**Date: 5/12/2015**

River basin, demand for livestock

Electricity

Demand, Availability, Scarcity

Disease model – due to climate change

OpenStreetMap

WebGL

Time to calculate SIR based on the pixel rendering – feasibility analysis

SIR – Sick Infected and Recover Model

MATLAB to WebGl or d3

10\*10 sized grid

**Date: 5/15/2015**

SIR :

<http://en.wikipedia.org/wiki/Epidemic_model>

<http://www.maa.org/publications/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model>

<http://en.wikipedia.org/wiki/Kermack%E2%80%93McKendrick_theory>

Visualization:

<http://www.fastcoexist.com/3023416/outbreak-watch-how-quickly-an-epidemic-would-spread-across-the-world>

<http://digg.com/video/visualizing-disease-outbreaks-that-could-have-been-prevented-around-the-world>

Line chart : <http://bl.ocks.org/mbostock/3883245>

**Disease modeling Week 1 Report**

Visualization for multiple cities on the same graph plot was completed today.

**Week 1 Task: Migrating MATLAB code to d3 visualization of Disease model.**

1. Completed for multiple cities plot on the same graph

2. User can input number of cities and get corresponding plot of Infected humans vs Time(in years).

3. All the values are presently hard coded except the initial state of parameters for cities.

4. Runge Kutta method run over similar number of iterations in MATLAB and D3 (~73000)

**Time comparison study for 3 cities model evaluation:**

MATLAB: 6-12 secs

D3: 2.5-3.5 secs.

**Next week planned task:**

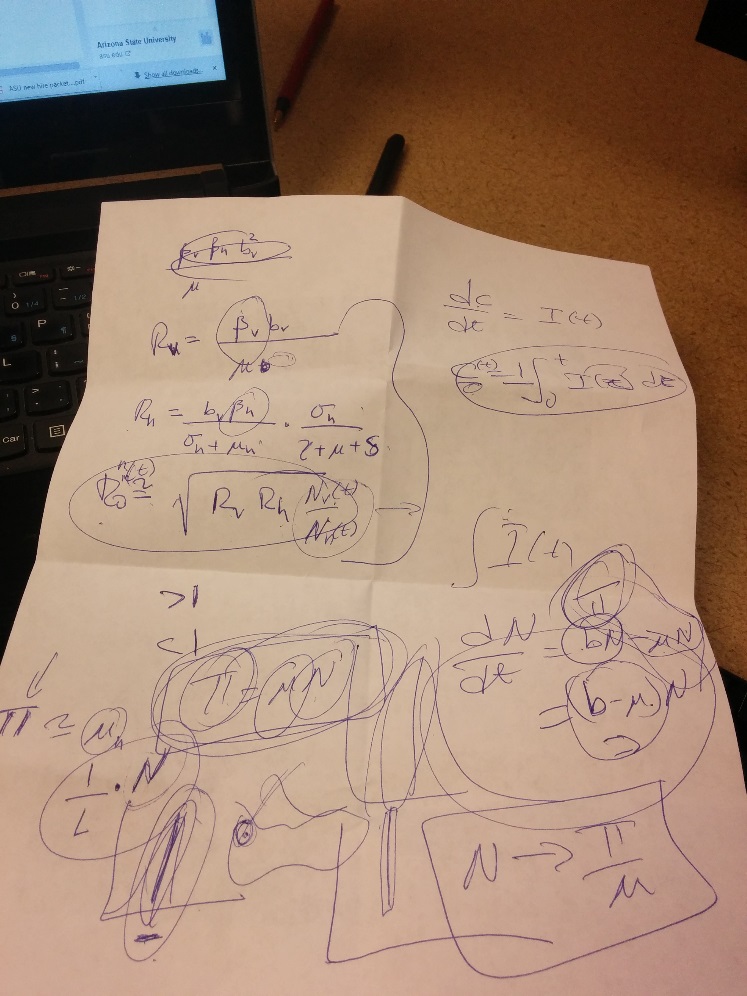
1. Validation of plots

2. Code cleanup and optimization

3. User interface for entries on parameters

All the code and references are shared at this location: <https://drive.google.com/open?id=0B91uUYlaIEDtUllOOE1LNUVibEk&authuser=0>

## **Week 2**

5/26 Tuesday: Meeting with Alhaji.

