ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:

Modelling and Simulating Social Systems with MATLAB

Project Report

Insert Title Here
...

Barton Patrick & Beglinger Lars & Brennan Liam

Zurich

December 2017

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Barton Patrick & Beglinger Lars & Brennan Liam

Contents

1 Abstract

2 Individual contributions

GitHub allowed us to work individually on our project. After sitting together for a few hours, brainstorming and defining upcoming tasks, the code from each contributor has been coding and debugging on their own for the most part. Lars and Liam started off by some research into the twitch API and how this data is being used already, trying to find a reliable source for our project where we want to reproduce existing, measured data with our model. In a next step, Liam made a skeleton with the core functions evaluating the viewer and streamer compatibility, focusing on the stochastics to get our desired normal distribution. Lars then extended the core and added more mechanisms like a factor of uncertainty before Patrick introduced the time dimension as well as the individual refresh rate. Additionally, he plotted our results similarly to the format the measured data is available. From there on, Lars added additional elements influencing the dynamics such as a streamers schedule a viewer threshold and viewer refresh inertia. To finish it off, Lars and Liam worked on an efficient and clean way to present the data. To sum it up, our efforts were well distributed amongst the team members and we all had our field of expertise: Liam focused on the statistical aspect, Patrick on the dynamical part and Lars was responsible for the implementation of real world mechanisms influencing viewership in MATLAB. On a side note, it should be noted that all three team members are part of a very time-consuming focus project. Therefore, we had to limit the time for this simulation, even though further extending the MATLAB script would be incredibly interesting.

3 Introduction and Motivations

Twitch is a live streaming platform mainly focused on video games. Since its launch in 2011, it has grown rapidly and now provides entertainment for 15 million unique daily users (twitch.tv/about), ranking it in the top 50 of the most frequented websites in the world. (alexa topsites) Everyone can set up their own live stream, build a community and show off their video game skills and entertain their viewers. Gamers streaming the same game are often competing against each other for viewers. Unlike in traditional television, there is a two-way interaction between the streamers and the spectators through a chat, adding a new layer of complexity to the viewer count dynamics. We all are frequent visitors on Twitch: There, we seek entertainment, we are following eSports tournaments and we are watching professional gamers play on an extremely high skill level. We often were amazed by how many viewers a streamer can attract, and ultimately by how much money a handful of streamers earns, all that while playing video games. We noticed how some streamers start their broadcast right around the time the most popular channel goes offline, effectively offering an alternative to the sudden lack of entertainment and trying to attract some bored viewers that would never tune in if more popular people are streaming. It is a truly fascinating and complex system and in this course, we saw the opportunity to analyse that behaviour in detail on a scientific level. Even though we are not planning to start our own Twitch streams, it undoubtedly is interesting to find out how a channel might be able to substantially increase their viewer count by some basic improvements.

4 Description of the Model

4.1 Skeleton

Undoubtedly, the human mind is extremely complex. Why we get along with a specific person depends on a virtually uncountable number of reasons. Some of them might be clear enough to verbally define and isolate, but a large part of those reasons is controlled by our subconsciousness. This is the first step and most important part of our simulation. How can we simulate how a viewer likes a streamer without getting lost in defining thousands of parameters? Even if we went for this approach, our results might have turned out to be wrong, since there is always a chance of forgetting a crucial characteristic. To come up with a more time effective solution than the one described above, we decided to generate a set amount of characteristics with random values per streamer while defining a weighted desire of characteristics a viewer is looking out for when watching a twitch channel. With this approach, we can achieve a normal distribution in total streamer quality and a matching viewer allocation without having to fine tune parameters until we match our given data (Figure 1). Only the amount of our undefined parameters has an impact on the viewer distribution, reducing our input to the system to one parameter.

4.2 Mechanisms

After creating the skeleton of our simulation, we added the most important mechanics observed in real life to our script. First, we wanted to create a schedule for streamers, adding the time dimension to our system. Again, defining a set schedule for every streamer would have been a lot of work and would have left room for errors and manipulation on our side. There again, we defined a random frequency for every streamer, resulting in a streaming time of four to eight hours per cycle. In a next step, we added a channel switch inertia on the viewers side. If every viewer checked

