# An Agent-Based Model of Information Diffusion

Mid-Year Report

Neza Vodopivec
Applied Math and Scientific Computation Program
nvodopiv@math.umd.edu

Advisor: Dr. Jeffrey Herrmann
Mechanical Engineering Department
jwh2@umd.edu

**Abstract**: Understanding how information spreads throughout a population can help public health officials improve how they communicate with the public in emergency situations. In this project, I implement an agent-based information diffusion model inspired by the Bass model. I compare my discrete-time simulation to a traditional differential-equation conceptualization of the Bass model. Finally, I test my model by seeing how well it predicts the real-life spread of information through a Twitter network.

## 1. INTRODUCTION

**Motivation.** In the weeks following the events of 9/11, seven letters containing dangerous strains of *Bacillus anthracis* were mailed to senators and news agencies. Although the FBI never determined a sender or motive, the attacks informed the country to the possibility of bioterrorism and spurred public health agencies to plan out responses to similar, larger-scale scenarios. Anthrax is not contagious, but its dynamics require a fast dissemination of targeted public health information because newly infected individuals have a far better prognosis when they are treated quickly. To increase effectiveness of a targeted public health message, its broadcasters must understand how information spreads through a population.

**Traditional models of information diffusion.** The goal of an information diffusion model is to describe how a piece of information spreads through a given population over time. We are interested in the successive increases in the fraction of people who are aware of the information. Traditionally, information diffusion has been modeled with differential equations that describe the dynamics of a global system -- in this case, an entire population. A disadvantage of such models is that they describe only aggregate diffusion patterns, not taking into account that individuals behave in complex ways and that they function within social networks.

**A different approach: agent-based models.** Recently, bottom-up modeling in the form of agent-based simulation has gained attention. Agent-based models capture how patterns of behavior at the macro level emerge as the result of the interactions of individuals, or agents, at the micro level. Agent-based models are discrete-time simulations of the interactions in an ensemble of autonomous agents. At each iteration, each agent evaluates its situation and makes decisions according to a ruleset.

In my project, I create an agent-based information diffusion model. I compare my discrete-time simulation to an analytical differential equation model. Finally, I test how well my model predicts the real-life spread of information through a Twitter network.

## 2. APPROACH

**The Bass model.** The Bass model (Bass, 1969) was originally developed by a marketer to model brand awareness, but it can also be applied more generally to the diffusion of information. The model is based on the assumption that people get their information from two sources, advertising and word of mouth.

**Formulation.** The Bass model describes the fractional change in a population's awareness of a piece of information by:

$$\frac{F'(t)}{1 - F(t)} = p + qF(t)$$

$$F(0) = 0,$$

where $F(t)$ is the aware fraction of the population as a function of time, $p$ is the advertising coefficient, and $q$ is the word-of-mouth coefficient.

We can express F(t) directly as:

$$F(t) = \frac{1 - e^{-(p+q)t}}{1 + \frac{p}{q} e^{-(p+q)t}}$$

The Bass model can be interpreted a hazard-rate model, where P(t) = p + qF(t) is the conditional probability that a person will become aware of information at time t given that they are not yet aware.

**An agent-based Bass model.** We can formulate an agent-based model inspired by the classical Bass model. First, we discretize the problem, giving agents an opportunity to become aware of the information (given that they are not yet aware) at each time step. Then, instead of taking a deterministic time aggregate at each time step, we update each agent's state probabilistically. Finally, we consider agents within the context of a social network: instead of allowing each agent to be influenced by the entire population, it is influenced only by its direct neighbors.

**Information diffusion through a Twitter network.** In my project, I implement an agent-based Bass model that simulates the diffusion of information through a Twitter network. (Twitter is a service which allows its users to post short messages and list which other users they read, or "follow".) In this case, each agent corresponds to a Twitter user. A word-of-mouth transfer of information represents the exchange of information in the form of a Twitter post. The effect of advertising is any external transfer of information, that is, information obtained from a source other than Twitter. We define a Twitter user to be aware when he or she posts a message that conveys the relevant piece of information to followers.

**Network formation.** The agent-based Bass model assumes agents are arranged in some fixed, known network. Formally, the network is a directed graph with agents as its nodes. An agent's neighbors are those who connect to it. The network structure for my simulations will be derived from real-world Twitter data. A directed edge from agent i to agent j denotes that agent j "follows" agent i on Twitter.

**The spread of information through the network.** The agent-based Bass model is a discrete-time model in which each agent has one of two states at each time step t: (1) unaware or (2) aware. At the beginning of the simulation, all agents are unaware. At each time step, an unaware agent has an opportunity to become aware. Its state changes with P, the probability that it becomes aware due to advertising or due to word of mouth. The probability that an agent becomes aware due to word of mouth increases as a function of the fraction of its neighbors who became aware in previous time steps. Once an agent becomes aware, it remains aware for the rest of the simulation.

