A THESIS FOR THE DEGREE OF MASTER OF SCIENCE

The Development of an Individual-Based Model to simulate the Progress of a Foot-and-Mouth Disease Epidemic

BY

GRIM HWANG

JULY, 2017

INTERDISCIPLINARY PROGRAM IN
AGRICULTURAL AND FOREST METEOROLOGY
THE GRADUATE SCHOOL OF SEOUL NATIONAL UNIVERSITY

The Development of an Individual-Based Model to simulate the Progress of a Foot-and-Mouth Disease Epidemic

GRIM HWANG

INTERDISCIPLINARY PROGRAM IN
AGRICULTURAL AND FOREST METEOROLOGY
THE GRADUATE SCHOOL OF SEOUL NATIONAL UNIVERSITY

ABSTRACT

Disease control strategies including quarantine or vaccination have been used to minimize damages incurred by the foot-and-mouth disease (FMD). Disease spread simulation models can be used for planning such strategies. Still, few models have been developed for detailed simulations of the FMD dispersal, e.g., within a farm and between farms, which would provide information for assessment of a given control strategy. The objective of this study was to

develop the FMD-IBM, an individual-based model (IBM) in order to simulate the intra- and inter-farm dispersal of FMD at a regional scale.

The FMD-IBM model was based on the Reed-Frost equation which determines the probability of infection events between hosts as well as between farms. The parameters affecting the within-farm disease spread process were fitted using the calibration set data generated by the Davis Animal Disease Simulation (DADS) model as little observation data have been open to the public. Parameter values within specific range were evaluated to determine the set of parameters that had the highest degree of agreement for the occurrence dates of events between the FMD-IBM and the DADS model in an intra disease dispersal. Parameterization of regional-level disease spread was also performed using the epidemic case data provided by the World Organisation for Animal Health (OIE). Simulations of the disease dispersal events using the FMD-IBM were performed for two FMD outbreaks including the Andong epidemic in 2010 and Chungcheongnam-do epidemic in 2016, which represent highly severe and slight outbreak of the FMD.

The FMD-IBM provided detailed information on disease dispersal including data specific to a farm, e.g., the date of infection. Such information was used to determine the speed of disease dispersal at each farm. It was found that the rate of the FMD spread ranged from 0.3 to 1.3 km per day at most farms. The parameter values that represent the infectability of a farm were lower in the 2016 epidemic than the 2010 epidemic. Those results suggested that the FMD-IBM could allow preparation of farm specific data for planning and decision-making on control measures against the FMD. Still, the FMD-IBM model had limited ability to predict long dispersal events, which may be improved by adding a new algorithm to take trajectory data into account.

Keywords: FMD (Foot-and-Mouth Disease), IBM (Individual-Based Model), OOP (Object-Oriented Programming), disease spread, simulation model

Student Number: 2014-22946

CONTENTS

LIST OF TABLES AND FIGURES

INTRODUCTION

Foot-and-mouth disease (FMD) is a viral disease of cloven-hoofed animals caused by a ribonucleic acid (RNA) virus in the *Picornaviridae* family (Alexandersen and Mowat, 2005). It is the most important contagious disease known to veterinary science and brings huge economic damage to the livestock industry (Domingo et al., 2002). Since 2005, FMD outbreaks occurred in more than 29 countries worldwide (Table 1). The FMD has occurred occasionally in the Far East regions including China and Mongolia since 2005 (Paton et al., 2010). Korea has also experienced several FMD outbreaks during 2010, 2014, and 2016. In April and November 2010, for example, the South-East Asian (SEA) topotype (Mya-98 lineage) of serotype O affected Korea. As a result of those epidemics, more than 1 million livestock had been destroyed in Korea since 2010.

Damages could be minimized under reasonable control programs for the FMD to protect livestock from contagious diseases (Bicknell et al., 1999). For example, some countries in Africa prevent wildlife from spreading the diseases to livestock farms with fencing and zoning (Vosloo et al., 2002; East, 2002). Australia focuses on blocking the entry of FMD virus at the border with enhanced quarantine system in order to maintain a status of freedom from FMD (Garner et al., 2002). Korea has been dependent on executing preemptive slaughter and culling of infected premises to suppress the disease spread at the initial stage of the epidemic (Joo et al., 2002; Park et al., 2013). Although specific control programs may differ from epidemic cases under different conditions, the importance of quick response has been emphasized in common (Ferguson et al., 2001).

Table 1. List of reported major FMD outbreaks worldwide from 2005 to 2015

Year	Country	Date of Notification	Outbreaks	Resolved Date
2005	China (People's Rep. of)	2005-05-13	9	2005-11-16
	Russia	2005-06-14	14	2005-10-15
	Brazil	2005-10-09	41	2006-04-21
2006	Turkey	2006-02-08	13	2006-09-29
	Egypt	2006-02-15	31	2006-06-10
	Botswana	2006-04-27	32	2006-11-08
	Congo (Dem. Rep. of the)	2006-05-26	12	2007-08-06
2007	Israel	2007-01-02	31	2007-07-01
	China (People's Rep. of)	2007-01-19	9	2008-03-27
	Bolivia	2007-01-26	5	2007-03-28
	Palestinian Auton. Territories	2007-02-15	26	(-)
	Vietnam	2007-07-18	15	(-)
	United Kingdom	2007-08-06	8	2007-12-31
	Botswana	2007-10-17	9	2009-11-30
	Namibia	2007-11-15	23	2008-12-01
2008	Colombia	2008-06-04	4	2008-07-19
	Namibia	2008-08-04	15	2009-01-20
	Lebanon	2009-02-20	11	2009-03-01
2009	Israel	2009-07-13	14	2009-07-12
	South Africa	2009-09-09	4	2010-05-14
2010	China (People's Rep. of)	2010-03-01	26	2012-01-09

	Japan	2010-04-20	292	2010-07-05
	Zimbabwe	2010-06-11	5	2011-05-26
	Mozambique	2010-09-30	11	2011-06-30
	Korea (Rep. of)	2010-11-29	155	2011-04-22
2011	Bulgaria	2011-01-07	12	2011-06-07
	Korea (Dem. People's Rep.)	2011-02-08	139	2011-04-06
	South Africa	2011-02-25	46	2011-07-18
	Israel	2011-04-28	19	2011-09-01
	Kazakhstan	2011-08-22	12	2012-08-21
2012	Libya	2012-01-31	53	2014-12-30
	China (People's Rep. of)	2012-02-22	7	2013-05-02
	Chinese Taipei	2012-03-03	6	2012-04-19
	Egypt	2012-03-14	49	2012-07-27
2013	Zimbabwe	2013-05-27	8	2013-12-23
	China (People's Rep. of)	2013-06-09	10	2015-05-22
	Russia	2013-06-10	17	2013-12-23
	South Africa	2013-08-21	13	2015-02-10
2014	Mongolia	2014-01-31	15	2014-04-04
	Korea (Dem. People's Rep.)	2014-02-19	24	(-)
	Tunisia	2014-04-29	150	2014-11-04
	Zimbabwe	2014-05-16	122	2016-10-14
	Algeria	2014-07-27	419	2014-10-12
	Korea (Rep. of)	2014-12-05	185	2015-05-22
2015	Mongolia	2015-03-06	5	2015-06-03

Algeria	2015-03-10	12	2015-05-03
Namibia	2015-05-13	27	2016-01-25
Zimbabwe	2015-06-23	63	(-)
Botswana	2015-07-31	9	2016-03-31
Morocco	2015-11-02	6	2015-11-13
South Africa	2015-12-11	6	2016-04-22

Note : Data were obtained from the World Animal Health Information System (WAHIS) database (http://www.oie.int/wahis_2/public/wahid.php/Countryinformation/countryhome) (accessed on 2017. 6. 17).

Epidemic models have been used to understand the epidemic dynamics, which would be useful for managing emergency-response programs (Keeling, 2005) (Table 2). For example, Morris et al. (2001) used a spatial-explicit model to assess the effectiveness of control policies for the epidemic occurred in the United Kingdom in 2001. Ferguson et al. (2001) also developed a disease spread model based on epidemic data from the United Kingdom crisis to investigate the spreading pattern of FMD. Keeling et al. (2001) developed Cambridge-Edinburgh model, an individual farm-based model that takes into account heterogeneous landscape and livestock farm conditions of the United Kingdom while simulating FMD dispersal. Green et al. (2006) used another individual farm-based model to simulate the initial stage of disease spread via movement of individual hosts.

Dispersal models of the FMD often depend on complex non-linear equations, which require a number of parameters to perform realistic simulations of epidemic cases. Still, those models have limited functionality to simulate a dispersal pattern of FMD. A small dose of the pathogen could initiate the epidemic, implying that the regional scale epidemic starts from within-farm disease spreading processes. (Sanson et al., 1999) However, current models tended to simulate such an event implicitly using a rate of dispersal over time and space, mostly due to computational efficiency and/or model complexity issues (Reeves, 2013). Although the Cambridge-Edinburgh model (Keeling et al., 2001) takes into account the heterogeneity of farms, it is limited in simulating the infection between animals within a farm.

An approach based on the individual-based model (IBM) would be useful to simulate the spread of disease between farms as well as between hosts within a farm. In an IBM, individuals that behave independently are defined as simulation units, which make it easier to focus on changes of their characteristics and properties through time (Grimm & Railsback, 2012). Thus, stochastic phenomenon under heterogenic conditions could be simulated effectively using an

IBM (Hare and Deadman, 2004). For example, IBMs have been broadly used in various fields in ecology to examine population dynamics, interspecies competition, and other individual-level behaviors (Huston and DeAngelis, 1988; Hogeweg and Hesper, 1990; DeAngelis and Mooij, 2005).

IBMs have been developed to investigate the epidemics of different diseases. For example, AusSpread developed by the Australian Government Department of Agriculture and Water Resources has been used to predict the spread of disease at a regional scale to develop contingency planning of epidemics (Garner and Beckett, 2005). Kiss et al. (2006) developed the IBM that focuses on livestock movement during the United Kingdom crisis. Australian Animal Disease Spread (AADIS), a hybrid model developed by linking equation-based model and IBM, was also developed to take into account heterogeneity of premises in simulating the disease spread in national scale (Bradhurst et al., 2015).

IBMs have been developed to simulate the epidemic processes at the intra-farm level although they would require heavy computational resources (Bobashev et al., 2007). Because the IBMs often depend on a stochastic computation processes, the addition of behaving entities would help more realistic simulation of the FMD dispersal. Still, the IBMs often consider herds or farms as the basic unit of entities rather than hosts under the assumption that intra-farm disease spread processes would be simple or negligible. For example, the Davis Animal Disease Simulation (DADS) model (Bates et al., 2003) has been used to simulate the intra-farm disease spread based on Markov chain equations. Such method could allow the model to perform time-efficient simulations by counting the daily number of state-changing hosts. Still, no individual host was used in the simulation of the FMD dispersal process.

