Reach Calculation Using BBD

Emerson, Tegan [1]

[1]Department of Mathenatics, Colorado State University

1 Problem Statement

It is our goal to select an appropriate model for estimating the reach of a given advertising schedule so that we can modify the currently implemented algorithms to include a reach based constraint. We define the reach of a single ad placed into a schedule to be the proportion of total viwers belonging to the target demographic who see that ad at least once. Effective reach, alternatively, is the proportion of the total viewers belonging to the target demographic that see the ad more than some threshold number of times (the threshold can be identified by the advertising company). In order to model effective reach, the standard approach is to generate an exposure distribution for the target audience. Consequently, this is what we would like to do in order to have the greatest flexibility in our reach constraints. Stochastic approaches build exposure distributions based on probabilisitic assumptions about the setting of the problem. Univariate models treat the entire schedule as an object and are less computationally expensive. Multivariate models would treat each hour on a specific day as an object, for example. The multivariate models also require estimation of pairwise duplications/correlations and do not scale well. The Beta Binomial Distribution (BBD) is a fundamental model used in both univariate and multivariate exposure diistribution approaches. In the BBD the probability of viewing a unit (a success) is assumed to Beta distributed (a very flexible distribution that can be fit to a wide variety of data). We closely follow the method proposed in the paper "Improving the Estimation Procedure for the Beta Binomial TV Exposure Model" by Rust and Klompmaker from 1981.
We first present the approach for calculating reach as described in the paper. Next, we describe the implementation of the algorithm in greater detail including which Matlab scripts are used in each step of the approach. Preliminary validation results are then shown for the scenario described. The preliminary results compare the performance of this method relative to reach calculations directly from the historical Nielsen data. All Matlab code is provided in the appendix while the SQL commands for pulling the needed Nielsen data are included as needed in the text of the document.

2 Using a Multiplicative Regression Model for Calculating Schedule Reach

As we have previously discussed, the Beta Binomial Distribution (BBD) is a modification of the Binomial distribution that assumes the chance of success (viewing a unit) is Beta distributed. The Beta distribution has two parameters associated with it and is denoted as $B(\alpha, \beta)$. A BBD is completely determined by the parameters of the Beta distribution and the number of units in the schedule. Consequently, the most common approach for modeling the BBD is to fit a Beta function to probability of exposure data. Rust's paper does this by estimating the moments of the Beta distribution (the mean and variance of the distribution) which are given by

$$\mu = \frac{\alpha}{\alpha + \beta}, \quad \text{and} \quad V = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}.$$

Rust's approach hinges on the following two assumptions:

1. The mean of the Beta distribution is the average rating of the slots in the schedule containing an insertion and

2. The variance of the Beta distribution can be interpreted as a measure of the audience duplication in the slots containing units.

In moving forward it is useful to define a variable which is closely related to the variance by

$$V^* = V_{sup} - V$$

where V is the variance of the Beta distribution and V_{sup} is the best upperbound of the variance. This variable is used instead of V because in the case that the Beta distribution is a very poor fit it is possible to have a negative estimate for the variance and even in that case the value V^* is necessarily positive. Using the equations for the variance and mean given above it is possible to write the variance as

$$V = \frac{\mu^2(1 - \mu)}{\alpha + \mu}.$$

For a fixed mean, the variance varies inversely with the shape parameter α. Consequently, best upperbound on the variance can be computed by taking the limit as α tends to zero. This produces the expression

$$V_{sup} = \mu(1 - \mu).$$

Finally, given V^* and μ the Beta parameters can be computed as

$$\hat{\alpha} = \frac{\mu V^*}{(\mu(1 - \mu) - V^*)} \quad \text{and} \quad \hat{\beta} = \frac{\hat{\alpha}(1 - \mu)}{\mu}.$$

In reality, Rust claims, that the variance is usually very small and consequently it is reasonable to estimate the value of V^* by V_{sup} for the fixed average rating of the slots containing the units in the schedule.

The key assumption is that the value V^* is supposed to be an indication of the level of duplication of the viewership for the schedule. As a result, we use a multiplicative regression model for the V^* that takes into consideration the characteristics of a schedule that would effect the degree of duplication. The multiplicative regression model is

$$\hat{V^*} = A(1+X_1)^{B_1}(1+X_2)^{B_2}(1+X_3)^{B_3}(1+X_4)^{B_4}X_5^{B_5}X_6^{B_6}X_7^{B_7}$$

where

- A is a scaling constant,
- X_1 is the proportion of same-channel unit pairs,
- X_2 is the proportion of same-daypart unit pairs,
- X_3 is the proportion of same-program type unit pairs,
- X_4 is the proportion of self unit pairs,
- X_5 is the average rating of the time slots containing a unit in the schedule,
- X_6 is the variance of the ratings of the time slots containing a unit in the schedule, and
- X_7 is the number of units in the schedule.

We add a one to each of the proportions in the model because multiplying by a power of just the proportion would result in shrinking V^*, i.e. the measure of duplication. However, by adding one to each of the proportions we cause the duplication to go up (or remain steady) depending on the proportion (a higher proportion should increase duplication more than a lower). The goal of the regression is to determine the exponents and scaling coefficient (A, B_1, B_2, B_3, B_4, B_5, B_6, and B_7). By taking the logarithm of both sides of the model we obtain a linear regression. Resultantly, we can generate random schedules and compute the statistics of each and estimate V^* for each schedule using the approximation $V^* = \mu(1-\mu)$ and perform the regression. It is important to note that for my experiements I used 5 dayparts (1:00-4:00,5:00-9:00,10:00-13:00, 14:00-18:00,19:00-24:00).
Having now explained the model and the approach, we can implement the model for the following experiment.

3 Implementation of the Model

In order to test the model we will use the Nielsen data available to us. For my experiment I chose the target demographic to be adults aged 21-34. Schedules were for the ten discovery channels. For simplicity it was assumed that there was a single avail per time slot and schedules were generated for a single week. Due to this assumption we leave out the proportion of self-pair unit pairs (same-program unit pairs). We also do not have a metric for evaluating the "same-program typeness" of the slots and so this proportion is also left out of the model. In a more realistic scenario where there are multiple avails per hour we would put back in

the self-pair unit pairs proportion.