1. How to validate the plot:
   1. Plot looks fine. It can be further validated by checking if the disease is exponentially spreading or dying out.
   2. Use of actual values by referring range in paper.
2. How to make program run faster
   1. Parallel computing: to be taken up this week as task to find out ways to evaluate numerical computations in parallel.
3. Memory
   1. Is preserving of all data values for all time range needed? For now it seems important.
   2. Data can be stored in a file and fetched in case of high storage requirements.
4. Input
   1. All independent variable needs to be inserted into a file for reading input. To be taken up this week.

5/27 Wednesday

1. 3000 vs 73000, step size?

2. how to validate the graph?

parallel computing

matlab has it

peak and peaktime

cumulative/ integral - number of people getting infected in one cituy for entire time

average

tomorrow's task

understand runge kutta method

create a sample file, read values from sample file. - done

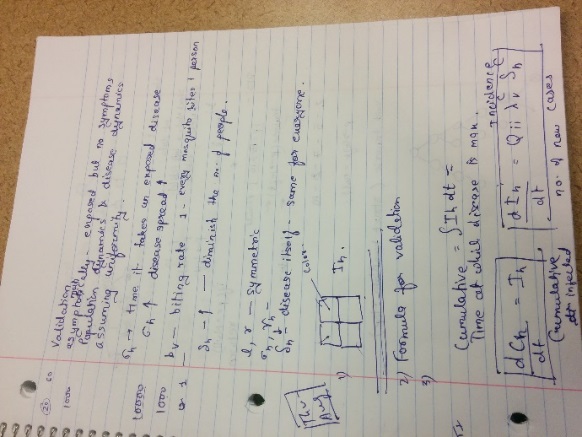
server vs javascript!

Legend reference:

view-source:http://output.jsbin.com/ubafur/3

**Week 3**

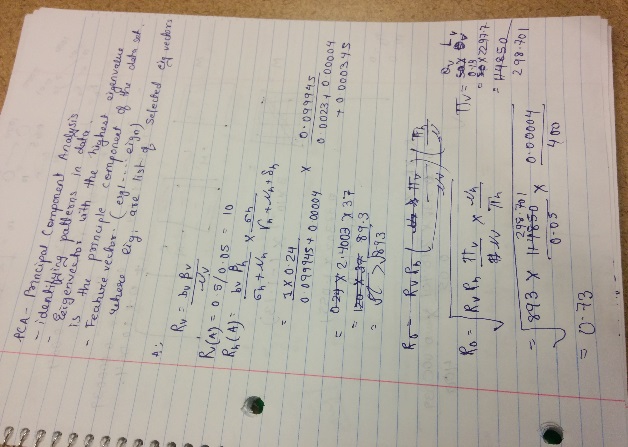
6/11

Completed parallelization. Code uses multithreaded implementation to calculate parameter values for each city. For 5 cities time difference – Parallel: 0.541 s Non-parallel: 2.477s. Need to check capacity of the system.

To validate the trend that the disease follow, the formula given in the picture is used, if >1 the disease becomes epidemic otherwise it dies out.

Next task: Check on 10 sample values to see if the plot goes on correct direction. Use the relations in second figure to check trends. - Done. Not correct