Little effort has been made to integrate both intra- and inter-farm disease spread models based on individual hosts at a regional scale (Kostova-Vassilevska et al., 2004). An alternative

approach based on the IBM could be used to develop the model that simulates a temporal pattern of FMD spread between farms as well as between hosts within a farm. The objective of this study was to develop the FMD-IBM, an IBM taking into account both intra- and inter-farm dispersal to simulate a temporal pattern of FMD spread in a region. At first, we calibrated the model by fitting the parameters using the DADS model and OIE observation data. The model was used to simulate the two FMD epidemic cases occurred in Korea. Such a model could be used to assess the effectiveness of disease management approaches by comparing parameters associated with inter-farm disease dispersal. Once the local records of disease outbreaks are accumulated, the FMD model would be useful for planning of control measurements (e.g. preemptive culling of farms), which would minimize the damages of disease spread within a region.

Table 2. Characteristics of key FMD spread simulation models

Type of modeling approach	Typical models	Model characteristics
Empirical model	○ Imperial model (Ferguson et al., 2001) ○ InterSpread (Morris et al., 2001)	- Perform simple and time-efficient simulation of regional-scale epidemic - Difficult to examine the impact of inter-herd dynamics on the epidemic cases
Individual farm-based model	○ Cambridge-Edinburgh model (Keeling et al., 2001) ○ AusSpread (Garner and Beckett, 2005) ○ Kiss et al., 2006. ○ Green et al., 2006.	- Model can readily take in account the heterogeneity of farms (e.g. farm sizes) - Difficult to examine the impact of inter-herd dynamics on the epidemic cases
Individual animal-based model	○ DADS (Bates et al., 2003)	- Model reflects the impact of intra-herd dynamics in simulating regional-scale disease spread - The individual animals do not behave independently but rather counted daily by host status similar to equation-based approach
Hybrid model	○ AADIS (Bradhurst et al., 2015)	- Simulates intra-herd dynamics based on equations and inter-herd dynamics based on IBM approach - Difficult to examine the impact of inter-herd dynamics on the epidemic cases

MATERIALS AND METHODS

1. FMD Outbreaks in 2010 and 2016 in Korea

Two epidemic events, which represent the most and the least severe epidemic events of FMD in Korea, were simulated using an FMD dispersal model based on individual based modeling approach (Table 3). Korea has suffered from major outbreaks of FMD since 2010. A considerable number of cattle and pigs have been slaughtered by the FMD outbreaks occurred in April 2010, November 2010, December 2014, and January 2016. In particular, the FMD outbreak occurred in November 2010 has been considered to be the most disastrous outbreak among all the recent events. The disease outbreak started from a farm in Andong city, Gyeongsangbuk-do (Fig. 1a), where several days have been passed before the disease was confirmed by the authorities (Park et al., 2013). During the first 10 days of the epidemic, 30 livestock farms in Andong city were affected by the disease. Those farms located within 14km from the index farm where the FMD was reported first. The epidemic lasted for about 5 months, resulting in culling 150,000 cattle and 3.32 million pigs during the outbreak. The FMD epidemic occurred in 2010 was used to represent the most severe disease outbreak in Korea.

The FMD outbreak occurred in 2016 would represent the least severe epidemic events of FMD in Korea. During the epidemic in January 2016, livestock farms in Chungcheongnam-do were affected by the disease (Fig. 1c). During the first 30 days of the epidemic, 15 livestock farms in Chungcheongnam-do, which located within 47km from the index farm, were affected by the disease. The epidemic lasted for about 3 and a half months, which resulted in relatively slight damages to the livestock industry in comparison to the 2010 epidemic. The clinical infection of

FMD was reported promptly by a local farmer, which allowed early implementation of disease control measures including the preemptive destruction of infected premises. Data from two epidemic cases in Korea were used to calibrate the FMD-IBM (Fig. 2).

Table 3. Comparison of the 2010 and 2016 FMD epidemic events occurred in Korea

	The 2010 epidemic	The 2016 epidemic
Start of event	2010-11-26	2016-01-11
Date of disease confirmation	2010-11-29	2016-01-12
Resolved date	2011-04-22	2016-04-23
Duration of the epidemic	147 days	102 days
FMD virus strain	serotype O	serotype O
Location of the disease	The whole country (started from Andong city in Gyeongsangbuk-do)	Chungcheongnam-do and Jeollabuk-do (mainly occurred in Chungcheongnam-do)
Number of outbreaks (affected farms)	155	21
Control measurements executed during the epidemic	● Movement restrictions ● Disinfection ● Stamping-out ● Epidemiological investigation	● Mandatory vaccination to all livestock farms ● Preemptive culling ● Standstill ● Epidemiological investigation

Note : Data were obtained from the WAHIS database

(http://www.oie.int/wahis_2/public/wahid.php/Countryinformation/countryhome)

(accessed on 2017. 6. 17).

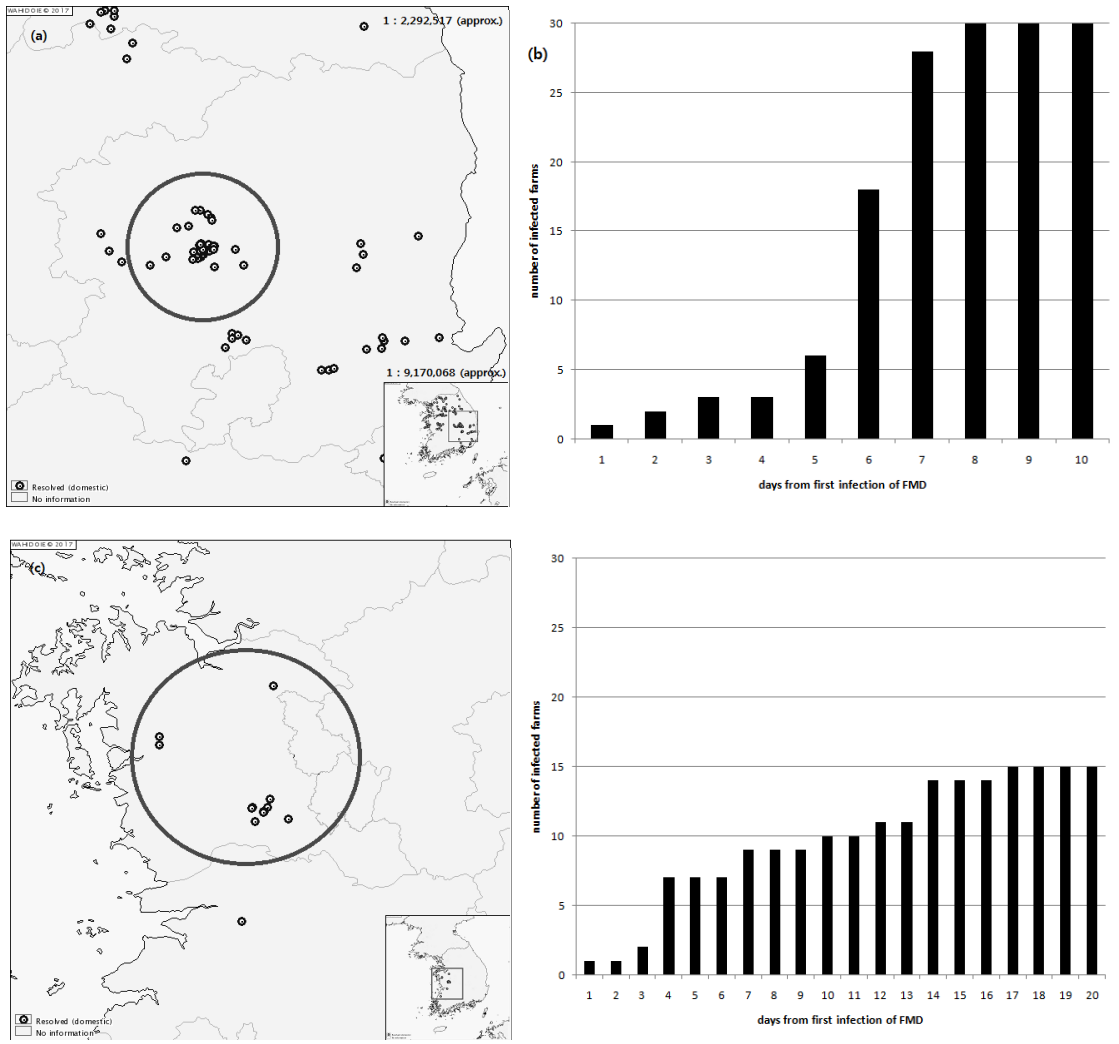


Fig. 1. Spatial (a and c) and temporal (b and d) distribution of infected farms during the 2010 and 2016 epidemics in Korea. Infected premises were indicated the circular boxes for the outbreak in Andong city (a) and in Chungcheongnam-do (c), respectively. FMD outbreak areas were indicated in the inset. Each bar indicates the cumulative number of infected farms in a region over elapsed time from the first report date during 2010 (b) and 2016 (d) epidemic events. Data were obtained from the WAHIS database (http://www.oie.int/wahis_2/public/wahid.php/Countryinformation/countryhome) (accessed on 2017. 6. 17).