The data that I needed to start with was the average rating of each hour-day-channel combination for the target demographic. I pulled this data from Nielsen over the time-span 11/01/2014-01/01/2015 using the following SQL command (*Yes, I am sure there is a more efficient way to do this*)

```
WITH s1 AS (SELECT DISTINCT household_number,person_id, AVG(viewing_weight_1)
AS avg_weight FROM nielsen.npm_57 WHERE viewing_source IN
('AHC','APL','DAM','DFC','DISC','DLIF','ID','SCI','TLC','VE') AND
age_gender_building_block_code IN ('G','H','I','V','W','X') AND
viewing_source_type_code='C' AND broadcast_date between  '2014-11-01' AND
'2015-01-01' GROUP BY household_number, person_id),

s3 AS (SELECT DISTINCT household_number,person_id, viewing_source,
broadcast_date,hour_start_time, day FROM nielsen.npm_57 WHERE
viewing_source IN ('AHC','APL','DAM','DFC','DISC','DLIF','ID','SCI','TLC','VE')
AND age_gender_building_block_code IN ('G','H','I','V','W','X') AND
viewing_source_type_code='C' AND broadcast_date BETWEEN  '2014-11-01'
AND '2015-01-01'),

s4 AS (SELECT s3.household_number, s3.person_id, s3.viewing_source,
s3.broadcast_date, s3.hour_start_time, s3.day, s1.avg_weight FROM s3 JOIN s1 ON
s3.household_number=s1.household_number AND s3.person_id=s1.person_id),

s2 AS (SELECT viewing_source,broadcast_date,day,hour_start_time,
SUM(avg_weight) AS viewing_weight FROM s4 GROUP BY viewing_source,
broadcast_date,day,hour_start_time)
SELECT viewing_source, day,hour_start_time, AVG(viewing_weight) FROM s2 GROUP BY
viewing_source,day,hour_start_time;
```

The CSV file produced from the above query is input into Matlab using the function "importnielsen_approach2.m" (which also checks for missing hour-day-channel combos and fills in with a rating of zero for those that are found to be missing). This produces a matrix for each channel that contains the average rating information. Once this average rating information has been formatted, the user specifies the number of random schedules to base the model off of. The number of schedules, average rating information, and an array containing the channel names included in the schedule are then fed input into the function "multiplicative_regression.m." Schedules are randomly created for the multiplicative regression by randomly populating an (hours)x(days)x(channels) array and then identifiying the slots with a value greater than 0.99 (the numbers are drawn from a uniform distribution over [0,1] and identifying the slots based on a threshold of 0.99 typically chooses about 20 avails out of the 1680 total avails). The previously described statistics are computed for each random schedule and based on those values the regression outputs a vector, B, that contains the regression exponents and scaling factor. In order to compute the frequency distribution for a new schedule you input the average rating data again (or the predicted ratings), the new

schedule, the channel label array, and the B vector into 'approach_2_reach_calc.m' (you also have the option to identify a reach threshold if you want to compute effective reach based on an advertiser threshold). This function outputs the frequency distribution and the effective reach of the schedule based on the provided inputs.
Breaking down the algorithm, there are four main steps.

1. **Compute Average Hour-Day-Channel Ratings:** Here we input the CSV that results from the SQL query into "importnielsen_approach2.m" (all Nielsen data from 11/01/2014-01/01/2015 for 21 to 34 year olds). The output of this step, for a single channel is shown in Figure 1.

2. **Generate Points to Perform Multiplicative Regression:** Here we generate random schedules and compute the needed statistics from each of the randomly generated schedules. This step is implemented by the code "multiplicative_regression.m" and requires average hour-day-channel ratings, an array containing channel identifiers for the schedule, and the number of random schedules to be generated. The output of this is a vector containing the multiplicative regession exponents and scaling factors. Each random schedule produces a vector containing the statistics, for example

$$\mathbf{X}_i = [X_{i,1}\, X_{1,2} \ldots X_{i,7}].$$

   We also compute the estimate for V_i^*, namely $\hat{V^*}_i = X_{i,5}(1 - X_{i,5})$, for each random schedule. Finally, the output of this step is a matrix $\mathbf{X}$ containing all the statistics for the random schedules and a vector $\mathbf{V}^*$ containing all the V^* estimates for the random schedules.

3. **Perform the Multiplicative Regression:** This step is also carried out in the function "multiplicative_regression.m." Our multiplicative regression model is

$$\hat{V^*} = A(1+X_1)^{B_1}(1+X_2)^{B_2}(1+X_3)^{B_3}(1+X_4)^{B_4}X_5^{B_5}X_6^{B_6}X_7^{B_7}.$$

   We perform the following arithmetic to create a linear regression for the coefficients

$$\begin{aligned}
\hat{V^*} &= A(1+X_1)^{B_1}(1+X_2)^{B_2}(1+X_3)^{B_3}(1+X_4)^{B_4}X_5^{B_5}X_6^{B_6}X_7^{B_7} \\
\ln(\hat{V^*}) &= \ln(A(1+X_1)^{B_1}(1+X_2)^{B_2}(1+X_3)^{B_3}(1+X_4)^{B_4}X_5^{B_5}X_6^{B_6}X_7^{B_7}) \\
\ln(\hat{V^*}) &= \ln(A) + B_1\ln(1+X_1) + B_2\ln(1+X_2) + B_3\ln(1+X_3) + B_4\ln(1+X_4) + B_5\ln(X_5) + B_6\ln(X_6) + B_7\ln(X_7).
\end{aligned}$$

   To solve for the exponents and scaling factor we first create the matrix

$$\tilde{\mathbf{X}} = \begin{bmatrix}
1 & 1 & \cdots & 1 \\
\ln(1+X_{1,1}) & \ln(1+X_{2,1}) & \cdots & \ln(1+X_{N,1}) \\
\ln(1+X_{1,2}) & \ln(1+X_{2,2}) & \cdots & \ln(1+X_{N,2}) \\
\ln(1+X_{1,3}) & \ln(1+X_{2,3}) & \cdots & \ln(1+X_{N,3}) \\
\ln(1+X_{1,4}) & \ln(1+X_{2,4}) & \cdots & \ln(1+X_{N,4}) \\
\ln(X_{1,5}) & \ln(X_{2,5}) & \cdots & \ln(X_{N,5}) \\
\ln(X_{1,6}) & \ln(X_{2,6}) & \cdots & \ln(X_{N,6}) \\
\ln(X_{1,7}) & \ln(X_{2,7}) & \cdots & \ln(X_{N,7})
\end{bmatrix}$$

where N is the number of random schedules. With this in hand we can write

$$\ln(\hat{\mathbf{V}}^*) = \mathbf{B}\tilde{\mathbf{X}}$$

where $\mathbf{B} = [\ln(A)\ B_1\ B_2\ B_3\ B_4\ B_5\ B_6\ B_7]$. The least squares solution to this is given by

$$\mathbf{B} = \ln(\hat{\mathbf{V}}^*)\tilde{\mathbf{X}}'(\tilde{\mathbf{X}}\tilde{\mathbf{X}}')^{-1}.$$

My code outputs this **B** vector and other information if you want to see what the statistics were for the different schedules as well.

