0*60));
```

Simulate_4 is a function that takes cream-addition time as an argument and returns the time at which the drinkable temperature is reached for the coffee.

In Part I, three values are investigated : the drinkable time as cream is added 1 minutes, 5 minutes, and 10 minutes after the simulation, respectively.

In Part II, the punchline plot is generated. This is done by measuring the drinkable time output from the cream-addition time (input) – ranging from 0 minutes(immediately) to 30 minutes – and then juxtaposing the result with the input time.

In Part III, the same simulation function undergoes the MATLAB function fminsearch in order to identify the exact optimal time at which to add the cream, as well as verifying the plot obtained from Part II.

1 Restructuring

Without further ado, simulate_4.m will be presented:

```
function drinkTime = simulate_4(addCreamTime)

%initial values
r = 8/2/100; %radius, m
```

```
h = 10/100; %height, m
A = pi*r^2+2*pi*r*h; %surface area, m^2
d = 0.7/100;%thickness, m
c = 4186; %specific heat, J/(kg*K)
rho = 1000; %density, kg/m^3
V = pi*r^2*h; %volume, m^3
m = rho*V; %mass, kg
k = 1.5; %heat conductivity, W/(m*K) => (in seconds)
T = 370;
T_e = 290;
E_0 = EbyT(T,m,c);

if(addCreamTime>0)
     zeroCross = @(T,E) zCross(T,E,m,c);%Event, reaching drinkable temperature.
 %passes additional parameters m,c (mass & specific heat) to compute the
 %temperature from energy.
    opt = odeset('Events',zeroCross);
    [t,E,alpha,beta,gamma] = ode45(@conduction,[0 addCreamTime],E_0,opt);

   if(length(alpha) ~= 0) % = there is a solution
        drinkTime = alpha;
        return;
   end

    [h,A,V,T,m,c,E_0] = addCream(h,r,V,m,c,E(end)); %update constants' values
    if(TbyE(E_0,m,c) < 320)% = adding cream resulted in drinkable mixture
        drinkTime = addCreamTime;
        return;
    end
else
     [h,A,V,T,m,c,E_0] = addCream(h,r,V,m,c,E_0); %update constants' values
    if(TbyE(E_0,m,c) < 320)% = adding cream resulted in drinkable mixture
        drinkTime = addCreamTime;
        return;
    end
end

%COFFEE Temperature simulation after adding Cream

 zeroCross = @(T,E) zCross(T,E,m,c); %Event, reaching drinkable temperature.
 %passes additional parameters m,c (mass & specific heat) to compute the
 %temperature from energy.
    opt = odeset('Events',zeroCross);
    [t,E,alpha,beta,gamma] = ode45(@conduction,[addCreamTime 30*60],E_0,opt);
    drinkTime = alpha; %assumption : the drinkable Time will be reached before 30 min passes.
     return;

%conduction flow
    function Q = conduction(~,E)
        Q = -k*A*(TbyE(E,m,c)-T_e)/d;
    end

```

```
%simulating addition of cream
    function [h_new,A_new,V_new,T_new,m_new,c_new,E_new] = addCream(h,r,V,m,c,
       E)
        T = TbyE(E,m,c);

        %CREAM STUFF
        T_c = 275; %K
        V_c = 80/1e6;% m^3
        c_c = 4186; %specific heat, J/(kg*k)
        rho_c = 1000; %density, kg/m^3
        m_c = rho_c*V_c; %kg
        E_c = EbyT(T_c,m_c,c_c); %J

        %END OF CREAM STUFF
        %update constants
        h_new = h + V_c/(pi*r^2);
        A_new = pi*r^2+2*pi*r*h;
        V_new = V + V_c;
        T_new = (E+E_c)/(m*c+m_c*c_c);
        m_new = m + m_c;
        c_new = (c*m+m_c*c_c)/(m_new);
        E_new = E_c+E;
    end
end