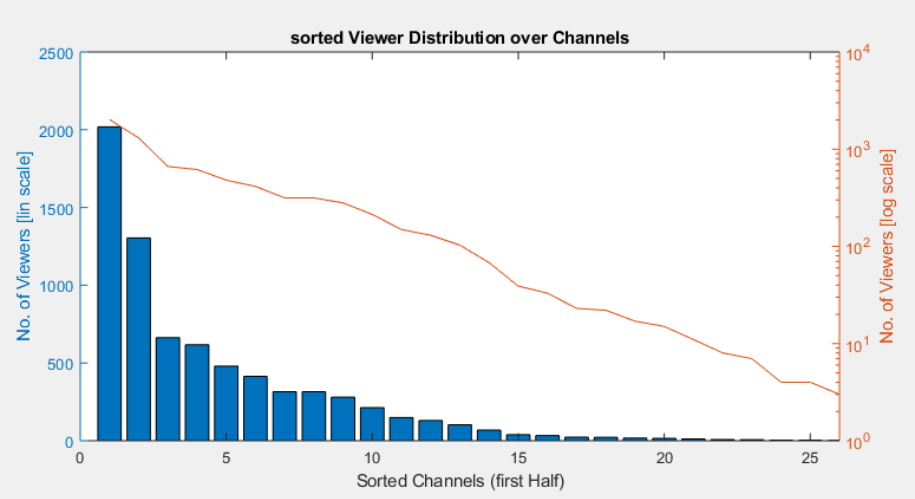

Figure 1: Only using the MATLAB function rand() to generate Matrices, we achieved a normal distribution of the overall streamer quality.

for a better entertainment source on Twitch infinitely often, the viewer distribution would shift with every change in streamer availability, but it would happen in very abrupt steps. Adding yet another randomly generated parameter to the system, every viewer has been assigned a chance to check whether a better option came online in one time step of our simulation. This mechanic results in a fast shift in viewership early on and approaches its limit over a longer period. Since the ever-changing streamer availability, this limit is never completely reached. So far, we assumed that every viewer would be online all the time, watching a channel with a very low compatibility out of lack of better alternatives. To guarantee a more exact model, we implemented a threshold for viewers, making them go offline if no streamer suits them. If the compatibility between a viewer and a streamer is below the mean of the compatibilities for the entire option in streamers minus the standard deviation, the viewer no longer watches any channel until the next time step.

5 Implementation

The implementation was done in two steps. First, we started out with a static model which calculated viewership at a particular time. In a second step, we then went on to add behaviour to our model which would allow us to simulate it over time.

5.1 Static Model

Here we started by defining the core variables we could adjust.

Variable	Explanation
n_viewers	The number of viewers to be simulated
n_streamers	The number of streamers to be simulated
n_attributes	The number of abstract attributes each streamer/viewer has.

Having set these, two matrices are generated. One, which randomly allocates a weight for each attribute to each streamer, and one that does the same for each viewer. By summing all the attribute weights of a viewer one receives his dedication - how much he or she watches things on twitch. Similarly, when summing over all attributes of a streamer one gets his overall quality - how good he/she is. In the 'Viewership_calculator' two histogram are made which show the number of streamers/viewers that have a certain overall quality/dedication. These figures where used to check the validity of our assumption that we can use random weights and have a Gaussian distribution of dedication/quality of the viewers/streamers. See Figure 2.

Furthermore, the compatibility between each viewer and streamer is calculated. This is done by taking the dot product of a streamer and all his attributes with a viewer and his attributes. The result, referred to as compatibility in this project, is a measure of how likely a viewer is too watch a certain streamer. Once this calculation