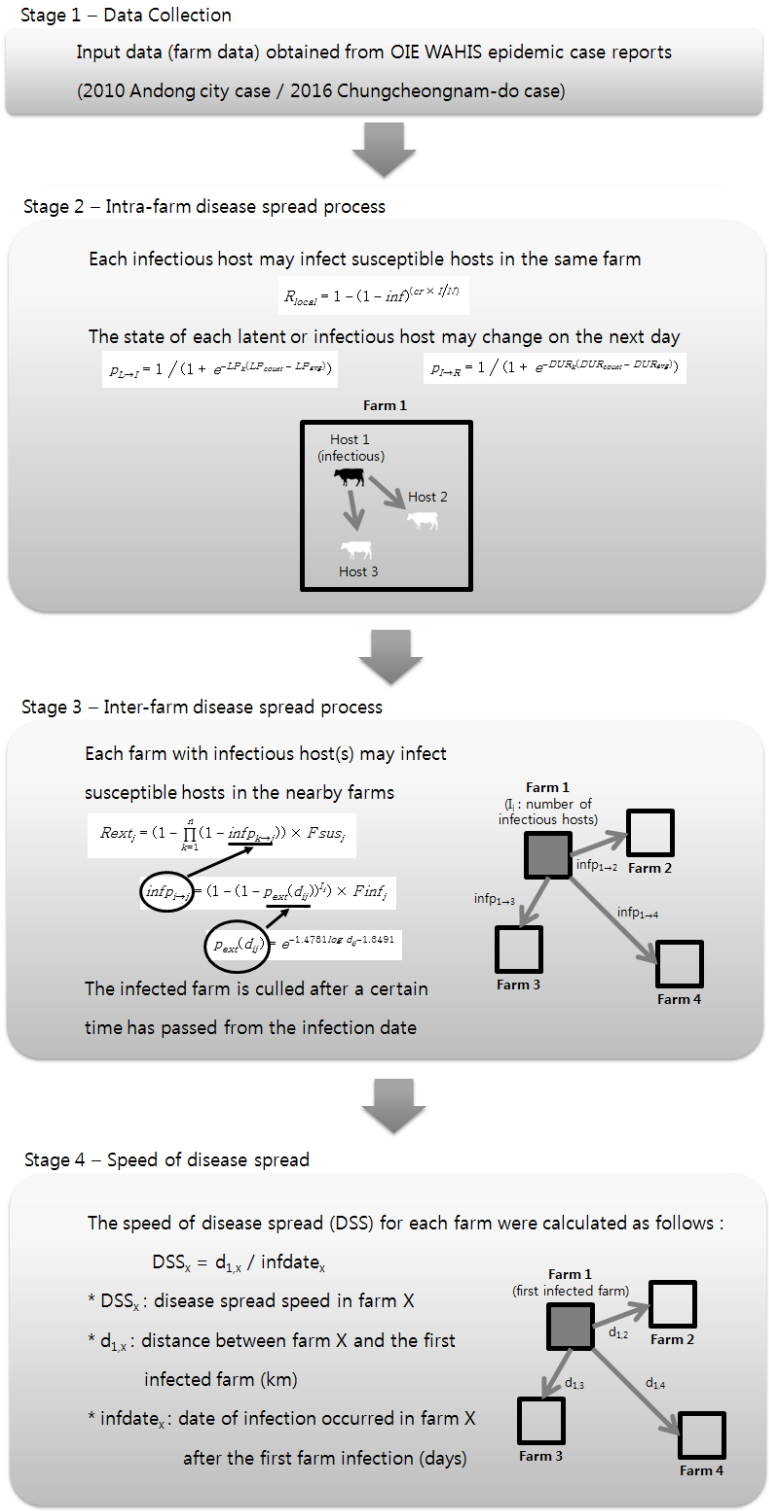


Fig. 2. Workflow of the model.

2. Design of the FMD-IBM

IBMs have been developed to simulate population dynamics of a given species (DeAngelis and Mooij, 2005). In particular, an IBM simulates a behavior of each individual that changes over time under given conditions. It is often assumed that individuals are homogenous and act independently under the given environment, e.g., temperature or prey density (Charles, 2008; Zurell, 2015). Interactions between individuals, which could alter the behavior of each individual, could also be simulated using the IBM. An IBM could simulate emergent complex behaviors at a macro scale using detailed local information, simple rules, and their interplays (Hogeweg and Hesper, 1990).

Reeves et al. (2013) suggested that the utility of a regional scale model would be improved by incorporating within-farm disease dynamics into the model for the FMD dispersal. The FMD-IBM model treats hosts and farms as individual entities to take into account disease dispersal process within a farm as well as between farms. For example, the chance such that a host in a managing unit, e.g., a farm, would be infected by an FMD virus differs by livestock allocated to the farm. Two types of hosts, i.e. cattle and pig are used to simulate the dispersal of FMD. Independent modules are also used to simulate the state and behavior of hosts and farms, which allow simulation of inter-farm disease dispersal.

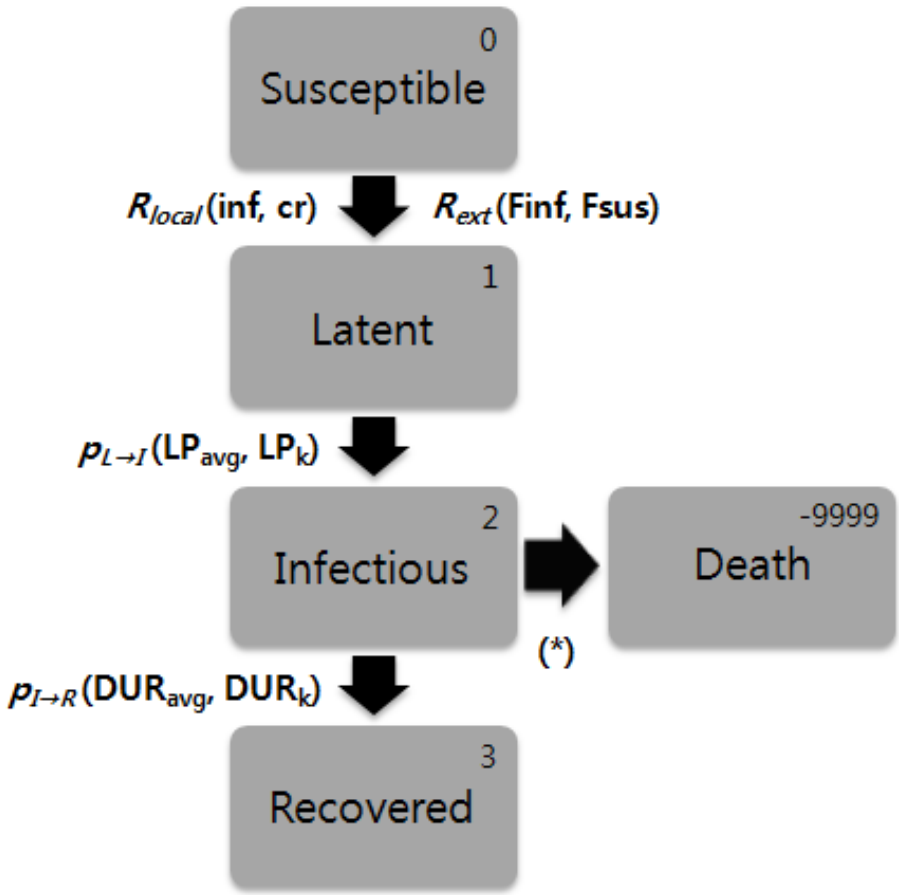


Fig. 3. Modified SIR host cycle and probabilities of state change events. The state of a host is represented in the FMD-IBM by the indicator value shown at the right top of each corresponding stages. The variables next to the arrows indicate the probability of the corresponding state change to occur on the next day, which was applied to each host associated with the corresponding population pool. Note that the event of infection (i.e. from Susceptible to Latent) could happen either by intra- or inter-farm disease spread processes. In the brackets are the parameters used to calculate the probability for each state transition. Parameter values differ by the type of hosts.

(*) : Only occurs when farm culling option was enabled during the simulation.

The FMD-IBM was designed to simulate the FMD dispersal based on the Susceptible-Infected-Recovered (SIR) host cycle (Fig. 3). Because the intra-farm dispersal occurs in a relatively short period of time, it was assumed that there is no inflow or outflow of hosts in all farms during the epidemic. The model does not consider the events of population change that may occur in farms (e.g. birth of new hosts or natural death of hosts) except for the effect of stamping-out infected premises. The probability such that the state of an individual host would change from one to another, e.g., susceptible to latent, was based on the Reed-Frost model, which has been widely used in conceptualizing the process of epidemics (Vynnycky, 2010). The basic form of Reed-Frost equation can be written as follows (Abbey, 1952) :

$$P_I = 1 - (1 - p)^I \qquad \text{(Eq. 1)}$$

where the probability of infection upon a contact (p) and the number of contacts with infectious premises (I) determine the probability of infection after I times of contacts were made with the infectious premises (P_I).

Susceptible individuals could become infected when they are contacted with the FMD virus released from an infected host within a farm. An infection process within a farm was simulated for the farm that had at least one individual in latent or infectious states. Each healthy host in a farm was exposed to the risk of infection, which is determined by the number of infectious hosts in the farm. The intra-farm infection risk (R_{local}) was defined using the modified Reed-Frost equation as follows :

$$R_{local} = 1 - (1 - inf)^{(cr \times I/N)} \qquad \text{(Eq. 2)}$$

where inf indicates the possibility of infection in contact with a single infectious host throughout a day, I/N represents the portion of infectious hosts in the farm on a given day, and cr is the number of contacts between hosts within the farm. inf and cr are parameters which their values differ by the type of host. All susceptible hosts in the farm are granted a probability of R_{local} whether to be infected or not on the next day of simulation.

A farm would be exposed to the risk of inter-farm infection when the FMD virus would be dispersed from infectious hosts in a nearby farm. The risk of inter-farm infection at a given farm j from nearby farms ($Rext_j$) was calculated as follows :

$$Rext_j = \{1 - \prod_{k=1}^{n}(1 - infp_{k \to j})\} \times Fsus_j \qquad \text{(Eq. 3)}$$

where $Fsus_j$ represents the susceptibility of farm *j*, which is determined by the type of hosts in the farm, e.g., cattle or pig. $infp_{k \to j}$ indicates the chance that farm *j* would be infected by the host in farm *k*, which is calculated as follows :

$$infp_{i \to j} = (1 - (1 - p_{ext}(d_{ij}))^{I_i}) \times Finf_j \qquad \text{(Eq. 4)}$$

d_{ij} is the distance between farms in kilometers. I_i represents the number of infectious hosts in farm *i*, which is determined by the intra-farm disease spread process of the given farm. $Finf_j$ indicates the infectability of farm *j*, which is determined by the type of host in the given farm. $p_{ext}(d_{ij})$ is the possibility function of farm-to-farm infection depending on the distance between farms, which is calculated as follows (Keeling et al., 2001) :

$$p_{ext}(d_{ij}) = e^{-1.4781 log\, d_{ij} - 1.3491} \quad \text{(Eq. 5)}$$

The value of $p_{ext}(d_{ij})$ would decrease as d_{ij} increases, indicating that the likelihood of farm-to-farm disease spread would be determined by distance. For all susceptible farm *j*, each host in the farms was given a probability of $Rext_j$ whether to be affected by inter-farm disease spread on the next day of simulation.

The FMD-IBM depends on a stochastic approach to simulate transition of states. Infection of each host occurred when the susceptible host had a higher probability of infection from a contact within a farm or a virus dispersed from a nearby farm than a random number (Fig. 4). Infected hosts would become infectious after the latency period had passed from the time of infection. The hosts would recover and become immune to the disease after infection. It was assumed that such hosts do not lose their immunity to FMD, therefore were removed from the pool of susceptible hosts. Those transitions of states were determined for each individual using a random number. Probability such that a given state of a host was turned into another state (i.e. $p_{L \to I}$ for the host to change from the latent state into infectious state and $p_{I \to R}$ for the host to change from the infectious state into recovered state) was as calculated as follows :

$$p_{L \to I} = 1 \,/\, (1 + e^{-LP_k (LP_{count} - LP_{avg})}) \quad \text{(Eq. 6),}$$

and

$$p_{I \to R} = 1 \,/\, (1 + e^{-DUR_k (DUR_{count} - DUR_{avg})}) \quad \text{(Eq. 7)}$$

where LP_{count} and DUR_{count} indicate the days elapsed since the date when the host became latent or infectious, respectively. LP_{avg} and DUR_{avg} are the parameters representing the

average latency period and infectious period of the host, respectively. LP_k and DUR_k are parameters that determine the variability of latency period and infectious period of hosts, respectively. The parameter values were different by the host type. Those values of probability were compared with a random number between 0 and 1. Once the probability of a given state to change, e.g., $p_{L\to I}$ or $p_{I\to R}$, was greater than those random number, it was assumed that the transition of a corresponding state occurred.