**Probability that an agent becomes aware.** At each iteration, an unaware agent i becomes aware with probability

$$P_i(t) = p\,\Delta t + q\,\Delta t\,[n_i(t)/m_i] - (p\,q\,\Delta t^2\,[n_i(t)/m_i]),$$

where $m_i$ is the number of neighbors of agent i, $n_i(t)$ is the number of neighbors of agent i that became aware before time t; and p and q are parameters which indicate the effectiveness of advertising and word of mouth per unit of time, respectively. The first term is the probability that an agent becomes aware due to advertising, the second term that it becomes aware due to word of mouth, and the third term that it becomes aware due to both.

## 3. IMPLEMENTATION

**Motivation.** In their paper, "Agent-Based Models of Information Diffusion", Auzolle and Herrmann (2012) describe their implementation of an agent-based diffusion simulation. The codebase, written in NetLogo (Tisue and Wilensky, 2004), a programming language used to develop agent-based simulations, turned out not to be fast enough to handle large networks. The goal of the current project is to code this model in MATLAB (The MathWorks Inc., 2010) with the hope of producing a faster, more memory efficient implementation. I implemented two versions of this model. First, I coded a basic implementation to use as a reference. Then, I implemented the model using a more efficient updating rule and taking advantage of sparse data structures. I call this second implementation the neighbor-set implementation.

**Basic implementation.** The basic implementation depends on the use of an adjacency matrix to store relationships between agents and to record agents' awareness statuses. A straightforward algorithmic description of the basic simulation is as follows:

Arbitrarily identify the N agents with the set 1, …, N. Let D denote the $|D| \times 2$ matrix listing all (directed) edges of the graph as ordered pairs of nodes.

INPUT: matrix D, parameters p and q.

1. Keep track of the state of the agents in a length-N bit vector initialized to all zeros.
2. Create an adjacency matrix A such that A(i,j) = 1 if (i,j) is a directed edge in D and 0 otherwise.
3. Create another adjacency matrix B to track aware neighbors. B(i,j) = 1 if (i,j) is a directed edge in D **and** agent i is aware.
4. At each time step, for each agent:
    I. Check the bit vector to determine if the agent is already aware. If so, skip it.
    II. Make the agent newly aware with probability p.
    III. Look up the agent's upstream neighbors in A. Look up the agent's aware upstream neighbors in B. Determine what fraction of the upstream neighbors are aware. Make the agent newly aware with probability q times that fraction.
    IV. Once all agents have been processed, record the newly aware ones as aware in the bit vector.
5. Stop once all agents have become aware, or after a maximum number of iterations.

OUTPUT: complete history of the bit vector.

**Neighbor-set implementation.** This implementation benefits from a more efficient updating rule and from custom sparse data structures tailored to this new updating rule. The result is a faster run time and more efficient use of memory. When run with the same random numbers, the basic implementation and the neighbor-set implementation produce identical results.

**A more efficient updating rule.** In order to decide whether to change the status of an unaware node, the node's number of unaware upstream nodes (its "awareness number") must be computed. The basic implementation effectively recomputes each node's awareness number from scratch at every time step. But because changes in the awareness number are entirely due to nodes which have just become aware, such a computation seems wasteful. This suggests a possible improvement: a preliminary pass through just the newly-aware nodes which updates just their downstream nodes. After this preliminary step, we can proceed as in the basic implementation, but without needing to recompute awareness numbers.

**Representing adjacency efficiently.** Our new updating procedure suggests a further possible improvement: replacing the network's adjacency matrix with a sparse data structure which reflects the structure of the updating rule. Information about adjacency can be stored by rewriting the adjacency relation as a function $f: V \rightarrow 2^V$ which returns a node's downstream nodes. Concretely, this function is most naturally implemented as a vector of length |D| concatenating the output sets of $f$ together with a list of pointers marking the start of each set. Note that the coding of this function can also be thought of as an |D| x 2 ordered list of the coordinates of the nonzero entries in the original adjacency matrix.

**Comparing the three implementations.** Table 1 gives the time and space requirements for the NetLogo (Auzolle and Herrmann, 2012), basic, and neighbor-set implementations of the agent-based Bass model for two datasets. The second column lists the time required for a single run of the simulation. Column three lists the total time required to run the simulation 100 times and then compute confidence intervals. Data listed for the NetLogo implementation is an estimate as the program is run in two stages. Moreover six out of the ten times it was executed, this program was stopped after running for longer than an hour. The best runtime (out of ten) was taken for the NetLogo implementation for each dataset.