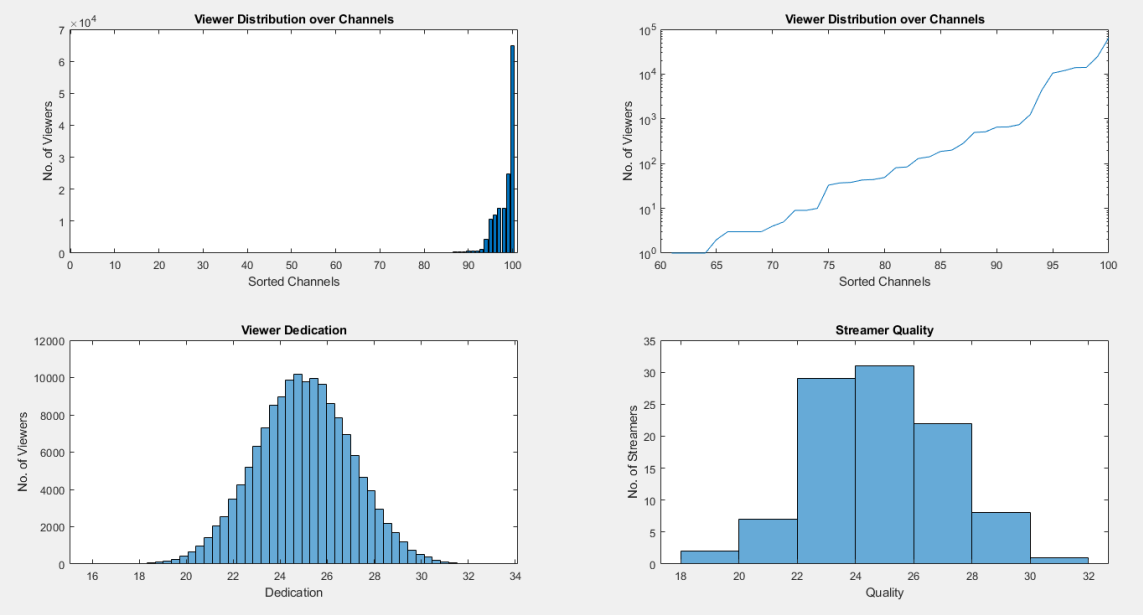

Figure 2: Top: Distribution over channels is exponential. The logarithmic view on the right shows that most channels have a small amount of viewers. Bottom: Viewer Dedication and Streamer Quality approximate Gaussian distributions. *Note that these graphs will vary slightly each time the simulation is run, as the attributes are generated randomly.*

has been performed, one can assume that a viewer will be watching the streamer with highest compatibility.

Having calculated which streamer is being watched by each viewer, the results are summed up to obtain the number of viewers each streamer has. A quick graph showing the viewership of each streamer in a sorted way proves the exponential distribution that we wanted to achieve. This represents the fact that the most people will be watching popular channels while there are many other channels with only a few people watching.

5.2 Dynamic Model

The dynamic model 'Viewership_animated' takes what has been developed in the static model and builds a framework around it to allow data to be changed and updated. As before, we can adapt the main variables n_viewers, n_streamers and n_attributes, to change the scale of the simulation.

We now define a simulation time t and a time step. The simulation can then be done by updating in discrete time steps which we set to represent 15min in real time.

To model the dynamics of Twitch viewership several further variables are introduced.

Variable	Explanation
streamerOnline	If a streamer is online or not
streamerOnTime	How long a streamer will stay online
viewerUpdateRate	How likely a viewer is to change channel

In addition, the time of each streamer needs to be kept to be able to determine when they will go online or offline.

With everything defined, things that don't need to be updated, such as viewer dedication, streamer quality and compatibility, are precalculated at the start. Everything else is written in a loop, in order to be updated iteratively.

In each iteration, streamer on and off times are checked and a streamer's online status is changed accordingly. With a certain probability, defined by the viewerUpdateRate, viewers will then check for a better compatibility streamer and swap channels if they find one. The resulting viewership is then recalculated and updated.

Viewership is plotted in bar graph showing the distribution of viewers over streamers. With each iteration of the simulation results are updated in the figures after a wait time of 0.5s. This results in an animated graph where 1s real time represents 30min simulated time.

6 Simulation Results and Discussion

Due to the fully random nature of our simulation, every run will yield a different result. In general, we can look at the resulting data and explain different phenomena in the data. In Figure 3 we can see how the Data is presented to the user.