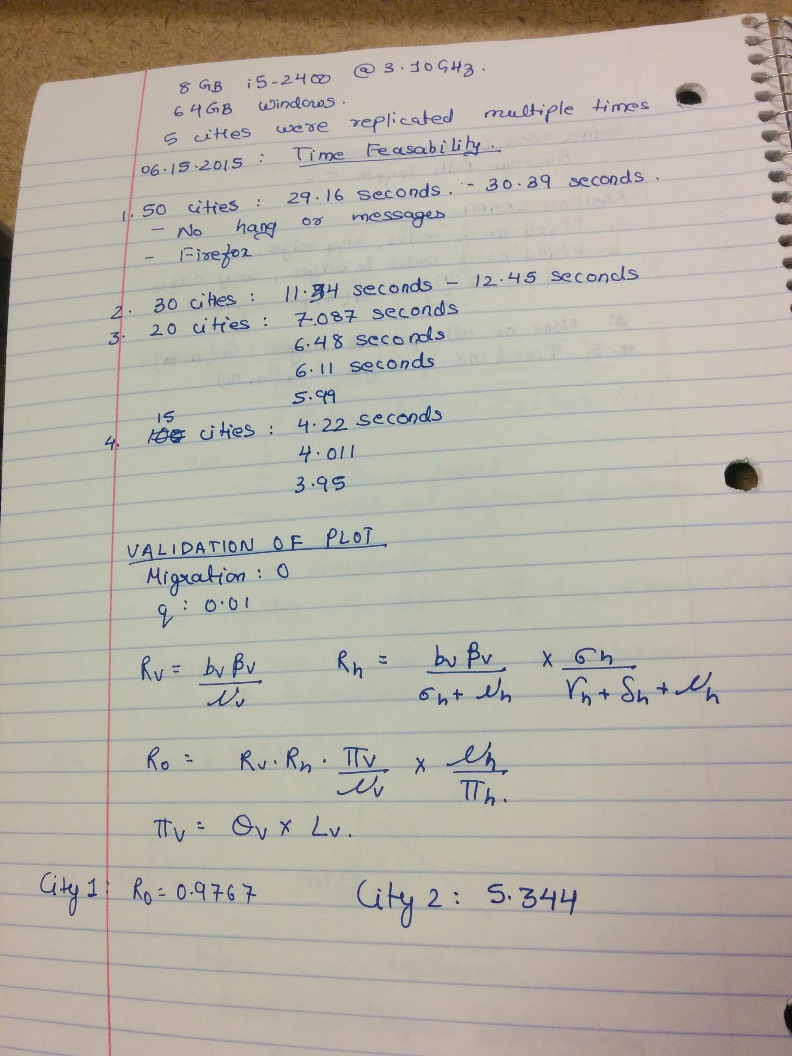
More tasks:

Find out the cumulative diseased during the entire period

Find out incidence (new cases) during the entire period.