Culling would be performed for farms with infected hosts. In the FMD-IBM, hosts in infected farms would be culled in a given period after its infection when the culling was chosen. After the culling of an infected farm, all hosts of the farm were removed from the simulation, which prevented the farm from spreading the disease.

```
for each host Y in farm X
{
    if (Y is susceptible)
    {
        if (s1 < Rlocal OR s2 < RextX)
        {
            Y is now infected
            Y is changed into latent state on the next day
        }
    }
}

s1, s2 : Random value (0 ≤ s1, s2 < 1)
```

Fig. 4. Pseudo code for infection of hosts in the FMD-IBM. The probability of state change was applied independently to each host of the corresponding population pool, which allowed the individual units to behave stochastically. If the risk of intra- or inter-farm disease spread were larger than the random number given on a specific day, the host would become infected and change itself into the latent state.

3. Implementation of the FMD-IBM

An object-oriented programming (OOP) approach was used to implement the FMD-IBM. The model was implemented in C# to maximize the benefit of OOP, which could easily handle entities with different hierarchies as multi-level classes. Behavior and state of individuals in the FMD-IBM was defined using classes (Figs. 5-6). Instances of the class, or objects, were used to represent individual entities in the simulation of disease dispersal. All objects had the properties inherited from their parent classes, which consist of common characteristics to describe dispersal events and host properties (Table 4). Those objects allowed simulation of daily disease dispersal processes based on behavior and state of independent individuals. For example, cattle and pigs in a farm were instances of the host class in the FMD-IBM. Although the behavior of each host would be defined using the same function, each host could have different behaviors and states depending on interactions between hosts.

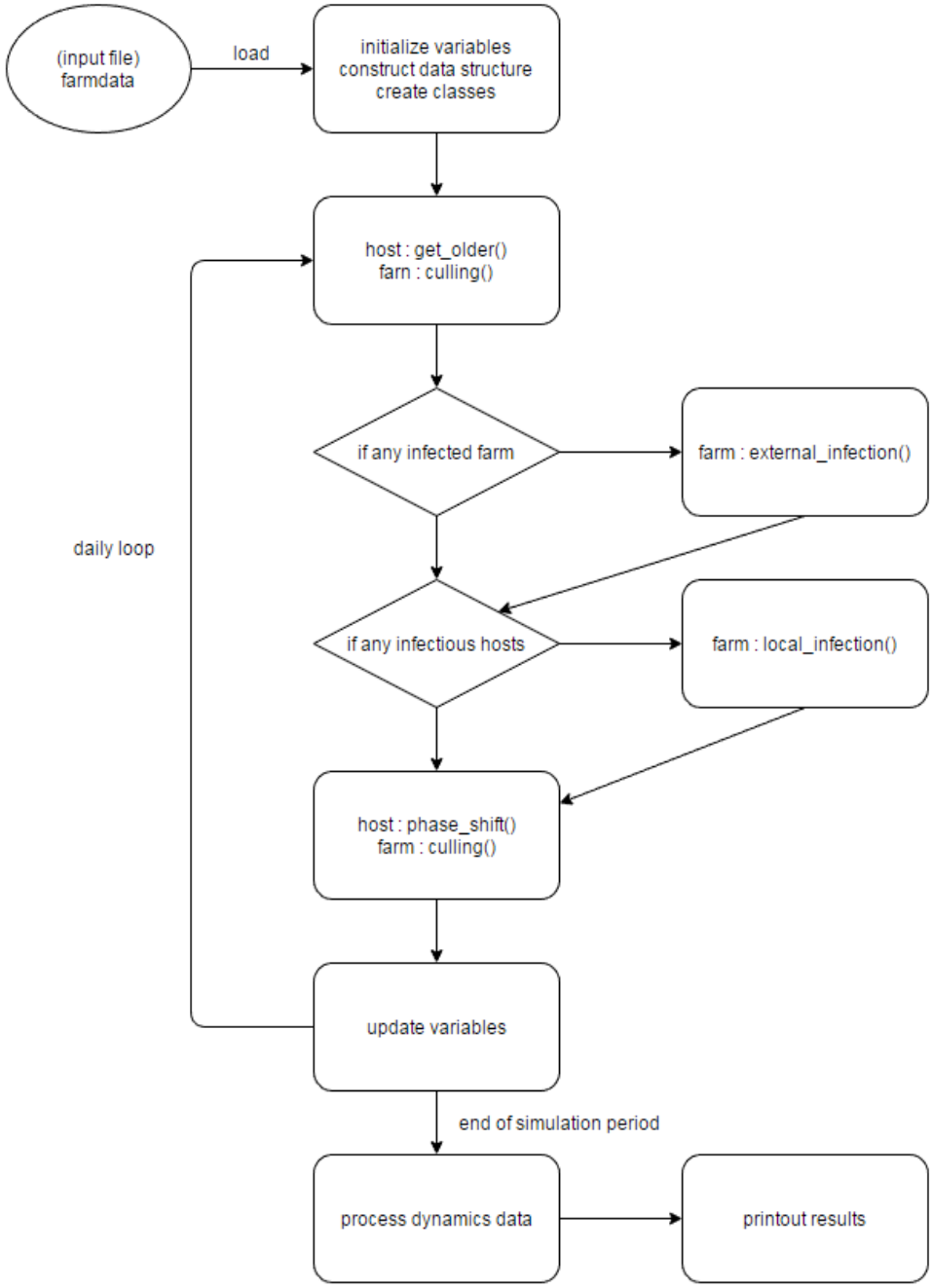


Fig. 5. Flowchart of the main function implemented in the FMD-IBM. Each object is affected by daily execution of the corresponding class subroutines. (e.g. host : get_older() increases the age value of all the objects in the host class.)

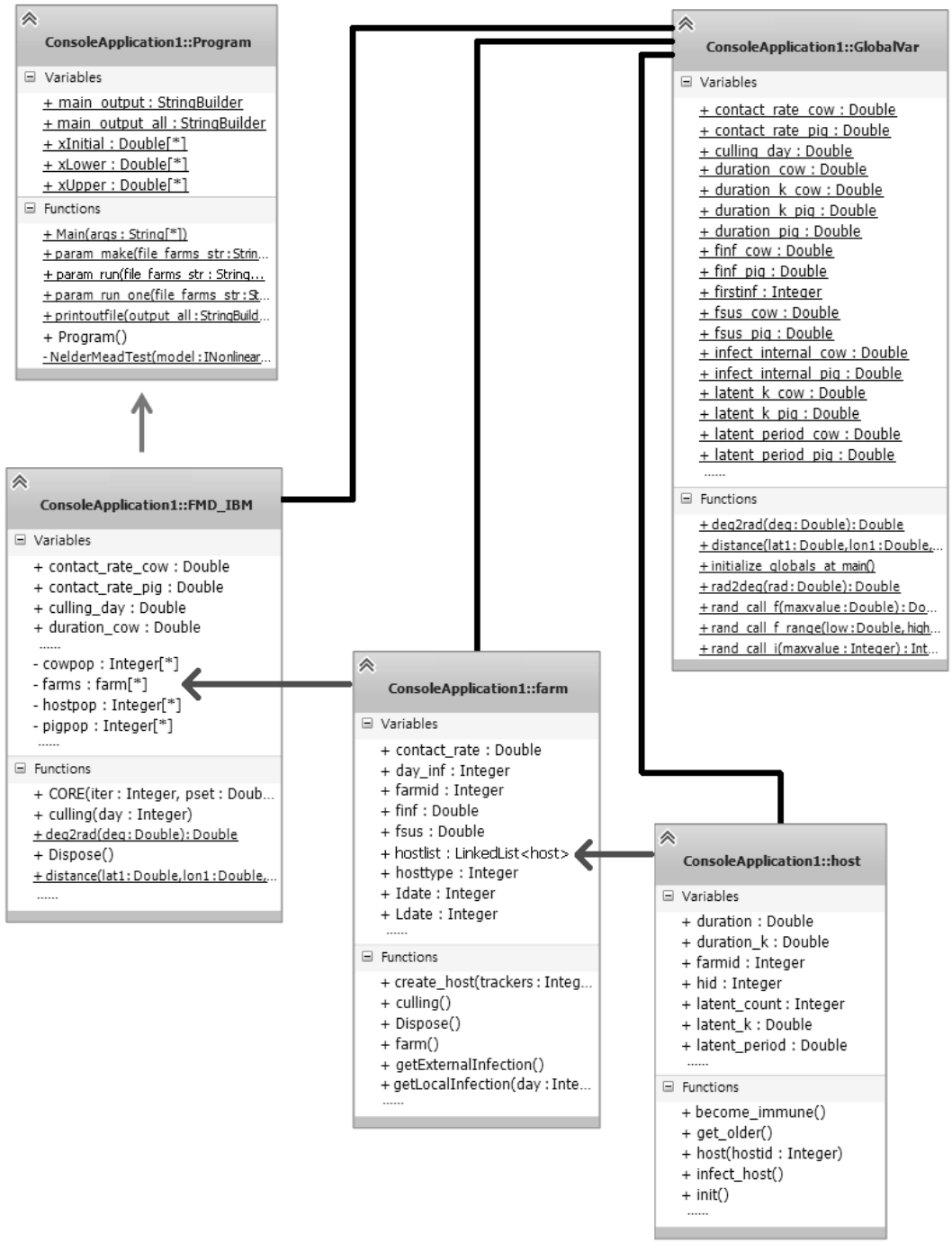


Fig. 6. Class diagrams of the FMD-IBM. (See appendix B for the full list of class properties implemented in the code.)

Table 4. List of class properties

Property type	Host class	Farm class
Constant properties	● ID of allocated farm ● Species type (1 : cattle, 2 : pig)	● Farm ID ● Location (longitude, latitude) ● Initial number of hosts
State variables	● Host status (0, 1, 2, 3, -9999) ● Host age (in days) ● Days passed after each phase shift	● Linked list of hosts - Number of hosts in each status (0, 1, 2, 3, -9999) ● Infected date of farm

The FMD-IBM determined the temporal state of disease outbreak in an area (Fig. 7). For example, date of infection occurrence on each farm and a daily number of infected farms were estimated using the minimum set of input data including the geographic coordinate of each farm and the number of hosts on the farm. It was assumed that disease dispersal started from one of the host infected in the first farm in the input data. The daily progress of disease outbreak was calculated to determine the date of initial infection and the number of hosts in given states in each farm.

FMD-IBM requires a simple form of farm data as an input file to perform inter-farm disease spread (Fig. 8). Minimum requirements for farm data are the identification number of farms, geographical location (i.e., longitude, latitude, and altitude), species type (i.e., 1 for cattle and 2 for swine) and the initial number of hosts of each farm in a region.