function res = TbyE(E,m,c)
res = E/(m*c);
end

function res = EbyT(T,m,c)
res = T*m*c;
end

function [v,isT,dir] = zCross(t,E,m,c)
v = TbyE(E,m,c)-320;
isT = 1;
dir = 0;
end
```

The script is written such that no global variables, other than immutable constants, were used; the functions are encapsulated, and is coordinated in terms of logical association. For instance, the flow function is a part of the simulation – therefore, it was nested.

The simulation itself is composed of three phases : before, during, and after adding the cream. At any time, if the coffee temperature crosses the 320K (drinkable temperature) border, the function terminates and returns the time at which the event occurred.

The output values, for the investigation of discrete values, are as follows:

Table 1: Drinkable Time Investigation Table

$CreamAddition(min)$	$Drinkable(sec)$
1	291.0789
5	300
10	319.3362

2 Punchline Graph

It can be easily seen that the function above can be applied to any point in time – in whole units of seconds – at which to add the cream. Applying this to the interval from 0 to 30 minutes, the punchline graph below was generated:

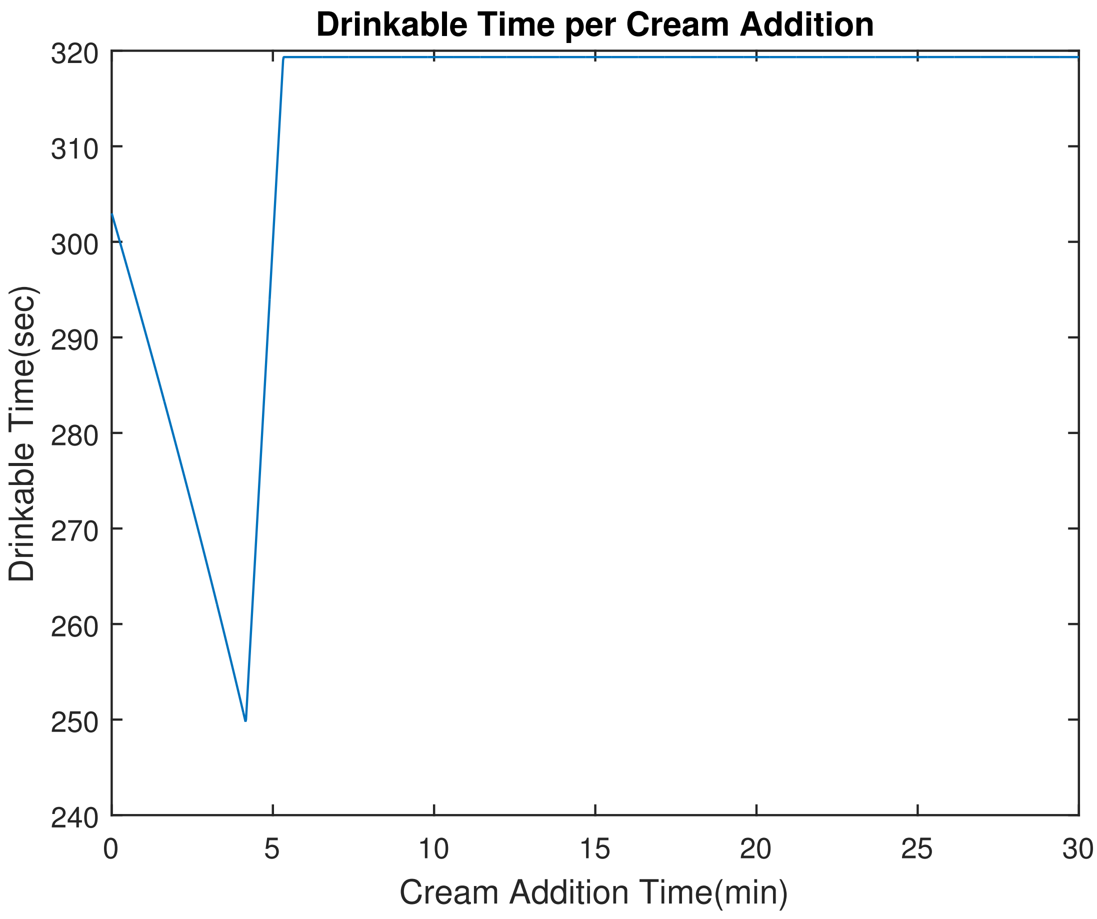


Figure 1: Part II Result: the plot of the time at which the coffee reached drinkable temperature (set to be 320 K) with respect to the time at which cream was inserted in the coffee.

The implications of this plots are consistent with intuition. It would be disadvantageous to insert the cream prematurely, as the heat flow is dependent on the difference between the temperature of the environment and the temperature of the system. Adding the cream would lower the temperature of the system, rendering it less efficient in terms of emitting energy to the environment.

However, it would be equally disadvantageous to stall for too long in adding the cream, as the coffee may simply cool down below the drinkable temperature without the aid of the cream – which would, of course, take longer than optimal. It is thus evident that the ideal scenario would be to insert the cream – such that at the precise moment, the coffee would cool down to drinkable temperature as a result thereof.

The natural question, of course, is when that point is. By utilizing the MATLAB function min, which yields the index of the array at which the minimum value occurs (as its optional secondary return value), it was found that optimal value is 250 seconds (in resolutions of unit seconds, as the simulation only runs in seconds intervals).

3 Optimal Cream-Adding Time