**WEEK 4: Validation**

6/15



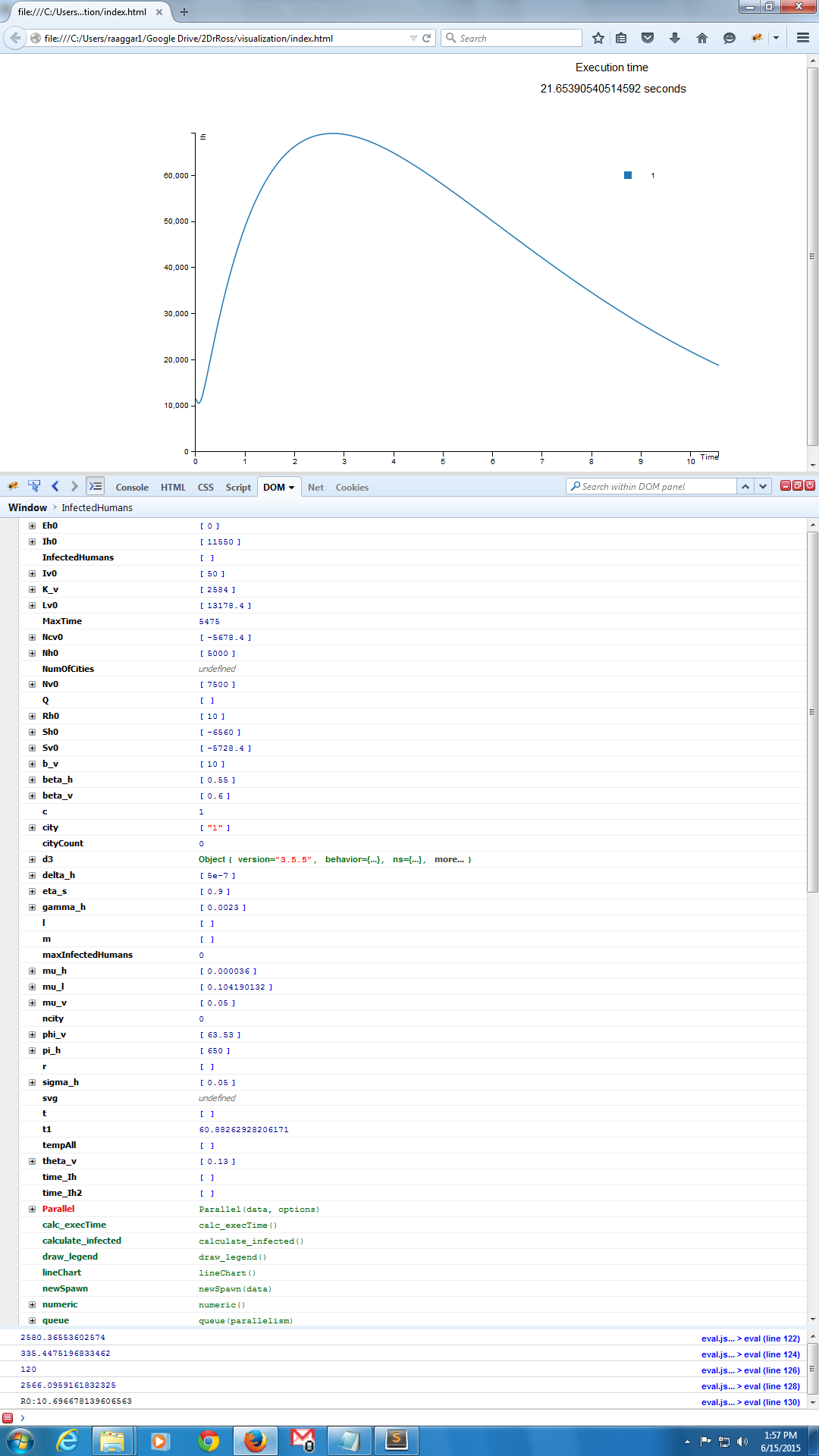