Table 1: Time and Space Efficiency Model Implementations

| | Bin Laden Dataset (7.27 MB, 4.7K nodes, 477K edges) | | |
|---|---|---|---|
| | Memory | Sim. Time | Total Time |
| NetLogo Implementation | -- | ~ 3 min. | -- |
| Basic Implementation | 506.73 MB | 12.92 sec. | 21.0 min. |
| Neighbor-Set Implementation | 18.55 MB | 0.74 sec. | 1.2 min. |

| | Irene Dataset (0.70 MB, 1.1K nodes, 46K edges) | | |
|---|---|---|---|
| | Memory | Sim. Time | Total Time |
| NetLogo Implementation | -- | ~ 10 sec. | -- |
| Basic Implementation | 30.31 MB | 0.73 sec. | 1.2 min. |
| Neighbor-Set Implementation | 1.85 MB | 0.06 sec. | 6.5 sec |

**Increasing code efficiency further through parallelization.** The current version of my neighbor-set implementation runs with a single thread of execution. In the spring semester, it will be parallelized so that multiple simulations run simultaneously. Each of the runs will be logged and the complete set of data will then be analysed.

## 4. SIMULATION RESULTS

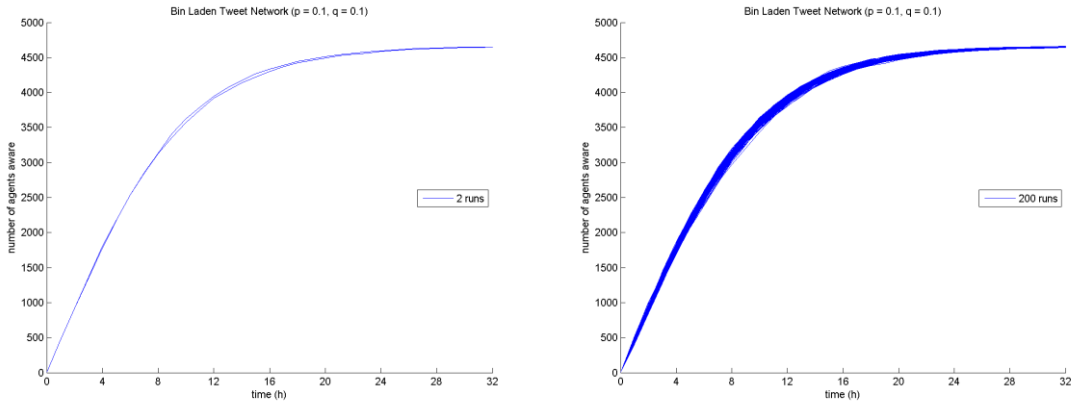


Figure 1: Plots of Simulation Results

The plots above show the number of agents aware at each time step of the simulation. Figure 1a shows the trajectories for two simulation executions; figure 1b shows trajectories for 200 executions. Since the simulation is stochastic, a single run provides, at each time step, only a sample of the network's true behavior – the trajectories are close but not the same.

## 5. STATISTICAL ANALYSIS

Results as random variables. As illustrated in the plots above, at each time step, our algorithm produces not a number but rather a random variable. We would like to know as much as possible about the underlying distribution of the network's behavior at each fixed time. The most important single number we could calculate for each time-indexed underlying distribution would be its mean. We could learn about this mean by computing a sample mean, then surrounding it with a confidence interval within

which we expect the true mean to lie. Another way we might gain insight about the underlying distribution is by constructing prediction intervals within which we think our next sample is likely to fall.

**Confidence intervals for the distribution of means.** We can summarize how confident we are that an underlying distribution's true mean $\mu$ lies within a given range by introducing the notion of a confidence interval. Every run of the simulation gives us a sample value $x_i$ at each time step. If we run the simulation numerous times, we obtain many sample values from which we can compute a sample mean $\bar{x}$.

Let $x_1, \ldots, x_n$ be a random sample from an arbitrary distribution with an unknown mean $\mu$ and a standard deviation $\sigma > 0$. A 95% confidence interval for the unknown mean $\mu$ is an interval with random endpoints $u(X)$ and $v(X)$ such that $P(\mu \in (u(X), v(X))) = 0.95$.

Let $\bar{x} = \frac{x_1 + \cdots + x_n}{n}$ be our sample mean at a given time step for $n = 200$ runs of the simulation. The Central Limit Theorem states that the random variable $W = \frac{\bar{x} - \mu}{\sigma/\sqrt{n}}$ has the normal distribution $N(0,1)$ in the limit as $n \to \infty$. In other words, the distribution of a mean tends to be normal, even when the distribution from which the mean is computed is decidedly non-normal.