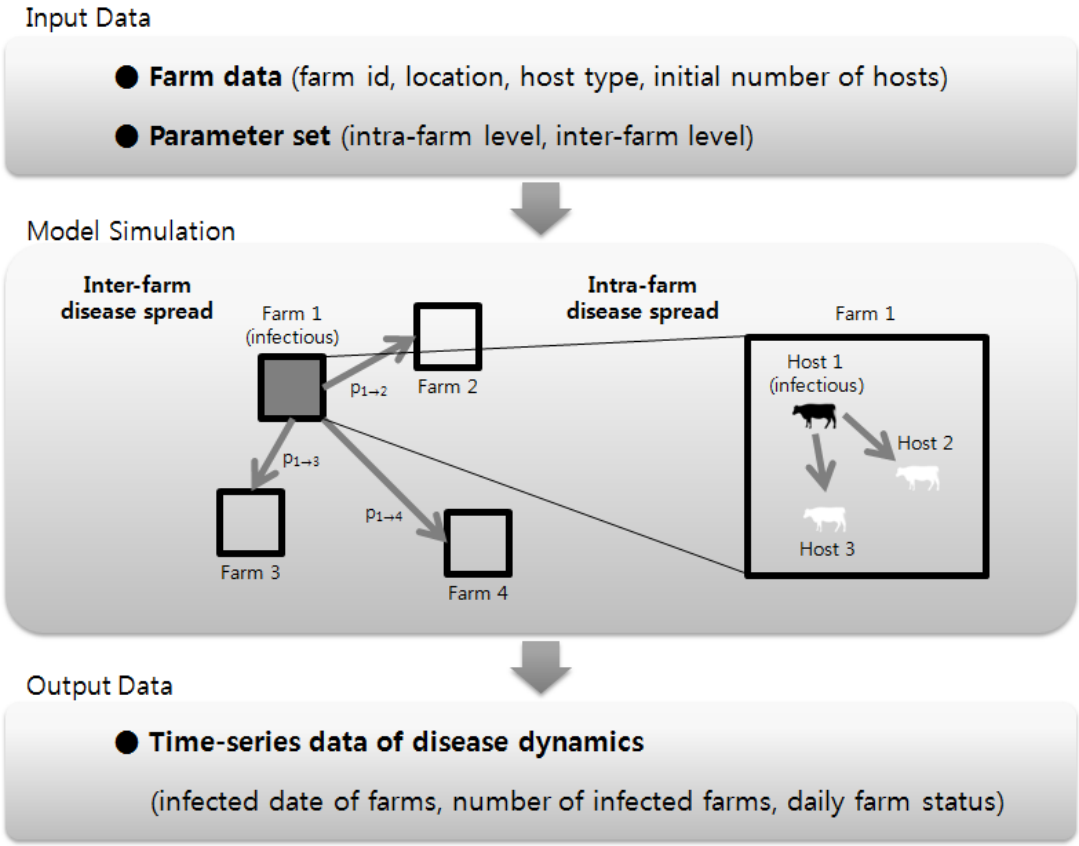


Fig. 7. An overview of disease spread simulation processes using the FMD-IBM

farmno	locx	locy	alt	hosttype	hostnum
1	36.616428	128.766547	208.98	2	5500
2	36.616428	128.766547	208.98	2	3500
3	36.616946	128.663858	119.59	1	5
4	36.598272	128.71336	195.77	1	210
5	36.648572	128.787871	235.67	1	3
6	36.655528	128.810485	208.61	1	150
7	36.646543	128.705766	142.15	1	29
8	36.651178	128.732077	188.94	1	80
9	36.607496	128.741584	190.09	2	4000
10	36.585434	128.683729	132.21	1	117
11	36.635927	128.69969	129.93	1	69
12	36.642104	128.72379	165.87	1	100
13	36.616838	128.690951	146.79	1	6
14	36.624273	128.739769	208.2	1	106
15	36.648747	128.734067	202.71	1	36

Fig. 8. Example of an input farm data for the FMD-IBM

4. Calibration of the Intra-farm Disease Spread Process of the FMD-IBM

The parameters of the FMD-IBM shown in Table 5 were determined to characterize the intra-farm disease dispersal for each type of hosts. Because little observation data were available to the public for model calibration, parameters for intra-farm dispersal were calibrated against another model that simulates disease spread among animals in a farm. The DADS model was used to generate the calibration data sets for intra-farm dispersal.

The DADS model consists of modules to simulate intra-herd and inter-herd dispersal of the FMD. The intra-herd transmission of the disease is simulated based on the Markov chain Monte-Carlo method. The number of infection occurrence on a specific day (C_{t+1}) is calculated using a modified Reed-Frost equation as follows (Carpenter et al., 2004) :

$$C_{t+1} = S_t(1 - (1 - p)^{CI_t}) \qquad \text{(Eq. 8)}$$

where S_t, p, and CI_t represent the number of susceptible hosts, the probability of an adequate contact, and the cumulative number of infectious animals, respectively. Although both the DADS model and the FMD-IBM model implements the concept of Reed-Frost equation, the DADS model calculates the daily number of newly infected hosts by multiplying the overall probability of infection by the number of susceptible hosts on a given date (S_t) using the Markov chain approach. On the other hand, the FMD-IBM evaluates the probability of infection by individual hosts to determine whether or not a given host was infected by chance (Eq. 2). The differences in model details between the DADS model and FMD-IBM were listed in Table 6.

The calibration data were prepared from 50 sets of dispersal simulations within a farm using the intra dispersal module of the DADS model. The initial number of hosts in a farm was set to be 700 and 7000 for cattle and swine farms, respectively, which represents a large-sized farm in epidemic cases at a regional scale. The probability of an adequate contact between hosts in a farm for the DADS model, or the value of p, were set to be 0.013 and 0.001 for cattle and pig, respectively (Table 7). It was also assumed that host recovered from the disease would retain their immunity to FMD during the simulation, and no culling or vaccination was performed in a farm of interest. The proportion of susceptible, infected, and recovered hosts tended to be similar over time irrespective of the total host numbers. Those suggested that calibrated parameter sets would be useful for farms with different sizes (Appendix A).

Table 5. List of parameters affecting intra-farm disease spread

Variable name	Description	Used in the following equation
inf	Probability of disease spread from infectious host to susceptible host upon contact (%)	Eq. 2
cr	Number of daily contacts made by hosts on a single farm	Eq. 2
LP_{avg}	Average latency period of host (days)	Eq. 6
DUR_{avg}	Average infectious period of host (days)	Eq. 7
LP_k	Variability of latency period among infected hosts	Eq. 6
DUR_k	Variability of infectious period among infected hosts	Eq. 7

Table 6. Comparison between the DADS model and FMD-IBM

	DADS model (Bates et al., 2003)	FMD-IBM
Type of modeling approach	Stochastic (IBM) Markov chain Monte-Carlo method	Stochastic (IBM) Reed-Frost equation
Scale of simulation	Intra-farm & Inter-farm (Regional scale)	Intra-farm & Inter-farm (Regional scale)
Mechanism used to simulate intra-herd dynamics	Daily calculates the number of state-changing hosts	All host units behave independently based on daily probabilities of state change
Features and details	- Distinguish 13 types of herds (cattle, pig)	- Distinguish two types of host species (cattle, pig)
	- Distinguish hosts with FMD symptoms into avirulent or virulent state (considers the reporting time of the disease)	- Consider all hosts with symptoms as virulent hosts (does not consider reporting time of the disease)
	- Recovered hosts may lose their immunity to FMD over time	- Recovered hosts do not revert back to susceptible state
	- Distinguish direct and indirect pathways of disease spread for inter-farm dynamics	- Probability of farm-to-farm disease spread is mainly determined by the distance between farms
	- Supports option for farm culling and vaccination	- Supports option for farm culling
	- Implemented in R	- Implemented in C#

Table 7. The value of *p* used in the intra-herd dynamics module of the DADS model

	Threshold of plume concentration to infect a host ($TCID_{50}$ / m^3)	Mean release of virus from an infected host (log_{10} $TCID_{50}$ / day)	Probability of making an adequate contact (*p*)
Cattle	0.07[a]	4.3[b]	0.013[d]
Pig	16[a]	5.6[c]	0.001[e]

[a] Sørensen et al., 2001.

[b] Alexandersen et al., 2002.

[c] Gloster et al., 2007.

[d] Carpenter et al., 2004.

[e] The value of *p* for a pig farm was calculated by multiplying that for a cattle farm by the adjustment factor (0.08), which was determined by dividing the ratio of threshold concentration (0.07 / 16) by the ratio of the mean virus release rate ($10^{4.3}$ / $10^{5.6}$).

Parameter values were determined to minimize errors in predicting the date of occurrence for specific events within a farm in FMD dispersal. The dates were determined for three events such that:

a) more than 1% of the total population became infected,

b) more than 1% of the total population became recovered, and

c) the number of susceptible hosts became less than 1% of the total population.

Because of the stochasticity of host state changing mechanism in FMD-IBM, parameter fitting was performed using the brute-force searching method. A set of parameter values listed in Table 8 was used to simulate dispersal of FMD within a farm. A range of each parameter was obtained from the epidemic reports from the farm investigators and local knowledge provided by Animal and Plant Quarantine Agency (QIA) in Korea. The simulation was performed for 500 times using each parameter sets and the averages of those results were compared with the calibration sets.

The parameter sets for cattle and pig farm were selected to have the least error compared to the calibration set. The least sum of absolute error between occurrence dates of disease outbreak events was calculated to determine the parameter set as follows:

$$\text{Error} = |a_{sim} - a_{obs}| + |b_{sim} - b_{obs}| + |c_{sim} - c_{obs}| \quad \text{(Eq. 9)}$$

where a_{sim}, b_{sim}, c_{sim} is the three event emergence dates from the FMD-IBM simulation results and a_{obs}, b_{obs}, c_{obs} is the corresponding dates from the calibration sets, respectively.

Table 8. The range and intervals of each parameter to generate parameter sets for calibration of intra-farm disease spread process

Variable name	Parameter value range (interval)
inf	0.01~0.25 (0.01)
cr	10.0~50.0 (0.1)
LP_{avg}	2.00~6.00 (0.01)
DUR_{avg}	6.00~10.00 (0.01)
LP_k	4.0~6.0 (0.1)
DUR_k	4.0~6.0 (0.1)

Note : All possible combination of parameter sets within each range of parameters (about 760,000,000 combinations) were generated and simulated using a parallel computing method.

5. Calibration of the Inter-farm Disease Spread Process of the FMD-IBM

The FMD-IBM model depended on parameters that represent the degree of infection in a given farm (Table 9). Those parameters were used to determine the overall likelihood of disease spread during the regional-scale epidemic case. Each host type was assigned a different parameter set for inter-farm disease spread process.