4. **Compute the Effective Reach and Frequency Distribution for a Novel Schedule:** Armed with the output of "multiplicative_regression.m" we can then feed in a new schedule, **B**, the array of channel names, and an effective reach threshold into the function "approach_2_reach_calc.m." This function computes the statistics of the new schedule and combines them with the multiplicative regression exponents to estimate V^* for the new schedule. Note that it would be possible to use the predicted ratings of each slot to estimate the average rating of the spots in the schedule ($\mu = X_5$) or the historical data. Once μ and V^* have been approximated we determine the parameters of the Beta distribution by

$$\hat{\alpha} = \frac{\mu V^*}{(\mu(1-\mu) - V^*)} \quad \text{and} \quad \hat{\beta} = \frac{\hat{\alpha}(1-\mu)}{\mu}.$$

These parameter estimates plus the number of units in the schedule is then all we need to compute the frequency distribution and from there the effective reach. In practice, this is the only computation that would need to somehow be incorporated into the schedule since the multiplicative regression is done ahead of time.

This completes a description of what it being implemented in Matlab. Next, we will describe the results of a preliminary validation of the method.

4 Preliminary Validation of the Method

In order to determine the quality of this approach, we need to compare the predicted reach estimates to reach computed from a historical week. What I did to test this was to pull the data for a single week (11/10/2014-11/16/2014) from within the time frame that I built my multiplicative regression model on. The SQL query used to extract this Nielsen data was

```
WITH s1 AS (SELECT DISTINCT household_number,person_id,
AVG(viewing_weight_1) AS avg_weight FROM nielsen.npm_57
WHERE viewing_source IN ('AHC','APL','DAM','DFC','DISC','DLIF','ID','SCI','TLC','VE')
AND age_gender_building_block_code IN ('G','H','I','V','W','X')
AND viewing_source_type_code='C' AND broadcast_date BETWEEN '2014-11-10' AND '2014-11-16
person_id),

s3 AS (SELECT DITINCT household_number,person_id,
```

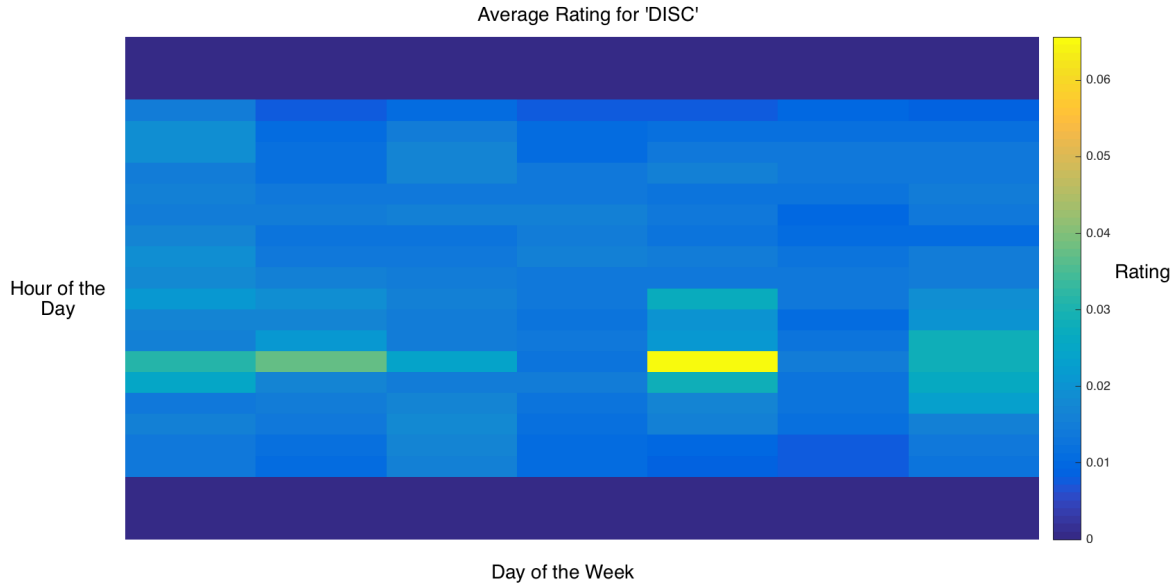

Figure 1: The weekly rating schedule for DISC from the historical data. The hour of the day starts at 12:00am in the top row and increases to 11:00pm in the bottom row. Sunday is the left most column progressing towards Saturday in the right most column.

```
viewing_source,broadcast_date,hour_start_time,day FROM nielsen.npm_57
WHERE viewing_source IN ('AHC','APL','DAM','DFC','DISC','DLIF','ID','SCI','TLC','VE')
AND age_gender_building_block_code IN ('G','H','I','V','W','X')
AND viewing_source_type_code='C' AND broadcast_date BETWEEN
'2014-11-10' and '2014-11-16'),

s4 AS (SELECT s3.household_number, s3.person_id,
s3.viewing_source, s3.hour_start_time, s3.day, s1.avg_weight FROM s3
JOIN s1 ON s3.household_number=s1.household_number AND
s3.person_id=s1.person_id)

SELECT household_number, person_id, viewing_source, hour_start_time,
day, avg_weight FROM s4 GROUP BY viewing_source,
hour_start_time,day,household_number, person_id, avg_weight;
```

In this SQL query I am assigning a single value to each viewer from the week based on their average viewing weight over the week. The frequency distribution for a specific schedule is obtained by counting the number of units from the schedule each viewer saw based on the historical data from the week. Reach is then computed as the percentage of the UE that was exposed to more than 1 (or we could define a reach threshold) unit in the schedule. Since the multiplicative regression model was built on random schedules, we wanted to compare the estimated reach to real reach for other random schedules. Recall the set up for a schedule: one avail per hour per day per channel for one week. With random schedules typically filling