The result above may be sufficient, but it is dissatisfying – the resolutions are limited in unit seconds – of course, this can be amended, but the alterations in the order of precision in the simulation would correspondingly incur a significant cost in time. Fortunately, I can use fminsearch in this case, with the starting value at 5 minutes, which is approximately where the plot reaches its trough.[1] The resultant output was 249.6337 seconds, which is definingly close to the simulated value. This therefore legitimizes the simulation.

[1]for reminders, the function was run as: $disp(fminsearch(@simulate\_4, 5.0 * 60))$.

ModSim Exercise 9

Yoonyoung Cho

November 9 2015

1 Model

The baseball is modelled as a particle, since the rotation is relatively trivial for the purposes of the investigation; modelling baseball as a rigid body involves more complex mechanics – namely aerodynamics, Magnus effects, etc. – which are beyond the scope of the question, as the primarily concern of the question is translational displacement.

2 Interactions

The forces that would act upon the ball are:

Gravity[1], $\vec{F_g} = -\frac{Gm1m2}{\vec{r}^2}\hat{\mathbf{r}}$

Drag[2], $\vec{F_d} = -1/2\rho C_d A\vec{v}^2\hat{\mathbf{v}}$

3 Free Body Diagram

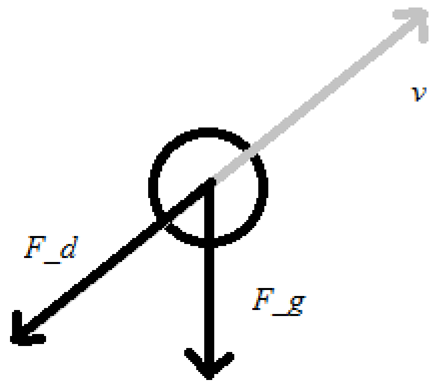


Figure 1: Free body diagram of the baseball.

In the diagram, F_d is Drag, F_g is Gravity, and v is the velocity(notated in grey since it does not contribute to net force).

4 Mathematical Abstraction

The ball was modeled in a 2D coordinate system, where sideways movement was ignored; in this manner, the x-axis was aligned parallel to the ground; likewise, the y-axis was normal to the ground. The mathematical

[1]Primary Factor

[2]Secondary Factor

relationships are as follows:

$$
\begin{aligned}
\frac{d\vec{r}}{dt} &= \vec{v} \\
\frac{d\vec{v}}{dt} &= \frac{\vec{F}}{m_b} \\
\vec{F} &= \vec{F_g}(\vec{r}) + \vec{F_d}(\vec{v}) \\
\vec{F_g} &= -\frac{Gm_e m_b}{\vec{r}^2}\hat{\mathbf{r}} \\
\vec{F_d} &= -1/2\rho C_d A\vec{v}^2\hat{\mathbf{v}}
\end{aligned}
$$

As for initial conditions, at time $t = 0$, the ball would be some height above the origin (to accurately reflect baseball scenario), but its x-value would be at the reference(0); the velocity would depend on the occasion.

5 Explanation of Terms

Term	Description	Value	Unit
$\vec{r}$	Position of baseball	$< 0, 5 >$	m
$\vec{v}$	Velocity of baseball	$< 39, 12 >$	m/s
G	Gravitational Constant	$6.67408e-11$	$m^3kg^{-1}s^{-2}$
m_b	Mass of Baseball	.145	kg
m_e	Mass of Earth	$5.9742e24$	kg