Since the standard deviation $\sigma$ of our distribution is unknown, we can approximate it with the sample variance $S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$ given that our underlying distribution is not 'significantly' skewed or contains 'too many' outliers. We make this assumption and examine it in a later section.

Because the Central Limit Theorem says that the sample mean is approximately normally distributed, for an unknown mean μ:

(1) $W = \frac{\bar{x} - \mu}{S/\sqrt{n}}$

will be approximately standard normal. So we have

(2) $P(-1.96 < W < 1.96) = 0.95.$

Substituting (1) into (2) and solving for μ we have:

$$P\left(\bar{x} - 1.96\frac{S}{\sqrt{n}} < \mu < \bar{x} + 1.96\frac{S}{\sqrt{n}}\right) = 0.95.$$

The rearranged bounds are known as a 95% confidence interval for the mean.

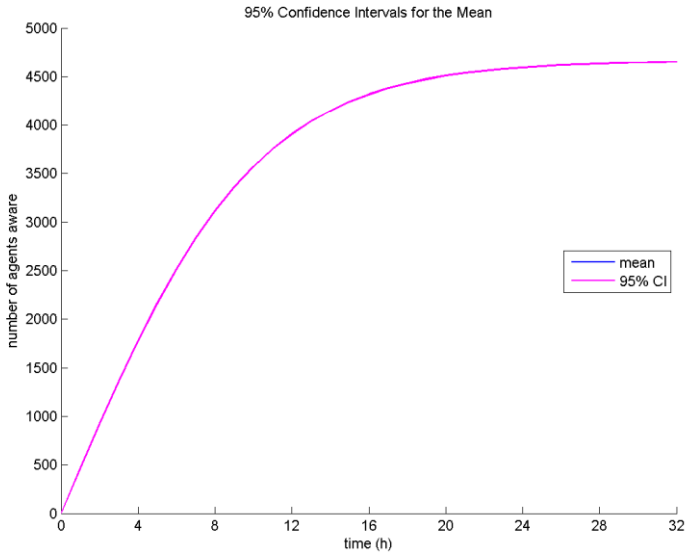


Figure 2: Examining Confidence Intervals Surrounding the Simulation Mean at Each Time Step

Confidence intervals about the mean are so narrow, it is difficult to see them on the plot. We are 95% confident the true mean lies in this small interval.

**Prediction intervals.** We can use our past observations to predict the outcome if we were to run the simulation one more time. Let $x_1, \ldots, x_n, x_{n+1}$ be a random sample from a with an unknown mean and an unknown standard deviation. A 95% prediction interval for $x_{n+1}$ is an interval with random endpoints $u\ X$ and $v\ X$ such that $P\ x_{n+1} \in (u\ X, v\ X) = 0.95$.

**Nonparametric prediction intervals for unknown distributions.** If we are sampling from an unknown distribution, we might try using quantiles to form our prediction intervals. Let $F\ x = P\ X \leq x = p$ be the cumulative distribution function for a random variable $X$. The quantile function $Q(p)$ is the inverse distribution function $F^{-1}\ p = \inf\ x \in R\ p \leq F(x)$, defined so it holds for both continuous and discrete distributions. Simply put, the quantile function gives the value of a random variable given the probability of obtaining at most that value.

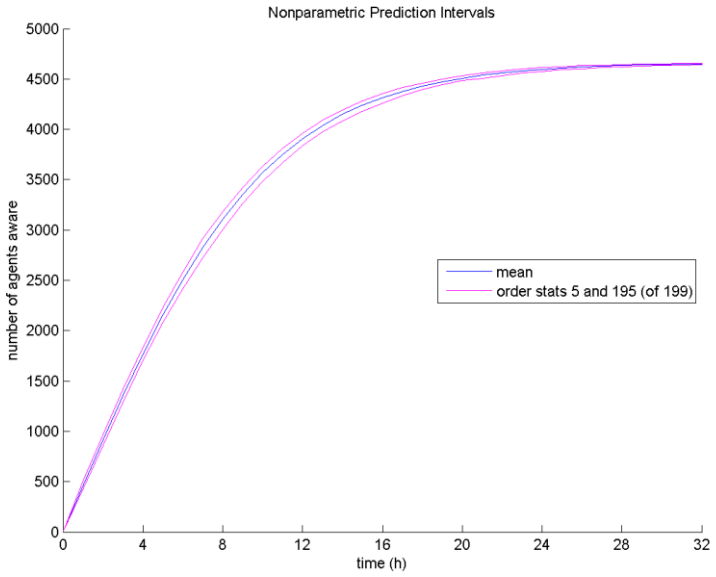


Figure 3: Nonparametric Prediction Intervals at Each Time Step of Simulation

**Parametric prediction intervals for unknown distributions.** Let's assume that the simulation results are normally distributed at each time step. This assumption allows us to analyze them using parametric prediction intervals.