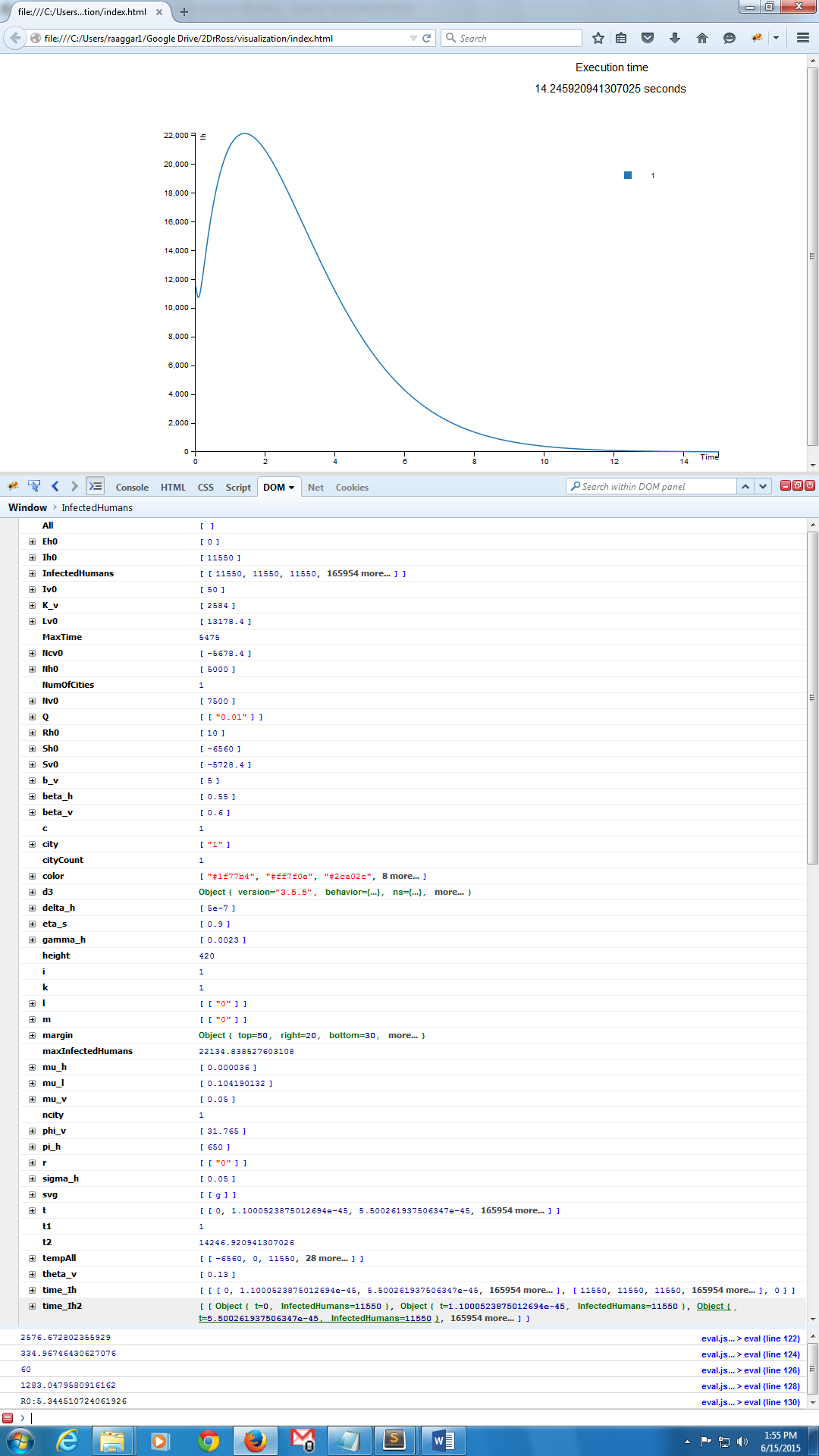
300,000 iterations for 15 years