A	Cross-sectional area of baseball	.004	m^2
$\vec{F_g}$	Gravity	$< 0, -1.42 >$	N
$\vec{F_d}$	Drag Force	$< -0.6, -0.09 >$	N
ρ	Density of Air	1.225	kg/m^3
C_d	Drag Coefficient of Baseball	0.3	-

Table 1: The terms in the differential equation; Vectors were notated element-wise. Constant values are taken verbatim; variable values are samples within reasonable range.

ModSim Exercise 10

Yoonyoung Cho

November 17 2015

1 Key Frames

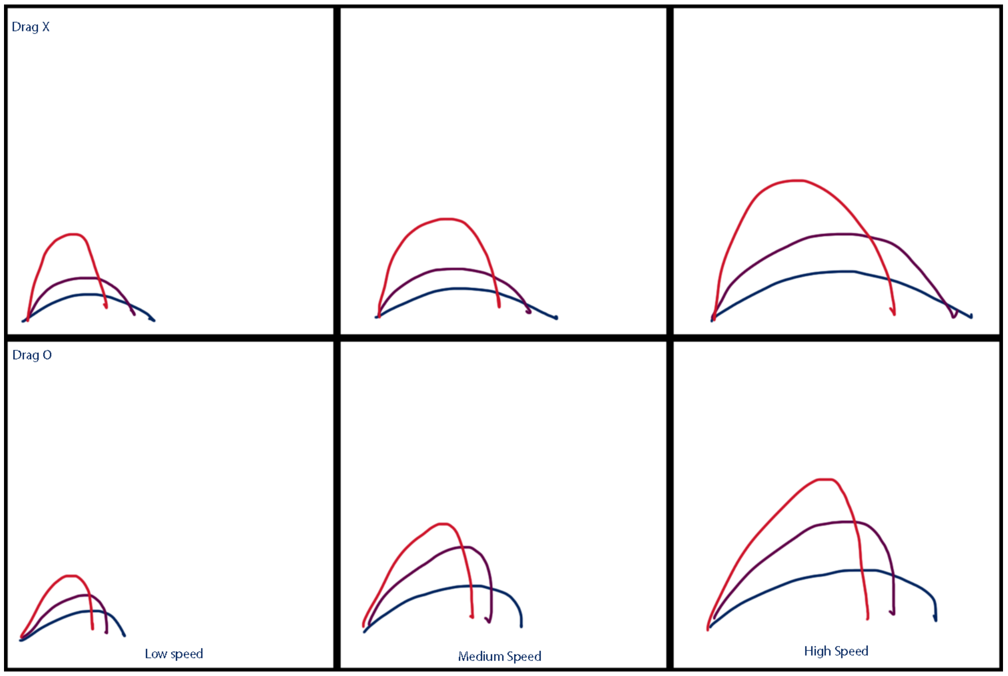


Figure 1: Key Frames of the mechanics involving the flight of the baseball; the graph depicts the trajectory of the baseball. The plot is color-coded based on the launch angle: red, purple, and blue corresponding to high, middle, and low angles, respectively.

In the key frames, I sought to capture the behavior that the distance covered was dependent on the magnitude of the initial velocity, as well as the launch angle; too high of a launch angle would expend much of the momentum in the upwards direction[1]. Moreover, I exaggerated the rightward skewing of the overall trajectory in the presence of drag, as a result of dissipation in the horizontal momentum due to drag.

2 Estimation

Based on the research parameters I had identified in the last exercise(replicated in Table 1), I calculated the initial drag and gravitational force for the starting velocity of 54 m/s

[1]The graph is, of course, faulty: later, I realized that the the angle couldn't be too low either, as the ball would fall to the ground prematurely.

Term	Description	Value	Unit
$\vec{r}$	Initial Position of baseball	$< 0, 1 >$	m
$\vec{v}$	Initial Velocity of baseball	$< 38, 38 >$	m/s
G	Gravitational Constant	$6.67408e-11$	$m^3kg^{-1}s^{-2}$
m_b	Mass of Baseball	.145	kg
m_e	Mass of Earth	$5.9742e24$	kg
A	Cross-sectional area of baseball	.004	m^2
$\vec{F_g}$	Gravity	$< 0, -1.42 >$	N
$\vec{F_d}$	Drag Force	$< -0.6, -0.09 >$	N
ρ	Density of Air	1.225	kg/m^3
C_d	Drag Coefficient of Baseball	0.3	-

Table 1: Constants.

The forces that describe the system are:

$$\vec{F} = \vec{F_g}(\vec{r}) + \vec{F_d}(\vec{v})$$
$$\vec{F_g} = -\frac{Gm_em_b}{\vec{r}^2}\hat{\mathbf{r}}$$
$$\vec{F_d} = -1/2\rho C_d A\vec{v}^2\hat{\mathbf{v}}$$

The above interaction was implemented in MATLAB as follows:

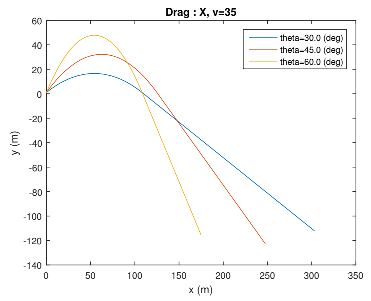
```
function res = Gravity(h_b)%gets height of ball
G = 6.67384e-11; %m^3 / kg*s^@ Gravity Constant
m_e = 5.9742e24; % kg, mass of earth
m_b = 145e-3; % kg, mass of baseball
r_e = 6378e3; %m, radius of earth
r = h_b + r_e; % h + r_e
res = [0 -G*m_e*m_b/r^2];
end