Parameters of inter-farm disease dispersal were calibrated using the disease occurrence reports in Korea. For example, epidemic reports for the 2010 and 2016 events, which were accessible from the WAHIS database provided by the World Organisation for Animal Health (OIE), were used to calibrate those parameters. Data of farms affected at the early stage of each epidemic were collected from WAHIS. As a result, 30 farms in Andong city and 15 farms in Chungcheongnam-do were obtained for the 2010 case and the 2016 case, respectively. Because the case reports of WAHID only provide the information of administrative districts where the farms belong rather than their exact location, we assumed that the minimum distance between farms in the same district was 0.01 km. The accumulated number of infected farms over time was calculated for the report from the OIE database and model outputs.

A set of parameters was obtained for each host type using the simulated infection pattern on a farm. The brute-force searching method was used to determine the parameter sets (Table 10). Because the FMD-IBM performs a stochastic simulation, simulations were repeated for 500 times to compare its averages with the OIE reports. The parameter set with the least sum of absolute error was selected to be used in the model during region scale simulations as follows:

$$\text{Error} = \frac{\sum_{k=1}^{n} |inf_{sim}(k) - inf_{obs}(k)|}{n} \quad \text{(Eq. 10)}$$

where *n* is the simulation period in days, which was set to 20 for 2010 Andong case and 30 for 2016 Chungcheongnam-do case. $inf_{sim}(k)$ and $inf_{obs}(k)$ indicate the simulated and reported number of infected farms on a given day *k*, respectively.

Table 9. List of parameters affecting inter-farm disease spread

Variable name	Description	Used in the following equation
Finf	Infectability of a farm (0~1)	Eq. 4
Fsus	Susceptibility of a farm (0~1)	Eq. 3

Table 10. The range and intervals of each parameter to generate parameter sets for calibration of inter-farm disease spread process

Variable name	Parameter value range (interval)	
Finf	0.001~0.999	(0.001)
Fsus	0.001~0.999	(0.001)

Note : All possible combination of parameter sets within each range of parameters (about 1,000,000 combinations) were generated and simulated using a parallel computing method.

RESULTS

1. Calibration of Parameters for Intra-farm Disease Spread

Parameters for cattle and pigs were relatively similar to each other except for ones associated with variability of infection processes (Table 11). For example, the average latency period (LP) for both types of hosts was determined to be 3.40 days on average. The number of daily contacts between hosts (CR) of the pig farm was slightly higher than that of the cattle farm. On the other hand, cattle had a longer average infectious stage period (DUR) than pigs.

The difference between the occurrence dates for a given event was considerably small between the FMD-IBM simulation results and from the calibration sets (Table 12). For example, it took about 7.4 days on average after the first host become in its latent state until the number of infected cattle became more than 1% of the initial population of a cattle farm. The same event occurred after 9 days of simulation in the calibration set data. The FMD-IBM model had the faster disease spread within a farm than the calibration set data (Figs. 9-11). The number of hosts in a given state was also different from the calibration sets and the model outputs.

Table 11. Calibrated parameter sets for intra-farm disease spread

	Cattle farm	Pig farm
inf	0.13	0.11
cr	23.5	27.0
LP_{avg}	3.40	3.40
DUR_{avg}	8.65	7.45
LP_k	5.9	5.0
DUR_k	4.4	5.6

Table 12. Comparison of occurrence dates for a given event between the simulation result of the FMD-IBM and the calibration set data

Event emergence dates	Calibration set	FMD-IBM simulations (average)	Percentage of simulation with 1 day or less error	Percentage of simulation with 2 days or less error
(a) in cattle farm	9	6 ~ 10 (7.4)	40%	90%
(b) in cattle farm	19	19 ~ 23 (20.4)	60%	95%
(c) in cattle farm	20	21 ~ 23 (22.3)	15%	55%
(a) in pig farm	14	12 ~ 15 (13.15)	85%	100%
(b) in pig farm	24	24 ~ 26 (24.95)	80%	100%
(c) in pig farm	25	27 ~ 29 (27.75)	0%	40%

Note : Event emergence dates are represented as the number of days passed after the first day of simulation when

(a) more than 1% of the total population became infected

(b) more than 1% of the total population became recovered

(c) the number of susceptible hosts became less than 1% of the total population

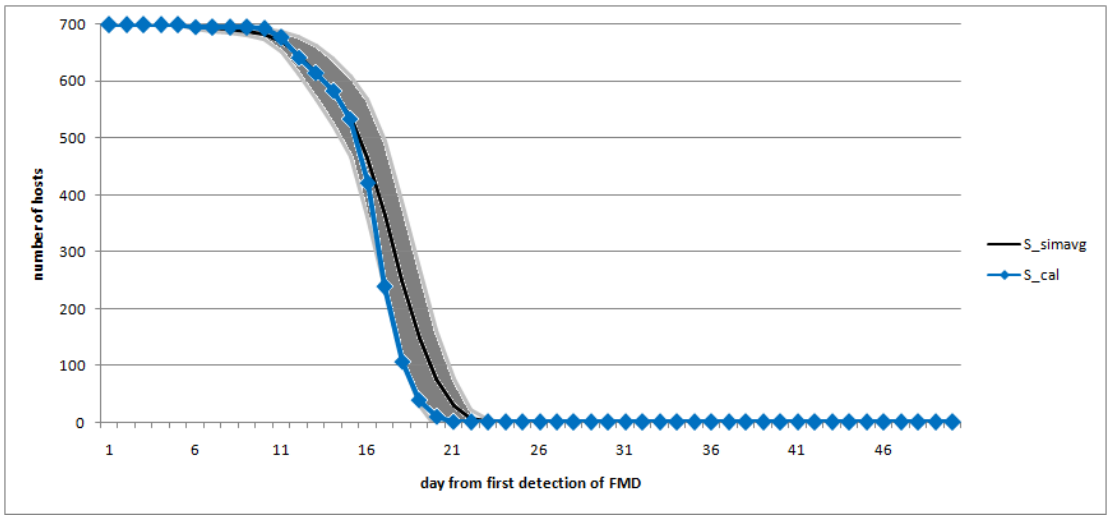


Fig. 9. Simulated number of susceptible hosts from the FMD-IBM in comparison to that in the calibration data set. A symbol on the solid line *(S_cal*) represents the number of susceptible hosts on a given day in the calibrated data set. The black line represents the average value of 500 simulations for 700 cattle in a farm using the FMD-IBM. The shaded area indicates the host numbers within 95% of stochastic simulation outcomes using the FMD-IBM.

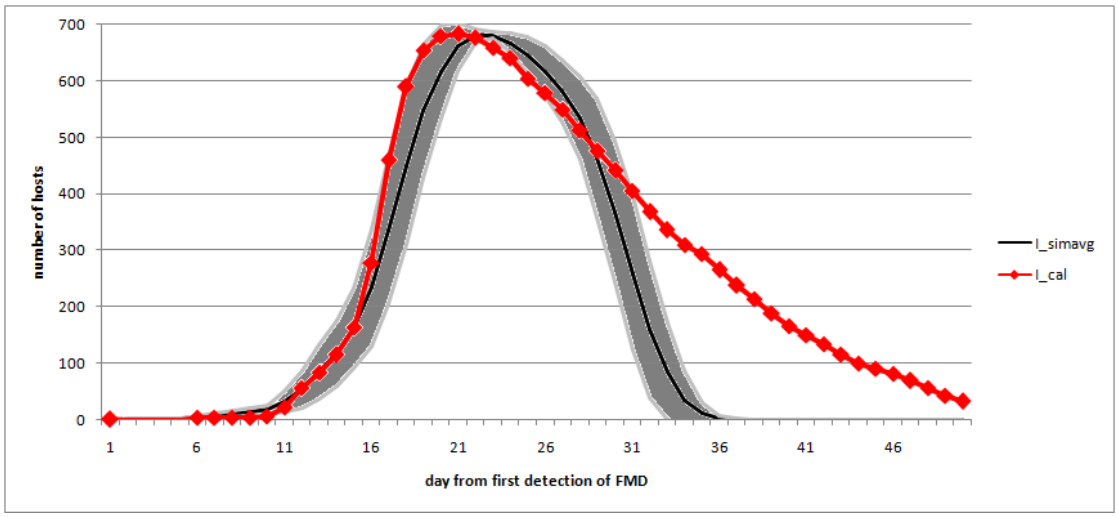


Fig. 10. Simulated number of infected hosts from the FMD-IBM in comparison to that in the calibration data set. A symbol on the solid line (*I_cal*) represents the number of infected hosts on a given day in the calibrated data set. The black line represents the average value of 500 simulations for 700 cattle in a farm using the FMD-IBM. The shaded area the host numbers within 95% of stochastic simulation outcomes using the FMD-IBM.

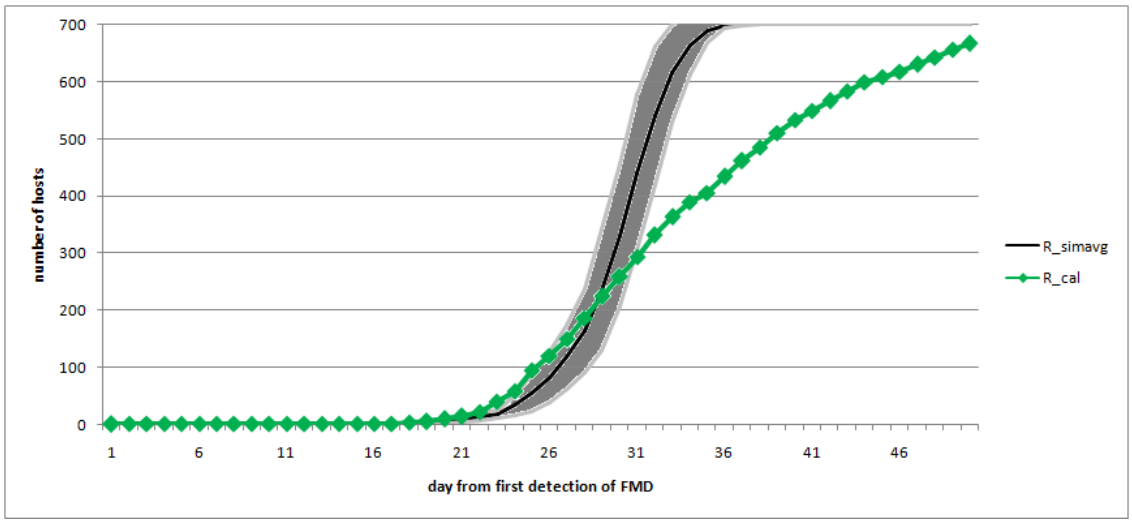


Fig. 11. Simulated number of recovered hosts from the FMD-IBM in comparison to that in the calibration data set. A symbol on the solid line (*R_cal*) represents the number of recovered hosts on a given day in the calibrated data set. The black line represents the average value of 500 simulations for 700 cattle in a farm using the FMD-IBM. The shaded area the host numbers within 95% of stochastic simulation outcomes using the FMD-IBM.