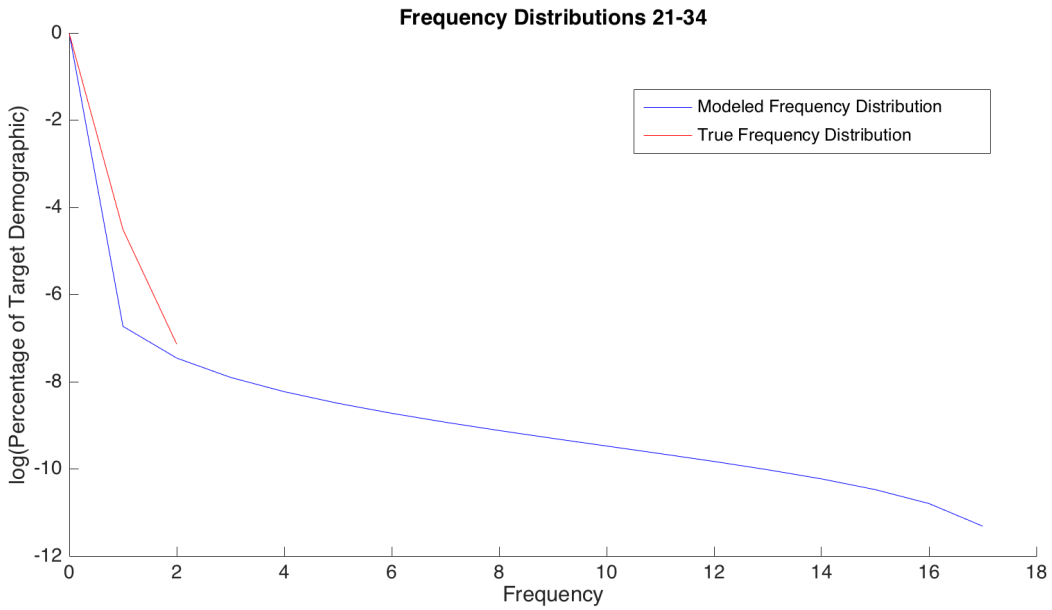

Figure 2: The red line is the hitorically computed frequency distribution for a random schedule and the blue line is the estimated frequency distribution based on the multiplicative regression.

about 15-20 avails, we would expect to get very low reach estimates and low reach in reality. It is also important to note that in our implementation we leave out same-program type unit pair proportion since we don't currently have a metric for measuring the sameness of shows. We also, because of the fact that the schedules I am generating only include one avail per hour for one week, don't include the self-pair unit pair proportion. There are three scripts associated with this intial validation test:

1. **"import_nielsen_week.m:"** reads in the CSV output of the weeklong data query,
2. **"hist_data_schedule_freq.m:"** which takes in the week of historical data and a schedule and performs the task of counting how many people were exposed to a particular number of the units, and
3. **"validation_script.m:"** which generates a multiplicative regression model and generates plots containing the reach errors and the true and modeled frequency distributions.

Figure 4 shows a sample of the frequency distributions for a random schedule computed both from the historical data and using the proposed estimation technique. The frequency distribution that is computed from the week in consideration stops at a frequency of 2 because for that week, no one person saw more than two of the units in the schedule.

We can see in Figure 5 that on average the proposed approach underestimates the reach of a schedule by 1%. These appear to be promising results, and need to be expanded on. We are rerunning this on a larger demographic (all adults 21+).

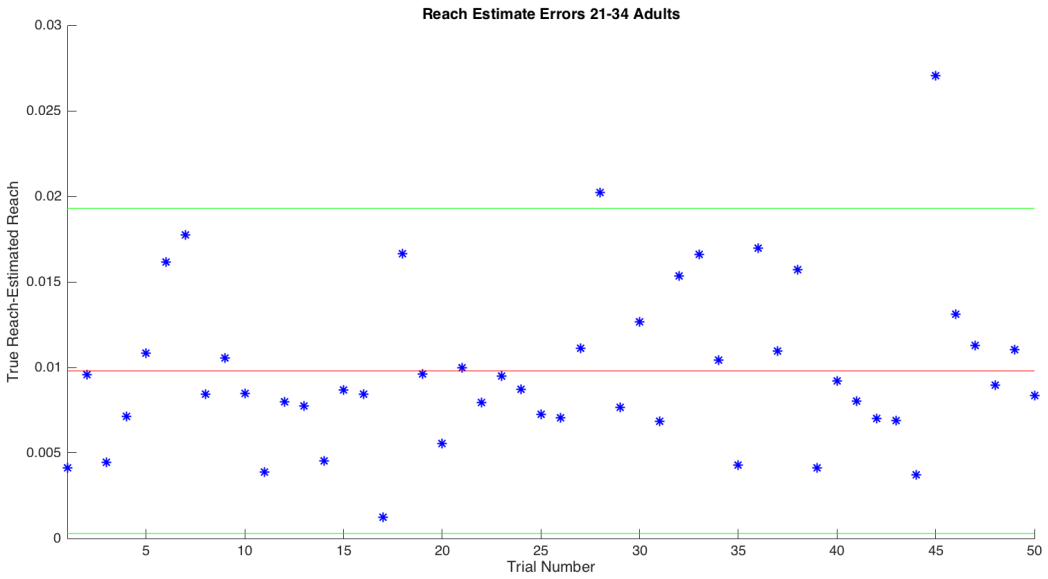

Figure 3: This figure shows the difference between the computed reach and the estimated reach for 50 random schedules. The red horizontal line indicates the average error and the green horizontal lines bound 95% of the errors.

I reran the validation for 21+ adults instead of 21-34 (a larger target demographic). Fewer random schedules were used for validation for reasons mentioned in Section 6. The preliminary results are shown in Figures **??** and **??**.

5 Future Directions

There are several things left that need to be done with this model.

- Generalize the framework for multi-week, multiple avails in the same timeslot, etc.
- Test and build the model on realistic schedules instead of random.
- Incorporate into the scheduler framework.
- Use predicted ratings for a schedule to test the model.
- Discuss how to account for the uncertainty associated with ratings into the modeling process.