function res = Drag(v_x,v_y)
r_b = 37e-3; %m, radius of ball
rho = 1.225; %kg/m^3, density of air
C_d = 0.3; %drag coefficient of basesball
A = pi*r_b^2;
speed = sqrt(v_x^2+v_y^2);

nv_x = v_x / speed; %normalize
nv_y = v_y / speed;
res = -1/2*rho*C_d*A*speed^2*[nv_x nv_y];
end
```

which yielded Drag = $2.304455N$ and Gravity = $1.421198N$[2]. Based on this comparison, it was clear that Drag force could not be disregarded from the calculation.

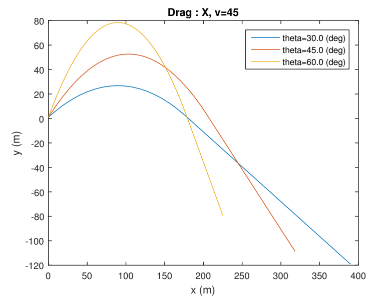
3 Basic Implementation

To avoid redundancies, the final code will be presented in the next section. Here, only the results of the basic implemention – without the later added constraints – will be shown:

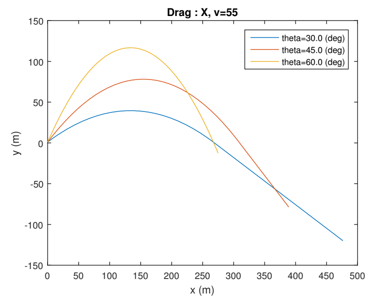
[2]since it is known that the height of the ball does not influence the magnitude of gravity very much, I simply chose 0 as the height in the above code.



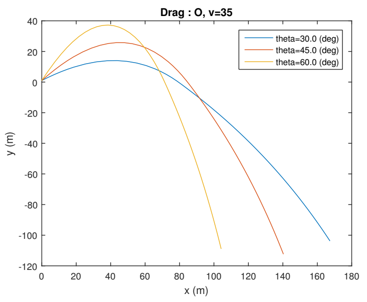
(a) 35m/s without Drag



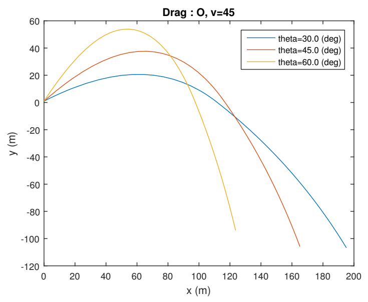
(b) 45m/s without Drag



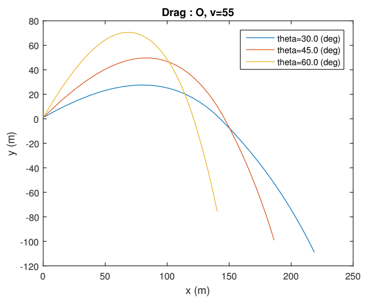
(c) 55m/s without Drag



(d) 35m/s with Drag



(e) 45m/s with Drag



(f) 55m/s with Drag

Figure 2: Matlab-generated keyframes

It can be seen that without the constraints that limit the ball above the ground (quite naturally) the ball goes farthest when shot in a lower angle, in a more horizontal orientation. As this does not accurately reflect what happens – and we are not currently equipped with the capacity to model the bouncing of the ball, a restriction will be enforced that limits the ball's location to above the ground, which leads to the next section:

4 Event Detection

The MATLAB implementation of the simulation is as follows:

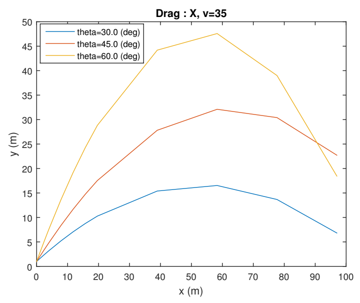
```
function res = simulateBaseball(speed,angle,useDrag,plotRes)
r_x =0;
r_y =1;%initial positions, m

v_x =speed*cos(angle);
v_y =speed*sin(angle);%initial velocity, m/s

y = [r_x r_y v_x v_y];
y = y';
opt = odeset('Events',@hitGround); %When it hits ground (r_y <=0)
netAccelDrag = @(t,y) netAccel(t,y,useDrag);
[t,Y] = ode45(netAccelDrag, [0 10],y, opt);

%PART 3
if(plotRes)
        plot(Y(:,1),Y(:,2));
        xlabel('x (m)');
        ylabel('y (m)');
end
```

```
res_raw = Y(:,2);
res = res_raw(end);
end
function res = netAccel(~,y,useDrag)
m_b = 145e-3; % kg, mass of baseball

dv = netForce(y,useDrag)/m_b;
res(1) = y(3); %dx
res(2) = y(4); %dy
res(3) = dv(1); %dvx
res(4) = dv(2); %dvy
res = res';
end

function res = netForce(y,useDrag) %returns [f_x f_y]
r_x = y(1);
r_y = y(2);
v_x = y(3);
v_y = y(4);
if(useDrag)
        res = Gravity(r_y) + Drag(v_x,v_y);
else
        res = Gravity(r_y);
end
        if(r_y <= 0)
                res(2) = 0; %downward force = 0
        end

end

function res = Gravity(h_b)%gets height of ball
G = 6.67384e-11; %m^3 / kg*s^@ Gravity Constant
m_e = 5.9742e24; % kg, mass of earth
m_b = 145e-3; % kg, mass of baseball
r_e = 6378e3; %m, radius of earth
r = h_b + r_e; % h + r_e
res = [0 -G*m_e*m_b/r^2];
end

function res = Drag(v_x,v_y)%force acting against velocity
r_b = 37e-3; %m, radius of ball
rho = 1.225; %kg/m^3, density of air
C_d = 0.3; %drag coefficient of basesball
A = pi*r_b^2;
speed = sqrt(v_x^2+v_y^2);

nv_x = v_x / speed; %normalize
nv_y = v_y / speed;
res = -1/2*rho*C_d*A*speed^2*[nv_x nv_y];
end

function [val,term,dir] = hitGround(t,y) %Event Condition to stop
%named hitground but also accomodates hitting the wall
val(1) = y(2); %y(2) = r_y = height
```

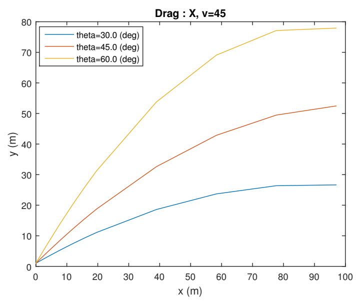
```
term(1) = 0;
dir(1) = 0;