Parameters:



|  |  |
| --- | --- |
| Q | 1 |
| 1 | 0.01 |

|  |  |
| --- | --- |
| M | 0 |
| 0 | 0 |



With b\_v = 10

**MATLAB vs JS**

The function numeric.dopri() is an implementation of the Dormand-Prince-Runge-Kutta integrator with adaptive time-stepping:

ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a one-step solver – in computing y(tn), it needs only the solution at the immediately preceding time point, y(tn-1). In general, ode45 is the best function to apply as a first try for most problems.

JS

T: 0 – 5 years

InfectedHumans: 57988.66092330328

Iteration: 300,000/363,018 (results same)

MATLAB:

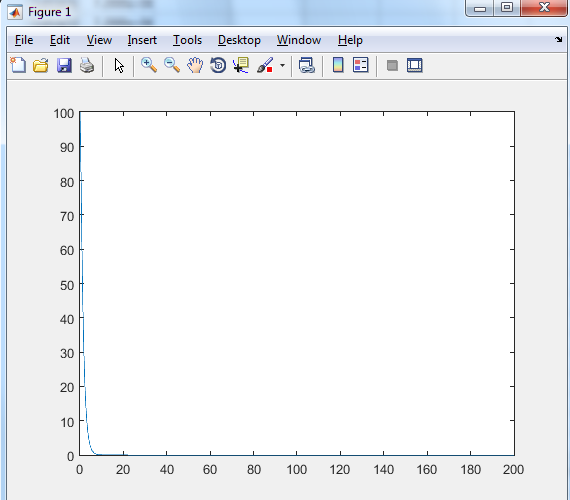
InfectedHumans: 55183.02

Iteration: 363,018

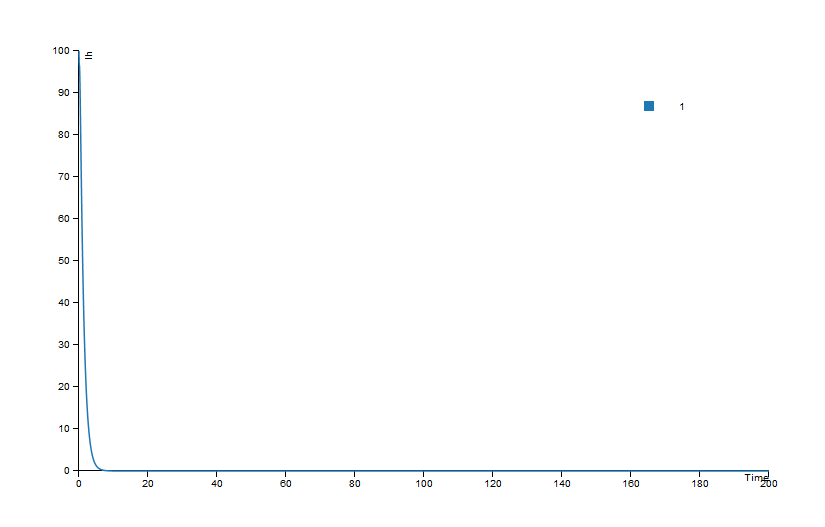
http://bl.ocks.org/robinhouston/ed597847175cf692ecce

**Date: 06.16.2015**

**MATLAB (execution time: 221 sec)**



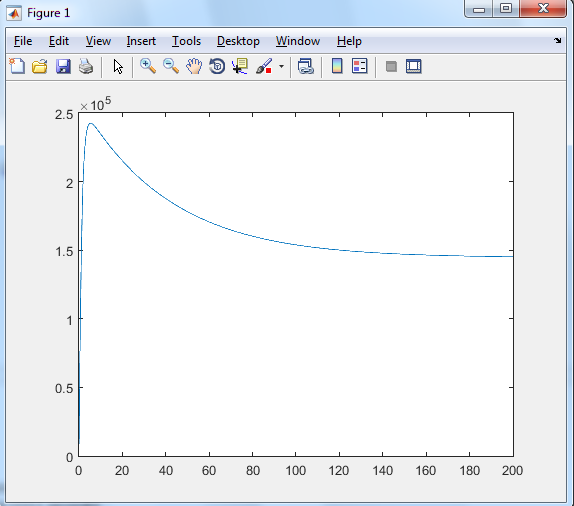
1.93723391171903e-72 at time 200 years

**JS (execution time: 93.64203014733 seconds )**

Object { t=200, InfectedHumans=6.186487905682725e-82}

Comparison 2:

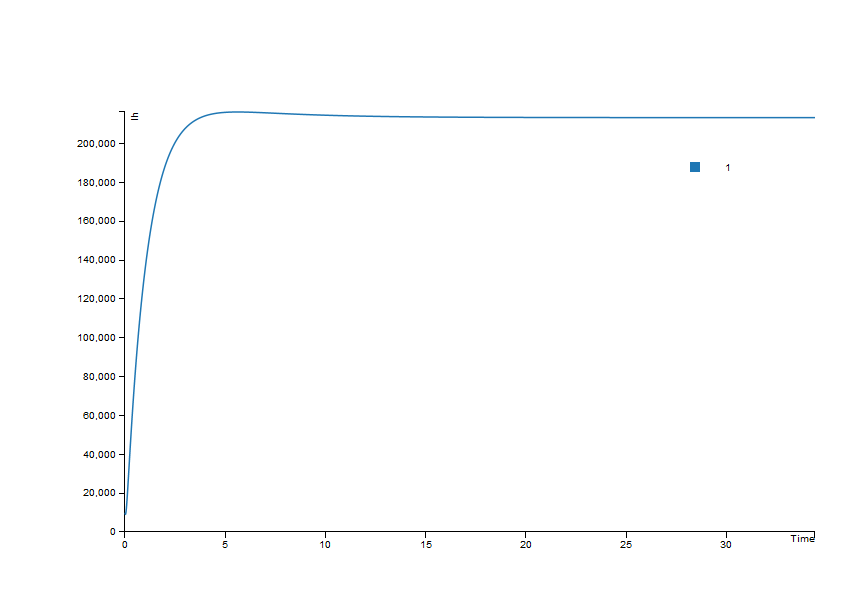
MATLAB (2211 seconds)



T=200, InfectedHumans: 145223.391914719

T=34.426, InfectedHumans: 194027.093396014

**JS (execution time: 202 seconds)**



Object { t=34.42647931444812, InfectedHumans=213361.2822384632}

06.23.2015

Slider reference: <https://github.com/turban/d3.slider>

Created temperature and time slider, yet to be integrated with computation.