6 Computational Considerations

Current implementations in Matlab are very slow when using large target demographics for validation. When you compute reach from a historical week, you are doing a very large

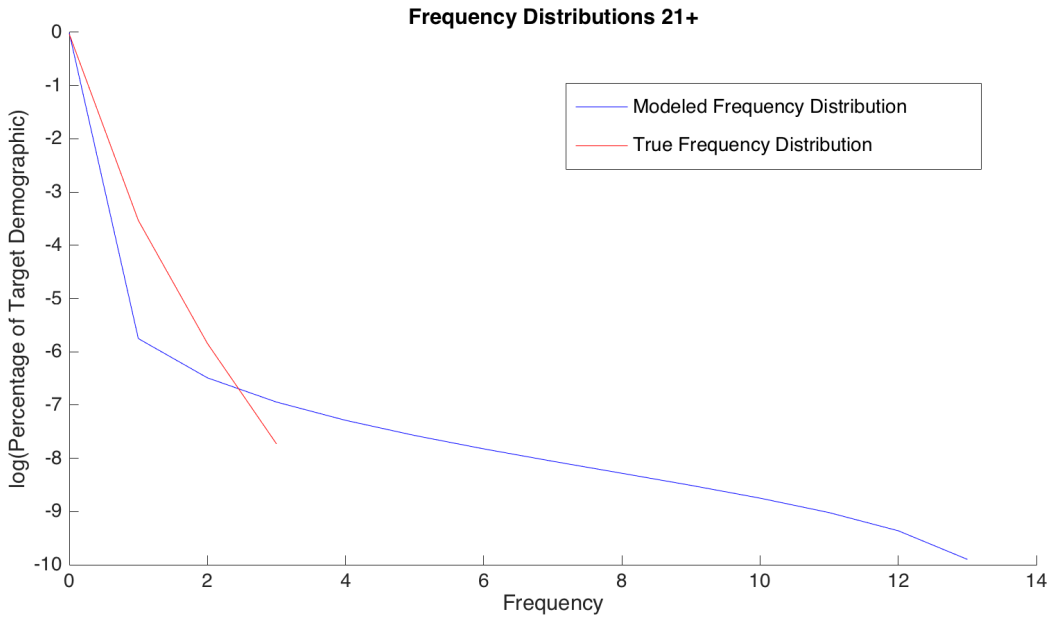

Figure 4: The red line is the hitorically computed frequency distribution for a random schedule and the blue line is the estimated frequency distribution based on the multiplicative regression.

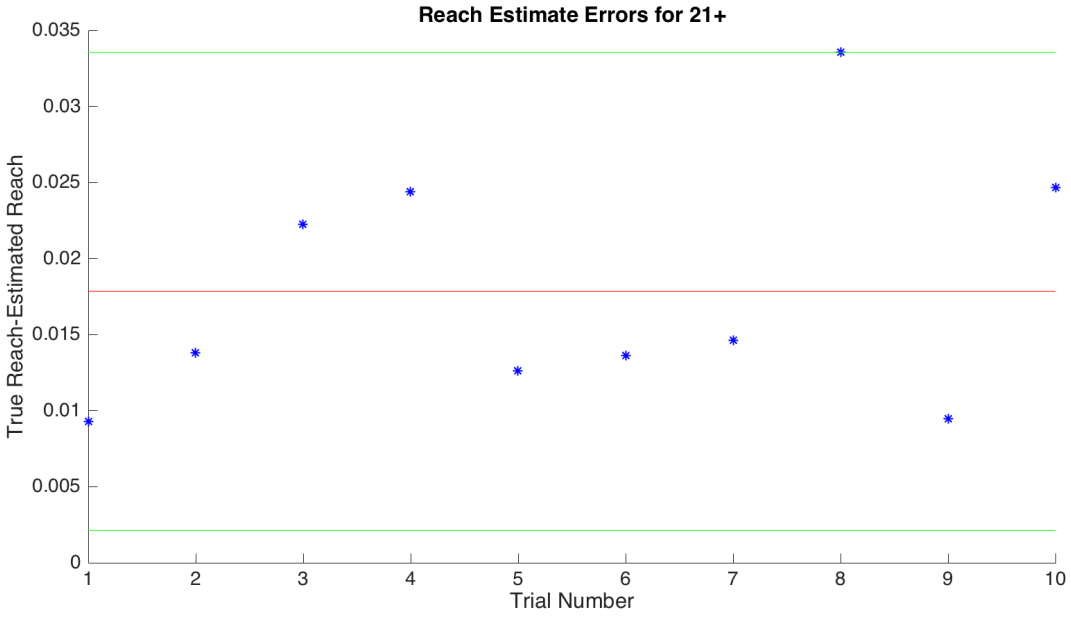

Figure 5: This figure shows the difference between the computed reach and the estimated reach for 50 random schedules. The red horizontal line indicates the average error and the green horizontal lines bound 95% of the errors.

number of string comparisons (to see if the day-hour-channel combination containing a unit appeared in a specific persons viewing history). Other programming languages are more efficient for this type of string comparison. Perhaps validation will be better done in another language. This is especially challenging if we want to do multiweek schedules to validate the model. There is a very high correlation between the value of V^* and the value of μ. This could be biasing the model and we may want to account for this.

A Importing Nielsen Data to Compute the Hour-Day-Channel Average Ratings

```
function [ch_d_h_av_rating] = importnielsen_approach2(filename, channels, UE_target_demo)
%IMPORTFILE Import numeric data from a text file as a matrix.
%   ch_d_h_av_rating = importnielsen_approach2(FILENAME) Reads data from text file FILENAME
%   a CSV file of the form exported from nielsen
%   data using the queries in the documentation.
%Inputs:        filename1: is the file containing the average viewership
%               information for a given demographic for every day and hour.
%               channels: is a cell array of the channels in the schedule
%               UE_target_demo: Is the universe estimate for the target
%               demographic of interest. Needed to compute the ratings.
%Output:        ch_d_h_av_rating: a cell array containing the information
%               about the average rating of each hour-day slot on each
%               channel in the schedule.
%

%% Initialize variables.
delimiter = ',';
    startRow = 2;
    endRow = inf;

%% Format string for each line of text:
%   column1: text (%s)
% column2: text (%s)
%   column3: text (%s)
% column4: text (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%s%s%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter', delimiter,
'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-startRow(block)+1,
     'Delimiter', delimiter, 'HeaderLines',startRow(block)-1, 'ReturnOnError', false);
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end

%% Close the text file.
fclose(fileID);
%% Create output variable
dataArray1 = [dataArray{1:end-1}];
```

```
%% Post processing to find the time slots missing for each channel.
load('day_labels.mat')
load('hour_labels.mat')
combos_in_data=strcat(dataArray1(:,3),dataArray1(:,2),dataArray1(:,1));
C=length(channels); %number of channels in the schedule
labels={};
for j=1:C
    num_char=length(channels{j});
    num_space_pad=5-num_char;
    if num_char==5
        CH_id=channels{j};
    elseif num_char==4
        CH_id=strcat(channels{j},{' '});
    elseif num_char==3
        CH_id=strcat(channels{j},{' '},{' '});
    elseif num_char==2
        CH_id=strcat(channels{j},{' '},{' '},{' '});
    elseif num_char==1
        CH_id=strcat(channels{j},{' '},{' '},{' '},{' '});
    end
for i=1:7
    Day=days(i); %pick a day
    for k=1:24
        Hour=hours(k); %pick a label
        label=strcat(Hour,Day,CH_id(1)); %concatenate to see if it showed up
                                         %in the data table.
        labels=[labels;label];
    end
end
end
%labels=labels(:,1);
[missing_slots,missing_ind]=setdiff(labels,combos_in_data);
%% Add in a zero value viewership for any missing day/hour combos
for zz=1:length(missing_ind)
      dataArray1=[dataArray1; labels{missing_ind(zz),1}(6:10) labels{missing_ind(zz),1}(5) labels{missir
end
channel_day_hour_proportion=cell(C*7*24,1);
for xx=1:C*7*24
    channel_day_hour_proportion(xx,1)={str2num(dataArray1{xx,4})/UE_target_demo};
end

%% Create output variable
ch_d_h_av_rating=[dataArray1 channel_day_hour_proportion];

end
```