val(2) = y(1) - 97; %second event : hit wall
term(2) = 0;
dir(2) = 0;
end
```

The function takes initial speed and angle as a parameter, as well as flags such as a whether or not to incorportate drag force into the system, and whether or not to plot the result.

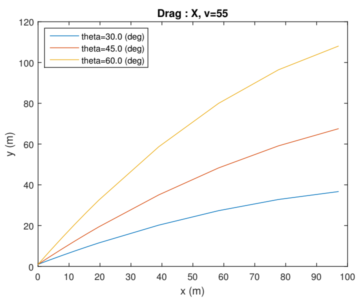
and the resultant plot:



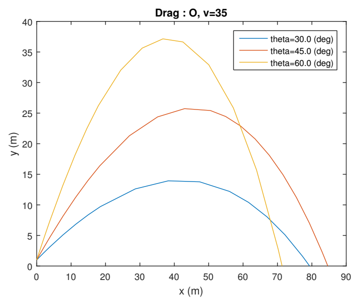
(a) 35m/s without Drag



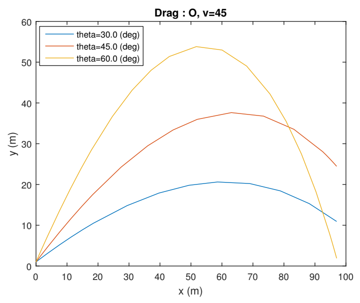
(b) 45m/s without Drag



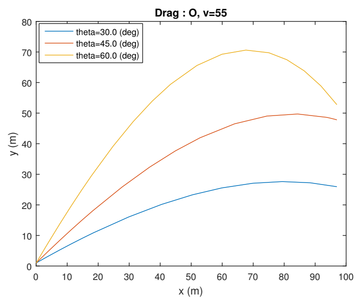
(c) 55m/s without Drag



(d) 35m/s with Drag



(e) 45m/s with Drag



(f) 55m/s with Drag

Figure 3: Matlab-generated keyframes, with events.

It is seen that, with enforced constraints, an angle neither too high nor too low performs best, as it has enough time in air to travel, as well enough horizontal velocity to travel far; there is evidence of the rightward skewing – as well as greatly reduced distance – in the presence of the drag force, though a bit more subtle than that hand-drawn in the keyframes.

θ(deg) / v (m/s)	30	45	60
35	0	0	0
45	10.93	24.45	1.89
55	25.94	47.80	52.76

Table 2: Final Height of the ball based on the initial launch state.

5 Range-based Analysis

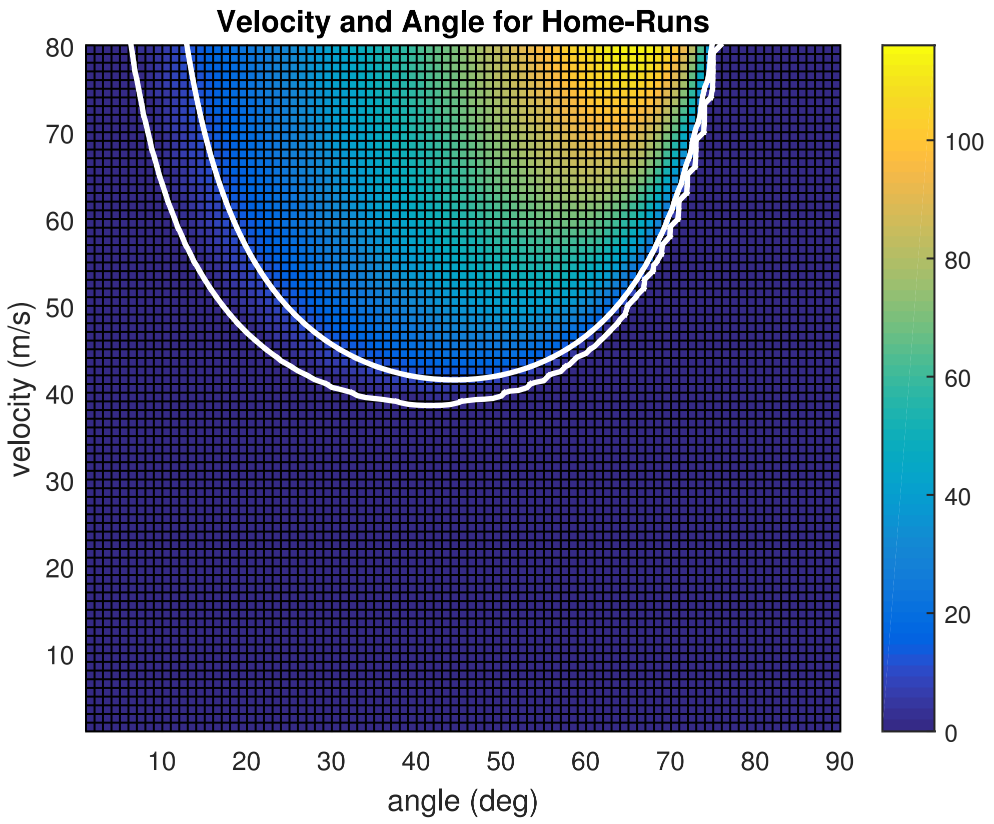


Figure 4: The final height of the ball, given velocity and angle as initial states.

From the graph, it is apparent that the minimum velocity to clear the green monster is around $40m/s$, at approximately 40 degrees above the horizontal.

ModSim Exercise 11

Yoonyoung Cho

December 1 2015

1 Gravity

$$\vec{r} = \vec{p_1} - \vec{p_2}$$
$$\vec{F_g} = -\frac{G m_e m_b}{\vec{r}^2}\hat{\mathbf{r}}$$

With the above relation, the implementation is trivial:

```
function res = Gravity(m1,p1,m2,p2)%corresponds to gravity_force_func
r = (p1-p2); %direction : from p2 to p1
G = 6.67384e-11; %m^3 / kg*s^@ Gravity Constant
f_g = G*m1*m2/norm(r)^2; %magnitude
res = f_g*r/norm(r); %force vector
end
```

2 Orbital Simulation

```
function [t,y] = simulateOrbit()
%position of the Sun is taken as origin
p_s = [0 0 0]; %m
%778e9 = distance from the Sun to Jupiter
p_j = [0 778e9 0]; %m
y0 = packParams(p_s',[0 0 0]', p_j', [13.06e3 0 0]');
[t,y] = ode45(@orbitFlow,0:86400:86400*4332,y0);

[~,~,p_j,~] = unpackParams(y(end,:));
%Verification
disp(p_j); %position of Jupiter after 4332 days
theta = atan2(y(:,8),y(:,7)); %angle
figure;
plot(t,theta);
hold on;
line(xlim,[theta(end) theta(end)],'Color','r');
xlabel('time (sec)');
ylabel('angle (rad)');
title('Orbital Angle of Jupiter');

figure;
if(nargout == 0)
                %comet(y(:,7),y(:,8));
                myComet(t,y(:,1),y(:,2),y(:,7),y(:,8));
```

```
        end
end

function res = packParams(p_s,v_s,p_j,v_j)
%format into column vector
res = [p_s v_s p_j v_j];
res = reshape(res,12,1);
end

function [p_s,v_s,p_j,v_j] = unpackParams(y)
%reassign column vector to 4 vectors
y = reshape(y,3,4);
p_s = y(:,1);
v_s = y(:,2);
p_j = y(:,3);
v_j = y(:,4);
end

function res = orbitFlow(t, y)
[p_s,v_s,p_j,v_j] = unpackParams(y);
%p_s = Sun Position, m
%v_s = Sun Velocity, m/s
%p_j = Jupiter Position, m
%v_j = Jupiter Velocity, m/s
m_s = 2.0e30; %Sun Mass, kg
m_j = 1.89e27;%Jupiter Mass, kg
f_g = Gravity(m_s,p_s,m_j,p_j); %dir = J->S
dv_s = -f_g/m_s;%opposite pull
dv_j = f_g/m_j;
dp_s = v_s;
dp_j = v_j;
res = packParams(dp_s,dv_s,dp_j,dv_j);
end
```

According to the simulation, the position of Jupiter after 4332 days was found to be $\vec{p_j} =< 8.64e10, 7.70e11, 0 >$; here, the discrepancy in the x value, which indicates the position of Jupiter went slightly over the beginning point, can be attributed to the assumption that the orbit of Jupiter is perfectly circular, in which case the mean velocity of Jupiter may not have been the harmonic value for the circular orbit.

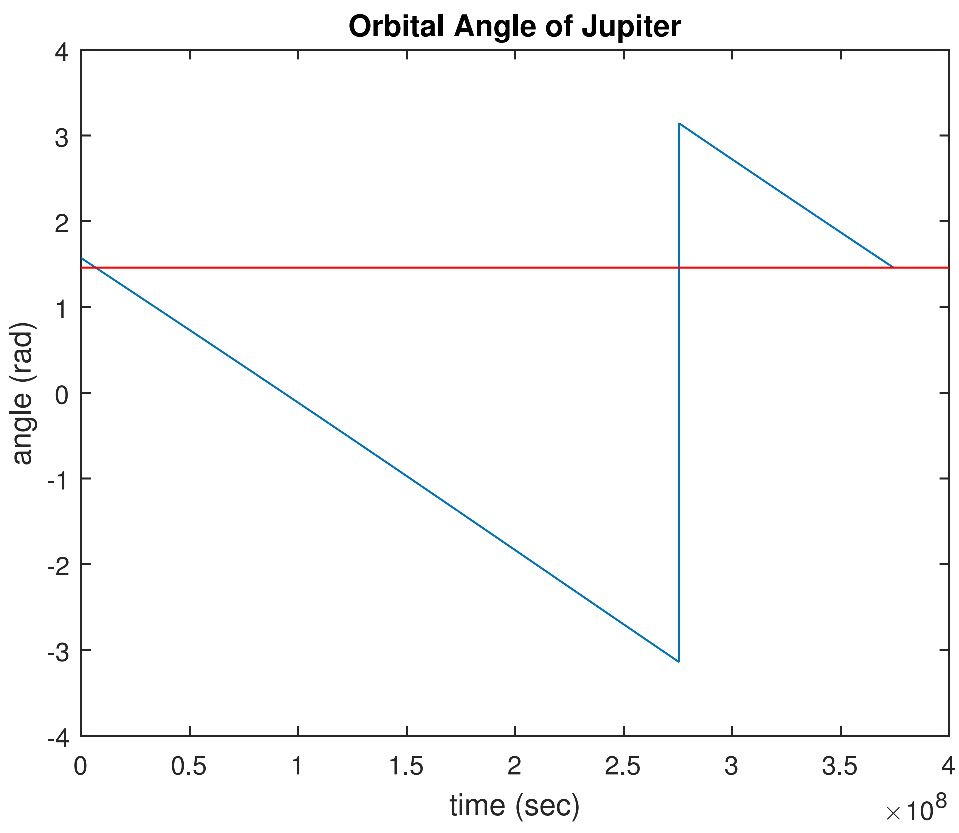


Figure 1: The Angle of Jupiter. The Sudden jump in the middle is the transition from $-\pi$ to π, which are equivalent.

Again, it is apparent that, in the course of 4332 days, Jupiter completes a cycle and exceeds the beginning angle by a bit.

3 Animation