Let $x_1, \ldots, x_n, x_{n+1}$ be a random sample from a normal distribution with an unknown mean and an unknown standard deviation. Again, let $\bar{x} = \frac{x_1 + \cdots + x_n}{n}$ be the sample mean and

$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$ be the sample variance.

To obtain a prediction interval for $x_{n+1}$, we first use the fact that the random variable $T = \frac{\bar{x} - x_{n+1}}{S \sqrt{1+\frac{1}{n}}}$ has a $t$ distribution with $n-1$ degrees of freedom with a probability density function given by

$f_T(t) = \frac{\Gamma[(v+1)/2]}{\sqrt{\pi v}\,\Gamma(v/2)} \left(1 + \frac{t^2}{v}\right)^{-(v+1)/2}$, where $\Gamma$ is Gamma function and $v = n-1$ is the number of degrees of freedom.

Since we know the distribution of $T$, we can compute $t_{2.5}$ the 2.5$^{th}$ percentile of the t distribution with $n-1$ degrees of freedom such that:

$$P\left(-t_{2.5} < \frac{\bar{x} - x_{n+1}}{S \sqrt{1+\frac{1}{n}}} < t_{2.5}\right) = 0.95$$

We solve the inequality for $x_{n+1}$:

$$P\left(\bar{x} - t_{2.5}S\sqrt{1+\frac{1}{n}} < x_{n+1} < \bar{x} + t_{2.5}S\sqrt{1+\frac{1}{n}}\right) = 0.95$$

The rearranged bounds are known as a 95% parametric prediction interval for the mean.

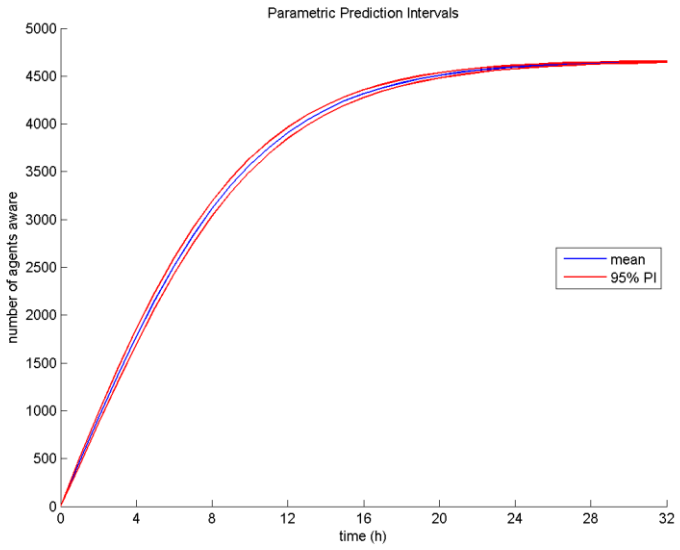


Figure 4: Parametric Prediction Intervals at Each Time Step of Simulation

Did the assumption that our results at each time step come from a normal distribution significantly change the prediction intervals? Figure 5 shows that that the nonparametric and the parametric prediction intervals nearly overlap – in this case, the normality assumption didn't produce significantly different results.

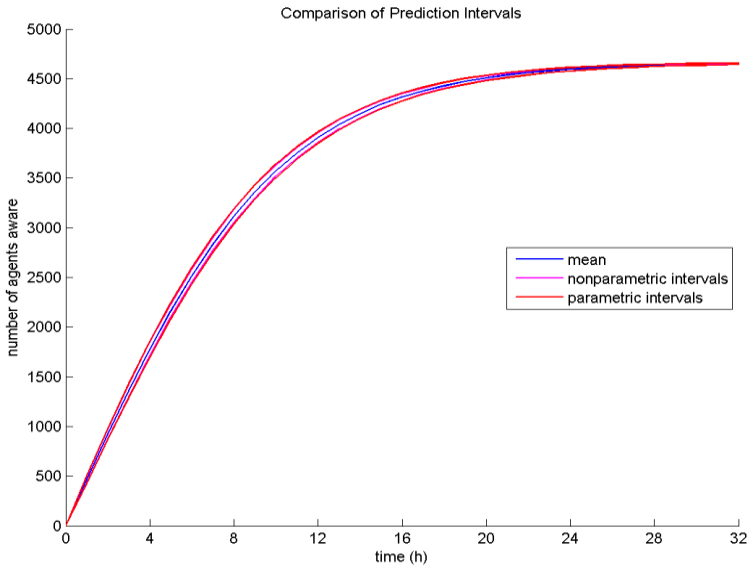


Figure 5: From Nonparametric to Parametric Prediction Intervals:
Did our assumption of normality significantly change the bounds?