2. Calibration of Parameters for Inter-farm Disease Spread

The parameter values for the inter-farm dispersal were considerably different between the two regions (Table 13). The parameter values for the 2016 Chungcheongnam-do case were significantly lower than those for the 2010 Andong case. In both cases, pig farms had a greater value of susceptibility than cattle farms. Cattle farms also had higher infectability than pig farms in those cases.

An exponential growth of infected farm was simulated using the FMD-IBM for the 2010 Andong case (Fig. 12). The FMD-IBM model identified a specific time period from 4 to 8 days after the first infection, during which a large number of farms would be infected. During this period, 25 farms were infected by the FMD in 2010. On average, 23 farms were infected during the exponential growth phase of disease dispersal in the simulation of disease spread using the FMD-IBM model. The FMD-IBM model tended to predict no outbreak of FMD at farms considerably distant from other farms within 20 days of simulation.

Table 13. Calibrated parameter sets for inter-farm disease spread

	Host type	2010 Andong	2016 Chungcheongnam-do
Finf	cattle	0.56	0.064
	pig	0.16	0.007
Fsus	cattle	0.13	0.023
	pig	0.85	0.064

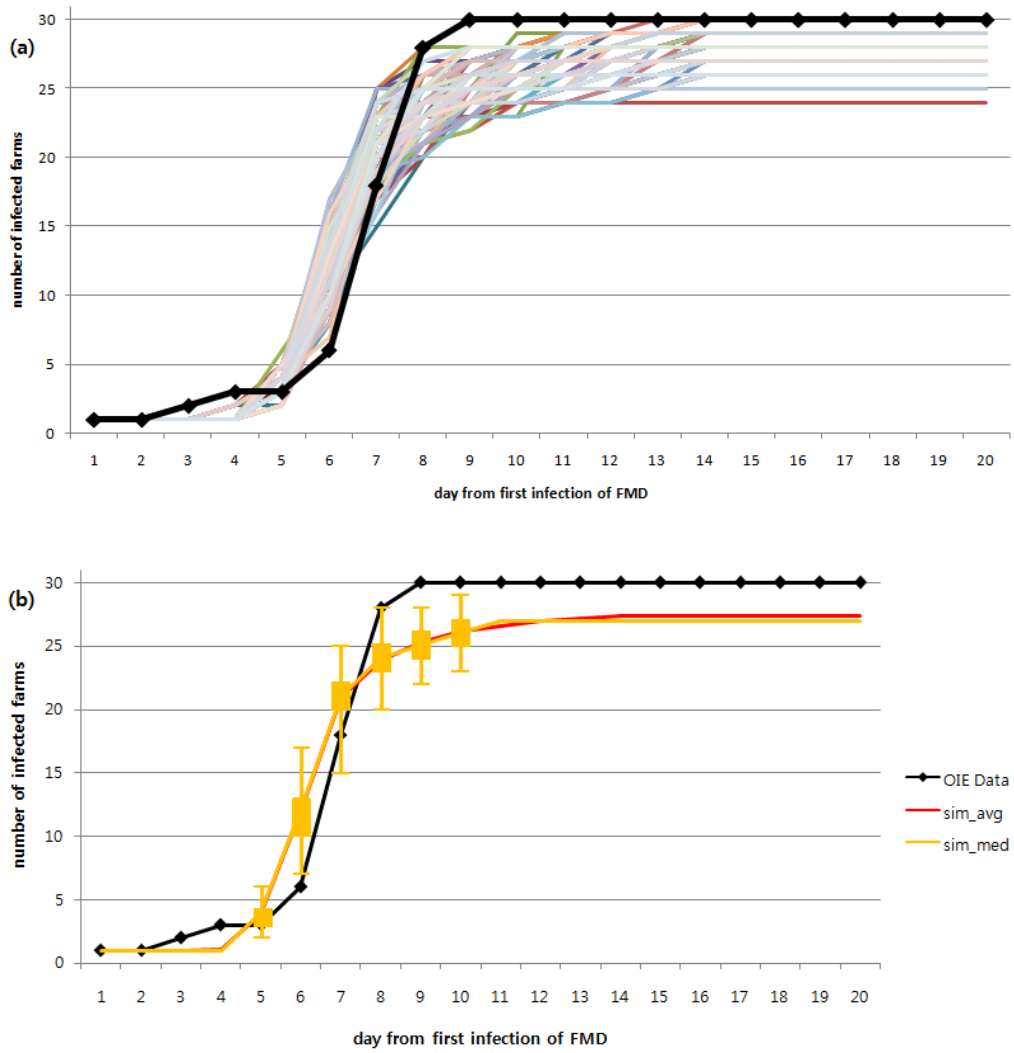


Fig. 12. Comparison between simulated results of the FMD-IBM and reported data of 2010 Andong FMD epidemic case. The black line indicates the reported number of infected farms over time. (a) shows the result of 500 repeated simulations under identical conditions, and (b) shows the average (red) and median (yellow) value of the simulated results. The box plots describe the quartiles of the simulated number of infected farms on a corresponding day.

The FMD-IBM predicted a gradual increase of infected farms during the Chungcheongnam-do outbreak in 2016 (Fig. 13). For example, the number of infected farms increased from 7 to 14 for 10 day periods, e.g., from 7 to 16 days after first infection date, in a simulation of disease dispersal using the FMD-IBM, which was similar to the observed data. However, a small number of farms were not identified as infected farms by the FMD-IBM model until the end of the simulation period.

The parameter sets associated with inter-farm disease dispersal were not effective to simulate the other case of the FMD epidemic (Fig. 14). For example, simulation results for Chungcheongnam-do case using the parameter set of Andong case (*sim_avg* and *sim_med*) indicated that all premises in the region were infected in 2~3 days. However, the timing of mass outbreak emergence had no drastic change because it was mainly determined by the intra-farm disease spread dynamics of the first infected farm.

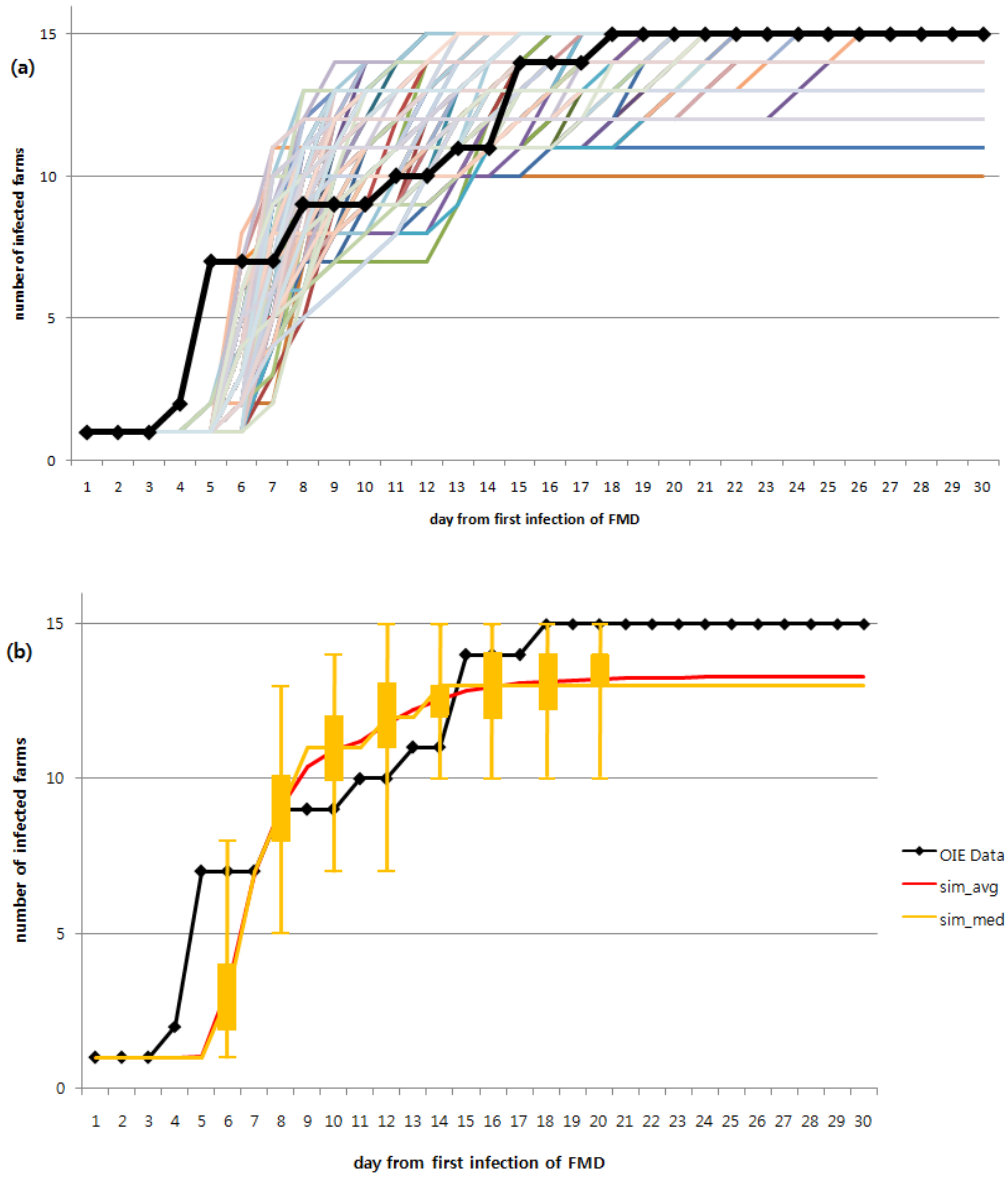


Fig. 13. Comparison between simulated results of the FMD-IBM and reported data of 2016 Chungcheongnam-do FMD epidemic case. The black line indicates the reported number of infected farms over time. (a) shows the result of 500 repeated simulations under identical conditions, and (b) shows the average (red) and median (yellow) value of the simulated results. The box plots describe the quartiles of the simulated number of infected farms on a corresponding day.