```
function myComet(t,x_s,y_s,x_j,y_j)
% t = real orbit time(in sec)
        hold on;
        %scale duration to .001 sec/day
        duration = (max(t)-min(t)) / 86400 / 1000; %sec
        fps = 40; %frame/sec
        stepSize = floor(length(x_j)/(duration*fps));
        minmax = [min(x_j),max(x_j),min(y_j),max(y_j)]; %frame dimensions
        for i=1:stepSize:length(x_j) % # frames
                pause(1/fps);
                axis(minmax);
                plot(x_j(i),y_j(i),'r.','MarkerSize',20);
                plot(x_s(i),y_s(i),'y.','MarkerSize',50);
                drawnow;
        end
        xlabel('x (m)');
        ylabel('y (m)');
        title('Orbit of Jupiter Around the Sun');
```

```
end
```

Here, the pause duration was chosen such that the duration of the animation would be scaled to $.001 sec/day$:

$$
\begin{aligned}
dur &= time/1000/(60*60*24) \\
step &= \frac{length(x_j)}{dur*fps} \\
numFrame &= \frac{length(x_j)}{step}
\end{aligned}
$$

$$
\begin{aligned}
t &= \frac{1}{fps} * numFrame \\
&= \frac{1}{fps} * \frac{length(x_j)}{\frac{length(x_j)}{dur*fps}} \\
&= \frac{1}{fps} * dur * fps \\
&= dur
\end{aligned}
$$

A frame from the resultant animation is as follows:

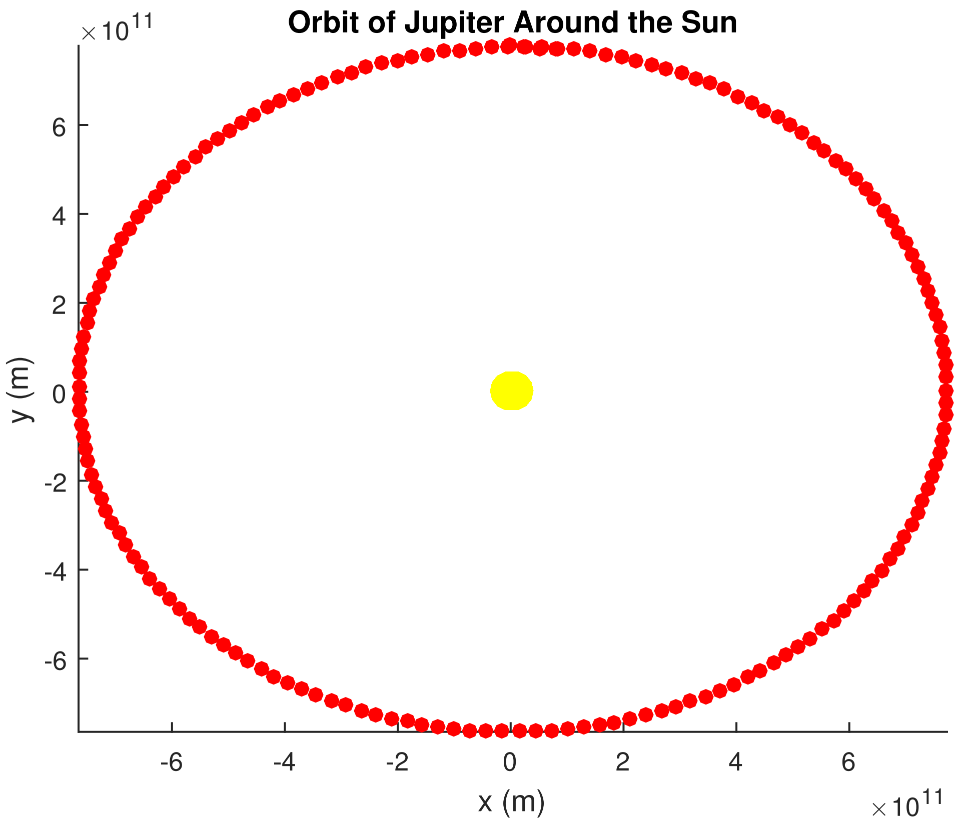


Figure 2: The orbit of Jupiter around the sun, after a cycle.