**Challenging the assumptions.** In order to use confidence intervals and parametric prediction intervals to analyze the simulation results, several assumptions were made about the underlying distributions at each time step. Computing prediction intervals, required the assumption that our sample of results comes from a normal distribution. Computing confidence intervals, requires that the distribution is not significantly skewed and does not contain too many outliers. To what extent are these assumptions justified? We use quantile-quantile plots and perform the Kolmogorov-Smirnov goodness-of-fit test to explore the answer.

**Quantile-quantile plots.** A q-q plot compares the shapes of two distributions by plotting their quantiles against each other. This method is often used to determine if the underlying distribution of a data sample matches a particular theoretical distribution. We use q-q plots for two purposes: (1) to graphically represent how closely the underlying distribution in fact resembles a normal distribution, and (2) to test how skewed the underlying distribution is and whether it has outliers.

Formally, let $Q_1(p)$ and $Q_2(p)$ be quantile functions for two distributions respectively. A q-q plot is a parametric curve with coordinates $(Q_1(p), Q_2(p))$ and quantile intervals as the parameter. All q-q plots were generated with MATLAB's 'qqplot' command.

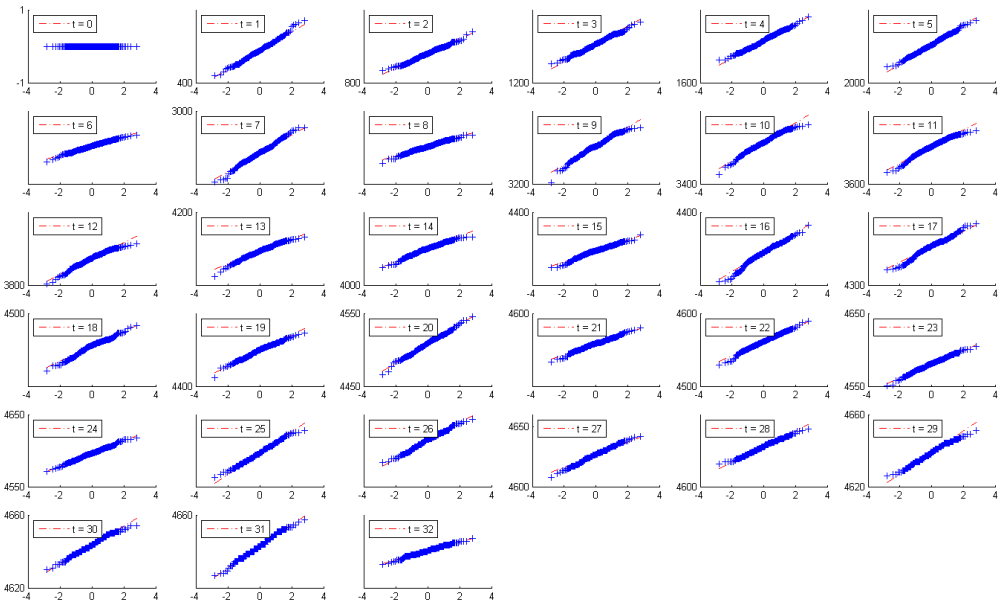


Figure 6: Quantile-Quantile Plots

Figure 6 shows that most points lie roughly on the line $= x$, suggesting that the distributions of the simulation results at most time steps are close to normal. There are a few points that lie off the line, suggesting outliers in the data. Almost all slopes are very close to 1, indicating that the data is not significantly skewed. A trajectory with the left end below the line and right end above the line indicates longer tails at the ends of the distributions at some time steps. A stair-like pattern at the later time-steps reflects the discrete nature of the data.

**Kolmogorov-Smirnov Test.** Computes $L^{\infty}$ norm between cdf of normal distribution and underlying distribution at each time step.