B Perform the Multiplicative Regression

```
function [B,log_const,XX_hat,V_star_hat]=multiplicative_regression(ch_d_h_av_rating,
 channels, num_schedules);
%multiplicative regression performs the multiplicative regression to get
%out the coefficients needed to compute V* for a new schedule and
%consequently it's frequeny distribution. Takes in a the output of
%importnielson_approach2.m plus channel labels and the number of random
%schedules to be generated.
%
%Inputs:       ch_d_h_av_rating: the cell array containing the average
%               ratings from the desired demographic for each channel, day,
%               hour in the schedule.
%               num_schedules: number of random schedules you build the
%               model off of.
%               channels: The channel IDs for each schedule. Cell array of
%               strings.
%Outputs:       B: the vector containing the regression coefficients from
%               the model proposed by Rust.
%               log_const: the log of the scaling factor.
%               XX_hat: The measured statistics for each of the different
%               schedules
%               V_star_hat: Estimated values of V* used for the regression.
%

%start and end index for each of the five Nielsen dayparts
load('dayparts.mat')
%nielsen hour labels
load('hour_labels.mat')
%nielsen day labels
load('day_labels.mat')

C=length(channels);

XX=[];
VV_star=[];
M=num_schedules;
for i=1:M
    chance=rand(24,7,C);%create a random array the size of a schedule
    [rows,cols]=find(chance>0.99); %pick the slots where we will put an ad.
                                %above .98 will give us a (hopefully)
                                %sparse schedule.
    schedule=zeros(24,7,C);
    schedule(find(chance>.99))=1;
    sheets=ceil(cols/7); %will be used to identify the channels
    cols=mod(cols,7); %will be used to determine the day of the week
    A=find(cols==0);
    cols(A)=7;
    L=length(rows); %number of units in the schedule
    filled_slot_info=cell(L,3);
    %^allocate space for the information about the time slots filled
for j=1:L
    filled_slot_info{j,1}=hours{1,rows(j)}; %hour label
    filled_slot_info{j,2}=days{1,cols(j)};  %day label
```

```
    num_char=length(channels{sheets(j)});
    num_space_pad=5-num_char;
    %this part is gross because when the nielsen data is exported it space
    %pads the channel name to length 5
    if num_char==5
        filled_slot_info{j,3}=channels{sheets(j)};
    elseif num_char==4
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '});
    elseif num_char==3
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '});
    elseif num_char==2
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '},{' '});
    elseif num_char==1
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '},{' '},{' '});
    end
end

 same_channel_pairs=0;
for s=1:C
    W=sum(sum(schedule(:,:,s)));
    %^counts the number of units in a sheet (a single channel)

    %the lines below compute the number of same channel pairs. There are
    %less than two units on a channel then there are no unit pairs for that
    %channel
    if W<2
        same_channel_pairs=same_channel_pairs;
    elseif W>=2
    same_channel_pairs=same_channel_pairs+nchoosek((W),2);
    end
end
    X1=same_channel_pairs/nchoosek(L,2);
    same_day_part_pairs=0;
    %^proportion of same channel pairs

for k=1:5
    day_part_slots=sum(sum(sum(schedule(dayparts(k,1):dayparts(k,2),:,:))));
    %^computes the number of units in each daypart based on the daypart
    %splits predefiined by me in the vector 'daypart.mat'

    %the lines below compute the number of same channel pairs. There are
    %less than two units on a channel then there are no unit pairs for that
    %channel
    if day_part_slots<2
       same_day_part_pairs=same_day_part_pairs;
    elseif day_part_slots>=2
       same_day_part_pairs=same_day_part_pairs+nchoosek(day_part_slots,2);
    end
end
    X2=same_day_part_pairs/nchoosek(L,2);
    %^proportion of same daypart pairs

    %NOTE:leaving out X3,X4 for the moment. No same program pairs and no
    %'same-program-type' pairs
```

```
    %the following section generates a list of the ratings of the units
    %Will be used to compute X5 and X6
    list=[];
    for w=1:L
        %pull the hour, day, and channel of each unit and reference the
        %ch-d-hr average rating information to retrieve the rating of the
        %unit
        hr=find(strcmp(filled_slot_info(w,1),ch_d_h_av_rating(:,3))==1);
        day=find(strcmp(filled_slot_info(w,2),ch_d_h_av_rating(:,2))==1);
        ch=find(strcmp(filled_slot_info{w,3},ch_d_h_av_rating(:,1))==1);
        hr_day=intersect(hr,day);
        hr_day_ch=intersect(hr_day,ch);
        list=[list; ch_d_h_av_rating{hr_day_ch,5}];
    end

    X5=mean(list);
    %^average rating of a spot in the schedule
    X6=std(list);
    %^variance of the ratings of spots in the schedule
    X7=L;
    %^number of spots in the schedule

    XX=[XX;1+X1,1+X2,X5,X6,X7];
    V_star=X5*(1-X5); %from the paper we can approximate V* by mu(1-mu) when mu is small
    VV_star=[VV_star; V_star];
end
XX_hat=XX;
V_star_hat=VV_star;
XX=[ones(1,M);log(XX)'];
%^XX is the matrix of all of the different input values for the multivariate
%regression equations for the corresponding duplication value in Y_values
VV_star=log(VV_star);

B=inv(XX'*XX)*XX'*VV_star;
%^computes the least squares solution to the linear regression problem
log_const=B(1);
B(1)=exp(B(1));

end
```