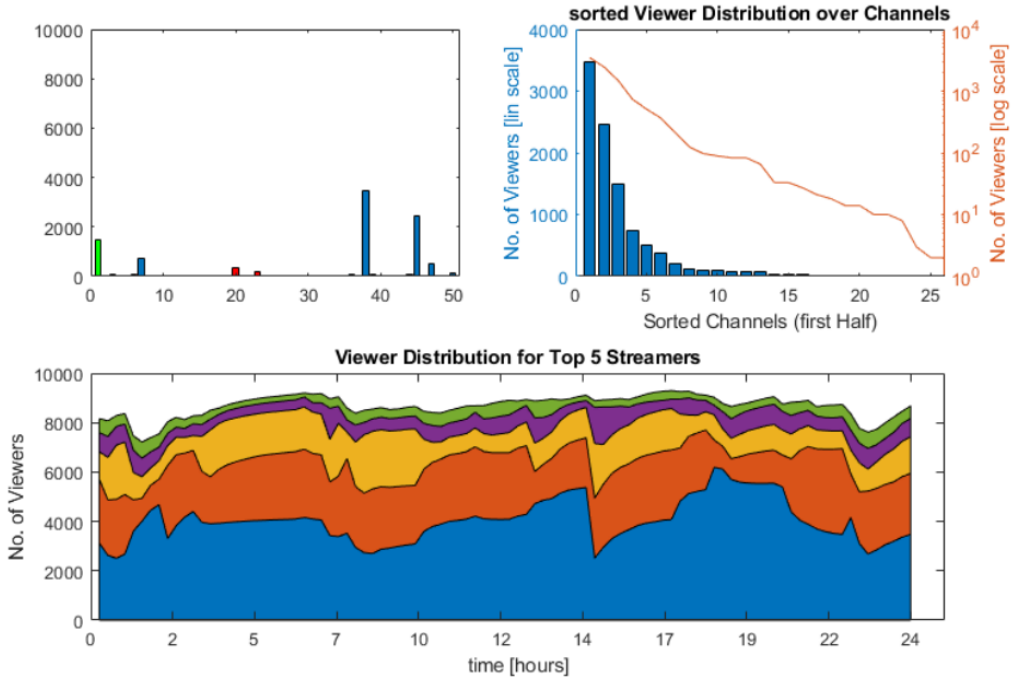

Figure 3: Resulting data from a random simulation run.

From upper left to lower right the figures show viewers per channel unsorted, with colours indicating changes of channel from online to offline and vice versa. The second figure is a sorted viewer distribution with a linear and a logarithmic scale. Based on how the compatibility is set up we expect an exponential distribution. This is also approximately what can be observed on the actual platform (Figure 4) During the simulation the actual distribution deviates from the expect one for different reasons.

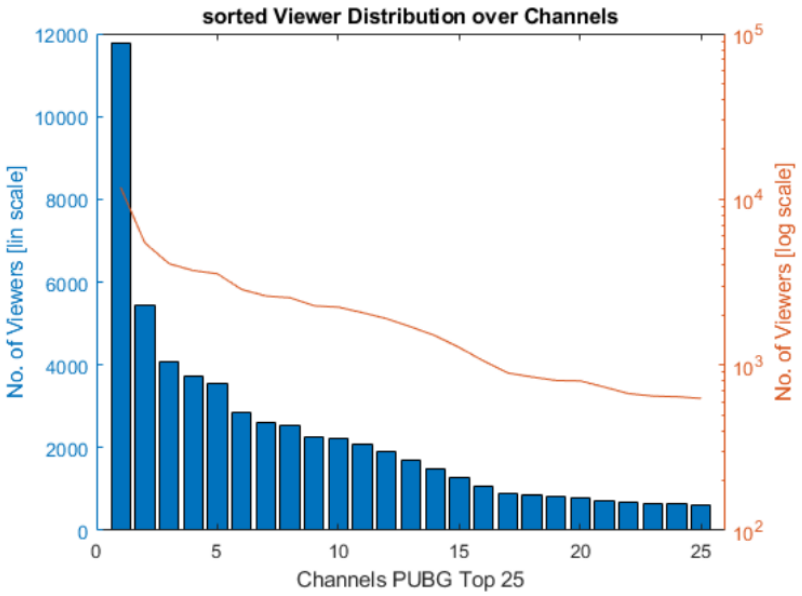

Figure 4: Viewer distribution among top 25 channels streaming PLAYERUNKNOWNS BATTLEGROUNDS on the 17.12.2017 at 15:00.

The most important of which are: Smaller number of simulated viewers and channels than is the case on Twitch itself. In the displayed Run we simulated 10000 viewers and 50 channels instead of 860000 and 25000 for the entire twitch-ecosystem or 90000 and 2000 for the PLAYERUNKNOWNS BATTLEGROUNDS (PUBG) ecosystem as it is Displayed in Figure 4. From here on, comparisons will be between the model and the data from PUBG streams, since this approximately represents any ecosystem, with the only difference being the total number of streamers and channels.

Average Factor Model	1.43021
Average Factor Reality	1.14563
Median Factor Model	1.245283
Median Factor Reality	1.092158
Standard deviation Model	0.516185
Standard deviation Reality	0.226382

Table 1: Analysis of the factors form one channel to the next better one (top 25) from real data and the simulation

As can be seen in Table 1 the simulation leads to a stronger drop off of viewers from one channel to the next lower one than can be observed on twitch itself.