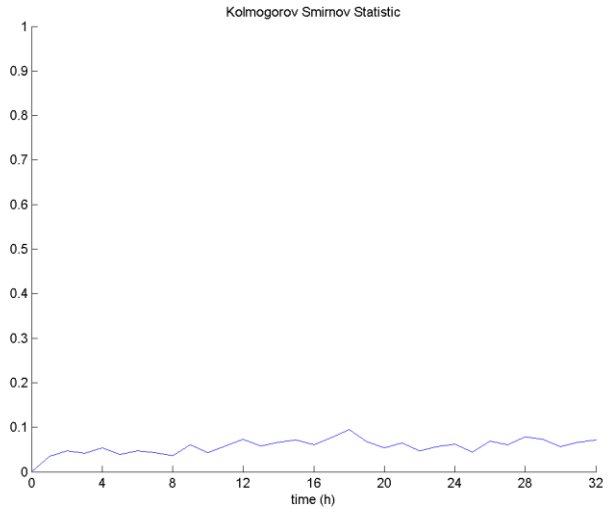


Figure 7: Kolmogorov-Smirnov Statistic at Each Time Step

## 6. VALIDATION

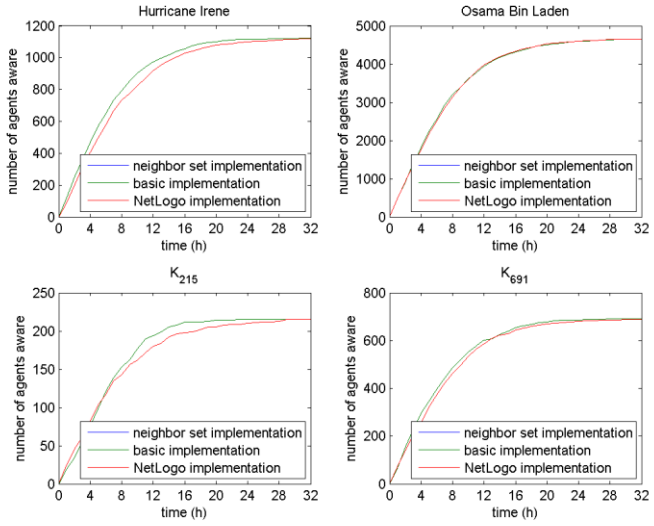


Figure 8: Mutual Validation among Three Implementations

Let's focus on the case of a fully-connected network K. Let $x(t)$ denote the fraction of agents who are aware at time $t$. If the probability that advertising makes an agent aware is $p$ and the independent probability that word of mouth makes the agent aware is $qx$, then, by the Inclusive-Exclusive Principle, the probability that an agent becomes aware is:

$$a\ t\ = p + qx\ t\ - pqx(t).$$

Estimating the fraction of agents who become aware over a single time step as the probability of becoming aware $a\ t$ times the fraction $(1 - x\ t\ )$ of unaware agents, we arrive at the recurrence relation:

$$x_{n+1} = x_n + a_n\ 1 - x_n\ = x_n + [p + qx_n - pqx_n][1 - x_n],$$

with the initial condition $x_0 = 0$.

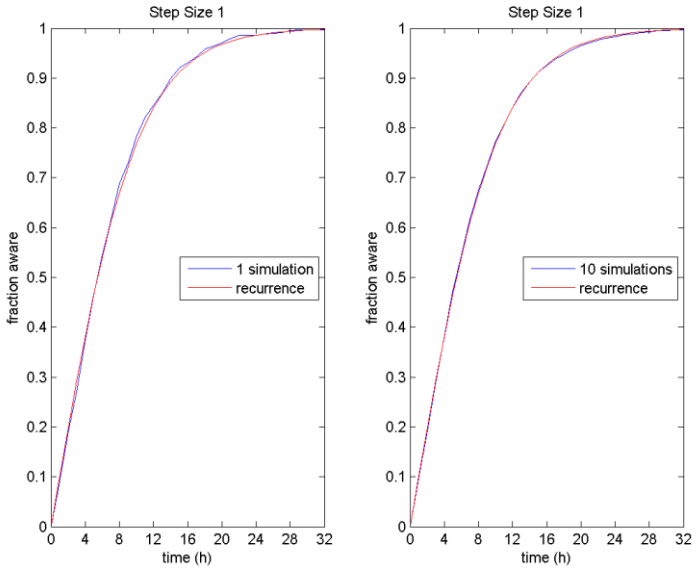


Figure 9: The solution to the recurrence indeed tracks the output of the simulation, suggesting that the implementation is correct.

Our simulation assumes an advertising probability of $p$ and a word-of-mouth probability of $qx$ over a time step of 1 hour. If we wish to rework the simulation to run over some smaller time step $\Delta t$, we must reimagine $p$ and $q$ as probabilities per hour and instead write out advertising probability as $p\ \Delta t$ and word-of-mouth probability as q $\Delta t\ x$. Inserting $\Delta t$ this way into the previous recurrence, we obtain:

$$x_{n+1} = x_n + [p\Delta t + q\Delta t x_n - p\Delta t q\Delta t x_n][1 - x_n],$$

again starting with $x_0 = 0$.