C Estimate the Reach and Frequency Distribution for a new Schedule

```
function [frequency_dist,effective_reach]=approach_2_reach_calc(B,schedule,
ch_d_h_av_rating,reach_threshold,channels)
%approach_2_reach_calc computes the reach/effective reach of a schedule
%accoring to the coefficients present in B and the characteristics of the
%schedule.
%Inputs:    -B is a vector of the coefficients output from multiplicative
%           regression.m
%           -schedule is the (24)x(7)x(number channels) binary array
%           corresponding to placement of slots.
%           -ch_d_h_av_rating is the channel, day, hour,average rating as
%           computed by the function importnielsen_approach2
%           -reach_threshold is the number of exposures after which we say
%           a person has been reached.
%           -channels is the cell array of channel labels for the schedule
%Outputs:   -schedule reach is the reach as computed from the BBD where the
%           parameters of the BBD are detemined from the schedule and the B
%           vector
%
%
%% compute the characteristics of the schedule
    load('dayparts.mat') %pull in the matrix containing the lower and upper
                         %bounds of each day part.
    load('hour_labels.mat') %load the hour labels as labeled by nielsen
    load('day_labels.mat')  %load the day labels as labeled by nielsen
    [rows,cols]=find(schedule>=1); %Identify the spots where the ads were
                                   %placed
    sheets=ceil(cols/7); %will be used to identify the channels
    cols=mod(cols,7); %will be used to determine the day of the week
    A=find(cols==0);
    cols(A)=7;
    L=length(rows);
    filled_slot_info=cell(L,3);
    %allocate space for the information about the time slots filled
for j=1:L
    filled_slot_info{j,1}=hours{1,rows(j)}; %hour label
    filled_slot_info{j,2}=days{1,cols(j)};  %day label
    %the following bit is identifying the channel label and space padding
    %with the appropriate number of spaces as is necessary because the
    %channel array just has the letters but the nielsen data is exported
    %with space pads to length 5. I know there is a better way to do this.
    num_char=length(channels{sheets(j)});
    num_space_pad=5-num_char;
    if num_char==5
        filled_slot_info{j,3}=channels{sheets(j)};
    elseif num_char==4
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '});
    elseif num_char==3
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '});
    elseif num_char==2
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '},{' '});
```

```
    elseif num_char==1
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '},{' '},{' '});
    end
end
 %compute the same channel pairs
 C=length(channels);
 same_channel_pairs=0;
for s=1:C
    W=sum(sum(schedule(:,:,s))); %counts the number of ads on each of the
                                 %channels
    %check to see if the number of inserts on the channel is more than two,
    %if it is then you compute the number of possible pairs that happened
    %within that channel. If not, there are no channel pairs.
    if W<2
        same_channel_pairs=same_channel_pairs;
    elseif W>=2
        same_channel_pairs=same_channel_pairs+nchoosek((W),2);
    end
end
%the proportion of possible ad pairs that happened on the same channel
 X1=same_channel_pairs/nchoosek(L,2);

 %compute the proportion of pairs that are placed in the same daypart
 same_day_part_pairs=0;
for k=1:5
        %count the number of insertions within a given day part.
        day_part_slots=sum(sum(sum(schedule(dayparts(k,1):dayparts(k,2),:,:))));
        %check to see if the number of placesments in the daypart is
        %greater than 2 otherwise there are no within daypart pairs for
        %that daypart
        if day_part_slots<2
            same_day_part_pairs=same_day_part_pairs;
        elseif day_part_slots>=2
        same_day_part_pairs=same_day_part_pairs+nchoosek(day_part_slots,2);
        end
end
%proportion of total pairs that are places in the same daypart.
    X2=same_day_part_pairs/nchoosek(L,2);
    %proportion of same daypart pairs

    %NOTE:leaving out X3,X4 for the moment. No same program pairs and no
    %'same-program-type' pairs

    %pull the average rating (averaged over time from the nielsen data
    %of each of the filled time slots from ch_d_h_av_rating
    list=[];
    for w=1:L
        %use the day, hour, and channel to pull the average rating
        hr=find(strcmp(filled_slot_info(w,1),ch_d_h_av_rating(:,3))==1);
        day=find(strcmp(filled_slot_info(w,2),ch_d_h_av_rating(:,2))==1);
        ch=find(strcmp(filled_slot_info{w,3},ch_d_h_av_rating(:,1))==1);
        hr_day=intersect(hr,day);
        hr_day_ch=intersect(hr_day,ch);
        list=[list; ch_d_h_av_rating{hr_day_ch,5}];
```

```
    end
    %compute the average rating of the schedule slots
    X5=mean(list);
    %compute the variance of spot ratings
    X6=std(list);
    %number of slots in the schedule, again, in the form of the paper
    %notation
    X7=L;

   %% Compute the V* for the schedule
   %formula for V* given the coefficients determined in multiplicative
   %regression.
   V_star=exp(B(1))*(1+X1)^(B(2))*(1+X2)^(B(3))*(X5)^(B(4))*(X6)^(B(5))*(X6)^(B(6));
   mu=X5;
   %the equations for the parameters from V* and mu
   %NOTE: I am getting negative alphas and betas because V* is larger than
   %mu*(1-mu)... this makes it infeasible to compute reach..... how to work
   %around this. I wonder if my average rating data is incorrect?
   alpha=abs((mu*V_star)/(mu*(1-mu)-V_star));
   Beta=abs((alpha*(1-mu))/mu);
   %based on the reach threshold we have to identify which frequencies we
   %say count as reach
   frequencies=0:L;
   frequency_dist=zeros(1,L+1);
   for i=1:L+1;
       k=frequencies(i);
       %add the percentage of people exposed to k viewings of the schedule
       %using the closed form of the beta binomial distribution. If we sum
       %the percentages that didnt see the needed frequencies we can
       %subtract that from one to get the reach
       frequency_dist(i)=nchoosek(L,k)*beta(k+alpha,L-k+Beta)/beta(alpha,Beta);
   end
   %reach is sum of frequency distribution values greater than or equal to
   %the threshold
   effective_reach=sum(frequency_dist(1,reach_threshold+1:end));
end
```