From these comparisons we can also tell, that our model is quite a good static representation for the top channels of a single ecosystem. It does not consider the dynamics and interactons between different game ecosystems. To implement this interaction would require adding another dimension of complexity to the

simulation that simulates a certain number of ecosystems parallel to each other and considers a restricted amount of interaction and overlap between the communities of different games. It is also not very accurate for the bottom 60% of channels, since these have 0 viewers in the simulations and 0-5 on twitch. This could be explained with the fact that most of these 0-5 viewers either know the streamer personally, are the streamer himself or bots, all of which are not considered in the model although this is pure speculation.

Comparing the different transient results leads to a different conclusion. Since our Model assumes, that the streamers and viewers are uniformly distributed among time zones and have a random non-optimized schedule, the viewer distribution over time looks very different to the observed data. If we wanted to recreate reality for a specific game ecosystem this could be done with the existing model, by tediously manually entering the schedule of the streamers and matching streamers in the model to real streamers using their quality. Another solution to this would be to implement time zones for viewers and streamers and distribute their number according to reality. This is also quite complex and would require more data than we have access to. The conclusion is that the model does represent the dynamic changes over time, but does consider the schedule of the population.

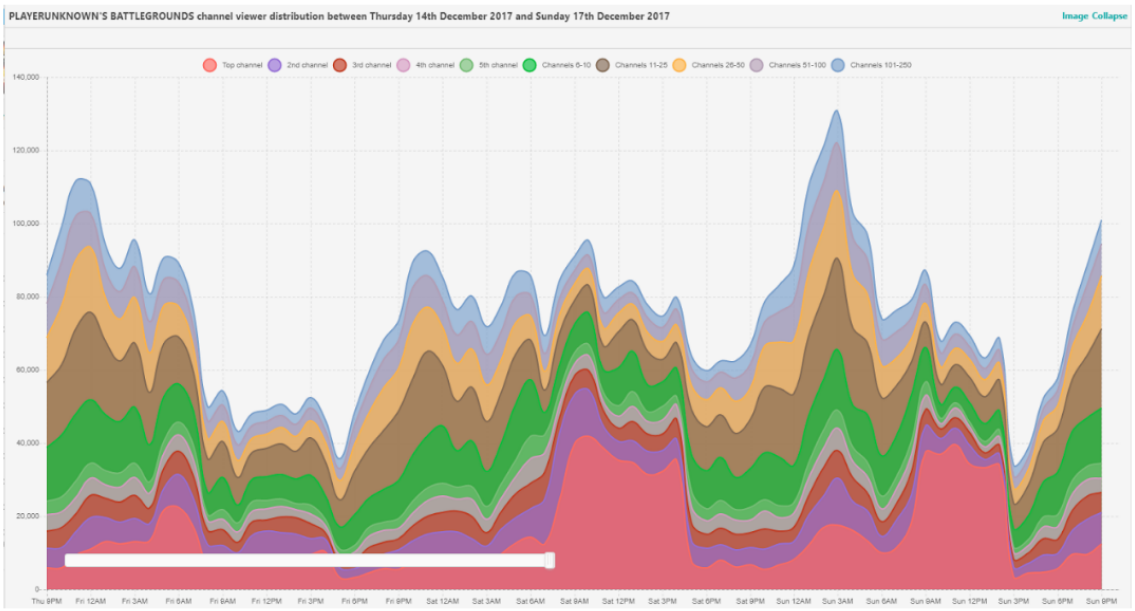

Figure 5: Viewer Distribution in PUBG over 3 Days

7 Summary and Outlook

By modelling all the viewers and streamers as people with different abstract qualities and interests we managed to recreate a good representation of the interaction inside different ecosystems inside the streaming platform Twitch. The model we created matches viewers to their most compatible streamers, assigns a tolerance for stream compatibility, assigns endurance to streamers and simulates the dynamic interaction over time. With this we created a solid foundation for modelling the complex dynamics of viewership. To further extend this model and make it usable one would have to implement a demographically accurate population inside the simulation and enter real world schedules and qualities. Extending the model to allow consideration of interactions and overlaps between different communities would require research and data of how these communities function. Some of these things can simply be done by tediously entering data and running the simulation as is, and others require a lot more work to be done implementing population dynamics. However, if we can manage to implement these proposed solutions, the tool could become indeed quite useful for newcomers in the streaming world to find their optimal streaming schedule and style, although some tweaking of certain parameters would be necessary.

8 References