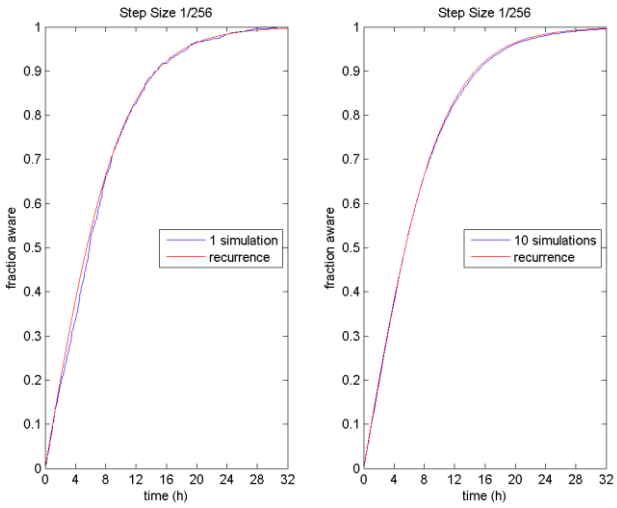


Figure 10: The solutions to the recurrence track our simulation over a range of step sizes.

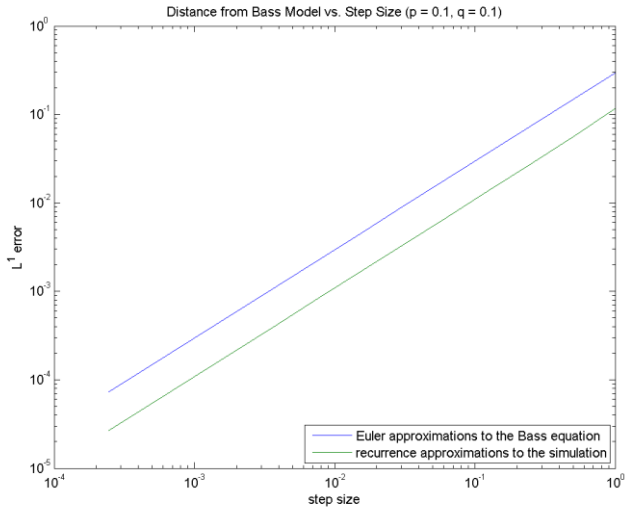


Figure 11.

The recurrence schemes are globally first-order approximations to the solution of the original logistic ODE (the Bass model).

## 7. PROJECT SCHEDULE AND MILESTONES

I have completed the following components of my project:

October: Develop basic simulation code. Develop code for statistical analysis of results.

November: Validate simulation code by checking corner cases, sampled cases, and by relative testing. Validate code against analytical model.

December: Validate simulation against existing NetLogo implementation. Prepare mid-year presentation and report.

January: Investigate efficiency improvements to code. Incorporate sparse data structures.

## 8. DELIVERABLES

I have completed all deliverables promised in my proposal for the end of the semester:

1. the code for my simulation
2. the code for my statistical analysis
3. a plot showing at each time step the mean and both ends of a 95 percent confidence interval based on data collected from numerous runs of the simulation
4. a comparison of my code's running time against that of the existing NetLogo implementation

## 9. REFERENCES


Auzolle, Ardechir and Herrmann, Jeffrey (2012). "Agent-Based Models of Information Diffusion". Working paper, University of Maryland, College Park, Maryland.

Bass, Frank (1969). "A new product growth model for consumer durables". *Management Science* 15 (5): p. 215–227.

Chandrasekaran, Deepa and Tellis, Gerard J. (2007). "A Critical Review of Marketing Research on Diffusion of New Products". *Review of Marketing Research*, p. 39-80; Marshall School of Business Working Paper No. MKT 01-08.

Dodds, P.S. and Watts, D.J. (2004). "Universal behavior in a generalized model of contagion". *Phys. Rev. Lett.* 92, 218701.

MATLAB version 7.10.0. (2010) Natick, Massachusetts: The MathWorks Inc.

Mahajan, Vijay, Muller, Eitan and Bass, Frank (1995). "Diffusion of new products: Empirical generalizations and managerial uses". *Marketing Science* 14 (3): G79–G88.

Rand, William M. and Rust, Roland T. (2011). "Agent-Based Modeling in Marketing: Guidelines for Rigor (June 10, 2011)". *International Journal of Research in Marketing*; Robert H. Smith School Research Paper No. RHS 06-132.

Tisue, S. and Wilensky, U. (2004). *NetLogo: A simple environment for modeling complexity.* Paper presented at the Fifth Proceedings of the International Conference on Complex Systems, Boston.