D Import a Single Week CSV File from Nielsen Data

```
function weeklongviewership = import_nielsen_week(filename)
%IMPORTFILE Import numeric data from a text file as a matrix.
%   WEEKLONGVIEWERSHIP = IMPORTFILE(FILENAME) Reads data from csv file
%   FILENAME containing the weeklong viewership data from nielson.


%% Initialize variables.
delimiter = ',';
    startRow = 2;
    endRow = inf;


%% Format string for each line of text:
%  all columns text (%s)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%s%s%s%s%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');


%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,
 'Delimiter', delimiter, 'EmptyValue' ,NaN,'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-startRow(block)+1,
    'Delimiter', delimiter, 'EmptyValue' ,NaN,'HeaderLines', startRow(block)-1, 'ReturnOnError', false);
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Create output variable
weeklongviewership = [dataArray{1:end-1}];
end
```

E Compute the Reach and Frequency Distribution from a Single Week of Data

```
function [frequency_dist, reach]=hist_data_schedule_freq(schedule,
 weeklongviewership,channels,UE_target_demo)
%hist_data_schedule_freq computes the frequency distribution of a schedule
%given a week of historical data from within the window of time used to
%generate the random schedules for the multiplicative regression model.
%Inputs:        -schedule: a schedule (for now randomly generated)
%               -weeklongviewership: the output of import_nielsen_week.m
%               -reach_threshold: the threshold for defining effective
%               reach.
%               -channels: the cell array of channel labels for the
%               schedule
%               UE_target_demo:nielsen UE for the target demo
%Outputs:       -frequency_dist: the freqeuncy distribution based off the
%               historical data.
%               -effective_reach: the effective reach of the schedule based
%               on the reach threshold defined as an input.
%start and end index for each of the five Nielsen dayparts
load('dayparts.mat')
%nielsen hour labels
load('hour_labels.mat')
%nielsen day labels
load('day_labels.mat')

C=length(channels);

[rows,cols]=find(schedule==1); %identify the spots
sheets=ceil(cols/7); %will be used to identify the channels
cols=mod(cols,7); %will be used to determine the day of the week
A=find(cols==0);
cols(A)=7;
L=length(rows); %number of units in schedule
filled_slot_info=cell(L,3);
%^allocate space for the information about the time slots filled
for j=1:L
    filled_slot_info{j,1}=hours{1,rows(j)}; %hour label
    filled_slot_info{j,2}=days{1,cols(j)};  %day label
    num_char=length(channels{sheets(j)});
    num_space_pad=5-num_char;
    %the next few lines zeropad to get the appropriate channel name based
    %on a characteristic of exporting the data from nielsen
    if num_char==5
        filled_slot_info{j,3}=channels{sheets(j)};
    elseif num_char==4
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '});
    elseif num_char==3
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '});
    elseif num_char==2
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '},{' '});
    elseif num_char==1
        filled_slot_info{j,3}=strcat(channels{sheets(j)},{' '},{' '},{' '},{' '});
```

```
    end
end
%identify the unique people who watched during the week
household_person_combos=strcat(weeklongviewership(:,1),weeklongviewership(:,2));
unique_viewers=unique(household_person_combos);

%allocate space for the number of units each unique viewer saw
person_counts=zeros(length(unique_viewers),1);
%keep track of the weight for each person who saw some number of units
weights=zeros(length(unique_viewers),1);
for n=1:L
for m=1:length(unique_viewers)
    %find the shows that each unique viewer saw
    person_shows=find(strcmp(household_person_combos,unique_viewers(m))==1);

    slot=strcat(filled_slot_info(n,1),filled_slot_info(n,2), filled_slot_info{n,3});
    shows=strcat(weeklongviewership(person_shows,4),
     weeklongviewership(person_shows,5), weeklongviewership(person_shows,3));
    %check whether or not each time slot appears in the persons show list
    W=intersect(slot,shows);
    [a,b]=size(W);

    if a+b>1
        %if the viewer saw the unit add one to their count
        person_counts(m,1)=person_counts(m,1)+1;
        X=find(strcmp(W,shows)==1);
        weight=str2num([weeklongviewership{person_shows(X),6}]);
        weights(m,1)=weight;
    elseif a+b<=1
        %if they didn't see it, continue
        continue
    end
end
end
%determine the possible number of units a person could have seen. We don't
%count them if they didnt see any of them.
frequencies=unique(person_counts);
frequencies=frequencies(find(frequencies>0));

%identify the people with each possible frequency and sum up the weights of
%those viewers and divide by the UE to get the percentage.
for k=1:length(frequencies)
    percentages(k)=sum(weights(find(person_counts==frequencies(k))))/UE_target_demo;
end
%defining reach based on 1+ frequencies
reach=sum(percentages);
%the percentage that saw 0 is 1 minus the sum of people who saw it at least
%once.
frequency_dist=[1-sum(percentages), percentages];

end
```

F Validation Script

```
UE_target_demo=117620+109040;  %21+
UE_target_demo=UE_target_demo*1000;
load('discov_channels.mat')
filename='ch_d_hr_viewing_21_plus.csv';
[ch_d_h_av_rating]=importnielsen_approach2(filename, channels, UE_target_demo);

num_schedules=10000;
[B,log_const,XX_hat,V_star_hat]=multiplicative_regression(ch_d_h_av_rating, channels, num_schedules);

filename1='week_long_viewership_21_plus.csv';
weeklongviewership = import_nielsen_week(filename1);

trials=25;
errors=zeros(trials,1);
reach_threshold=1;
for i=1:trials
chance=rand(24,7,length(channels));
schedule=zeros(24,7,C);
schedule(find(chance>.99))=1;
[frequency_dist,effective_reach]=approach_2_reach_calc(B,schedule,
ch_d_h_av_rating,reach_threshold,channels);
[frequency_dist_hist, reach_hist]=hist_data_schedule_freq(schedule,
weeklongviewership,channels,UE_target_demo);
errors(i)=reach_hist-effective_reach;
end

close all
figure,hold all
plot(1:10, errors,'b*')
mu=mean(errors);
VAR=std(errors);
plot(1:10,mu*ones(10,1),'r')
plot(1:10, (mu+1.97*VAR)*ones(10,1),'g')
plot(1:10, (mu-1.97*VAR)*ones(10,1),'g')

figure, hold all
plot(0:length(frequency_dist)-1,log(frequency_dist),'b')
plot(0:length(frequency_dist_hist)-1,log(frequency_dist_hist),'r')
```