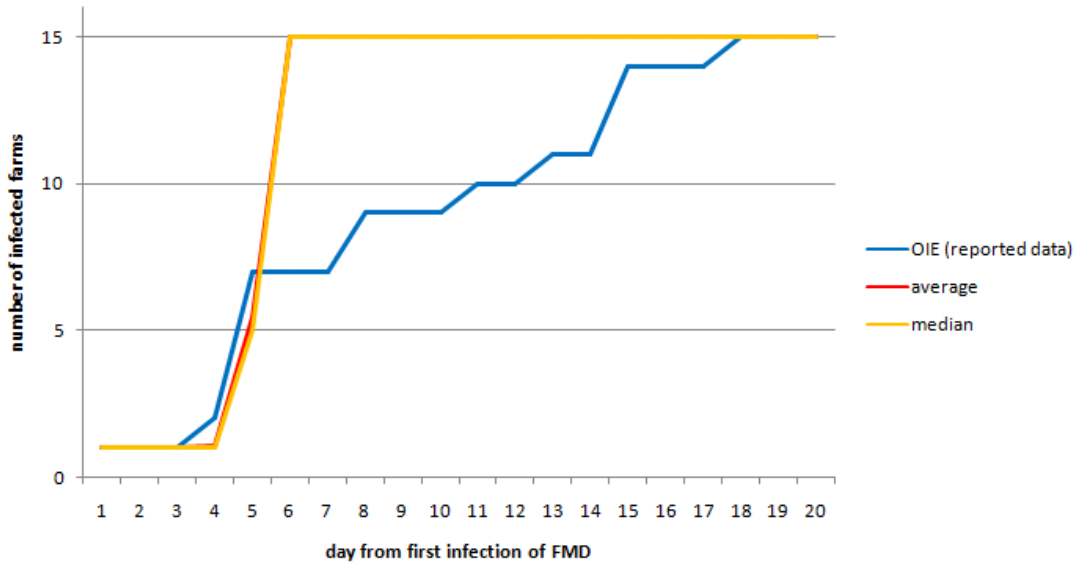


Fig. 14. Simulation result under conditions corresponding to the 2016 Chungcheongnam-do case using parameters calibrated for the 2010 Andong case. The average and median of the simulated number of infected farms were shown daily as red and yellow line, respectively, while the blue line represents the reported number of infected farms during the 2016 Chungcheongnam-do epidemic.

3. Speed of Disease Spread for Individual Farms

The speed of disease spread at a specific farm was calculated by dividing the distance between the farm and the origin of the disease (i.e. the location of the first infected farm) by the days for the farm to become infected. Although the speed of disease spread differs by the characteristics of farms as well as their locations, the FMD-IBM had a relatively high degree of agreement between the simulated and observed speed of disease dispersal for a given farm (Figs. 15-16). The speed of disease spread in most of the farms ranged from 0.3 to 1.3 km per day except for several sites distant from the farm where the FMD outbreak started.

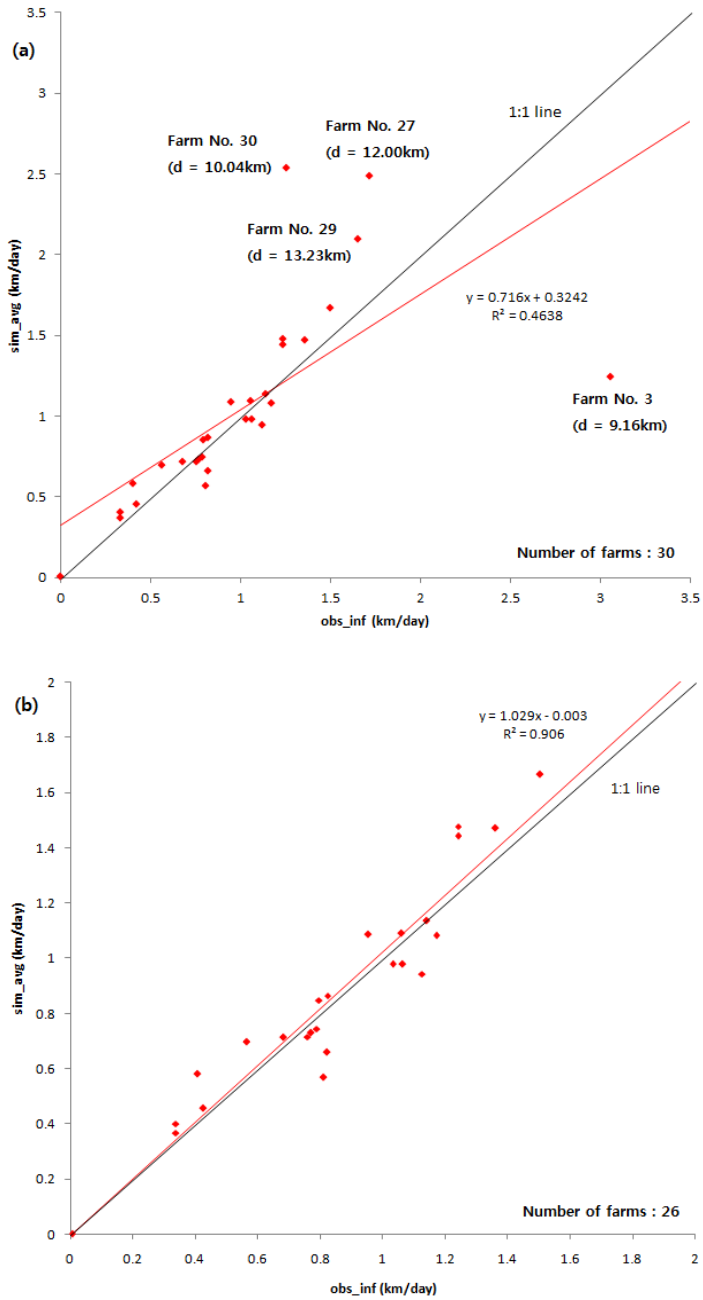


Fig. 15. Disease spreading speed of each farm from simulated and reported results of the 2010 Andong epidemic case (a) with all 30 farms in a region, and (b) without 4 farms distant from the disease origin. *obs_inf* and *sim_avg* represent the disease spread speed derived from reported data and simulation results of 500 iterations, respectively.

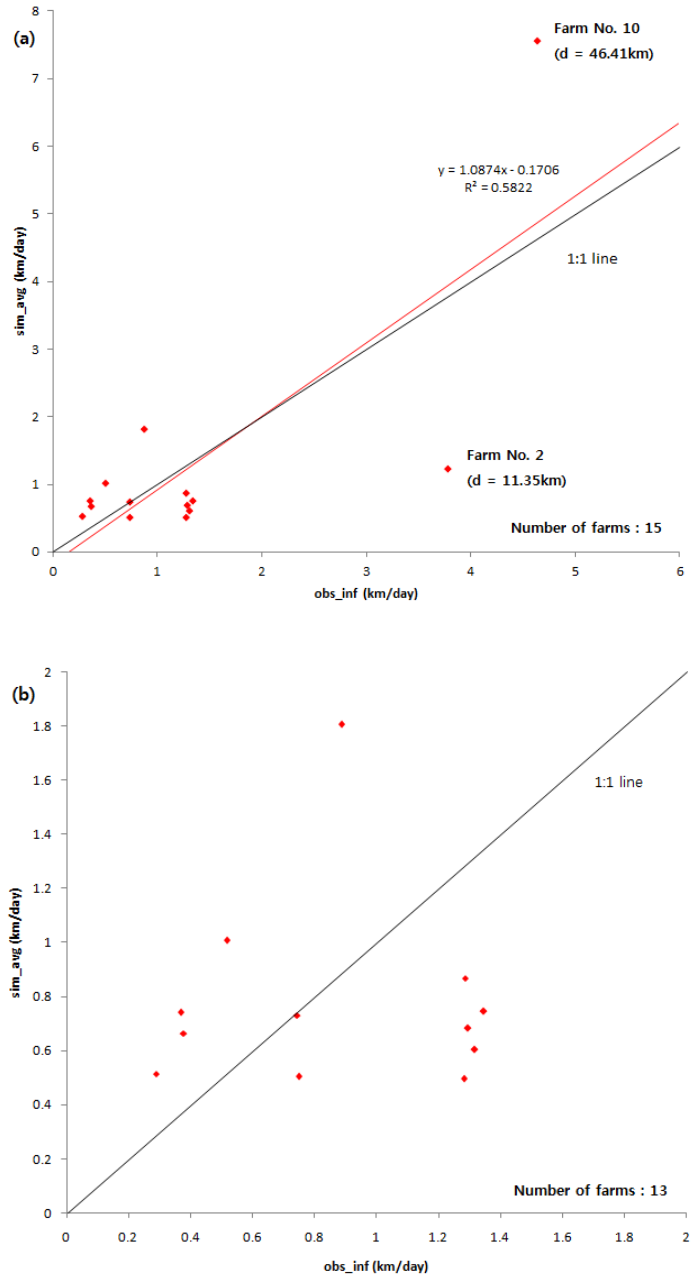


Fig. 16. Disease spreading speed of each farm from simulated and reported results of the 2016 Chungcheongnam-do epidemic case (a) with all 15 farms in a region, and (b) without 2 farms distant from the disease origin.

DISCUSSION

The FMD-IBM model would be useful to simulate the regional dispersal events of FMD. After parameters of the FMD-IBM was calibrated to perform the simulation of regional-scale disease spread, more than 75% of the intra-farm disease spread simulation were able to predict the occurrence date of infection events within two days of observation dates in the calibration set. The parameter values of inter-farm dispersal were considerably smaller for the 2016 Chungcheongnam-do case than the 2010 Andong case, which indicated that the disease dispersal was less favorable for the former than the latter. The FMD-IBM had the reasonable speed of disease spread for each infected premises during the epidemic, which ranged from 0.3 to 1.3 km per day.

Our results suggested that the FMD-IBM would be useful to estimate the speed of disease spread in a region. The FMD-IBM was designed for stochastic simulation of disease dispersal between farms. Although the probability of FMD dispersal between individual farms over distance was used, the FMD-IBM was able to characterize dispersal pattern of FMD in a region. Parameters of the model were defined to represent factors associated with the dispersal of FMD. Thus, parameter values of the FMD-IBM would be useful to assess the effectiveness of control measures used in a given epidemic. For example, parameters such as *Fsus* (i.e. susceptibility of a farm) would indicate the herd immunity of farms containing certain species of hosts during the outbreak. This suggested that the FMD-IBM can aid the decision-making in controlling of FMD outbreaks. For example, farms in a region can be prioritized for executing control measurements such as standstill or preemptive culling.

Characteristics of farms (e.g., properties of hosts) would be used to determine parameter

values of the FMD-IBM for an intra-farm dispersal. The parameter value of the infection period was greater for cattle farms than pig ones, which suggested steady releases of FMD virus from the former during the epidemic at a regional scale. In actual outbreaks of FMD, such an observation has been reported. For example, Alexandersen et al. (2003) reported that the incubation period of cattle-to-cattle disease spread was about twice as long as that of pig-to-pig disease spread. The parameter value of contact rate (*cr*) was greater for pig farms than cattle ones in both cases. Those results would be associated with the fact that a pig farm tends to have a greater number of hosts than a cattle farm in Korea.

It appeared that the preventive measures including vaccination and early quarantine were effective in the 2016 outbreak. Parameter values of inter-farm dispersal for the 2016 outbreak were 5~22 times lower than those for the epidemic in 2010. In Korea, considerable efforts have been made to establish preventive measures and to improve quarantine systems after the 2010 epidemic. For example, vaccination against FMD virus has been adopted for a large number of farms (Park et al., 2016). Therefore, it was likely that the speed of FMD spread would be considerably lower in the 2016 outbreak than the 2010 epidemic.

It was demonstrated that the FMD-IBM had a reasonable estimation of infection date for individual farms, which would allow identification of farms where immediate control would be needed. The risk of FMD outbreak of a farm could also be assessed using the FMD-IBM based on the distance from the first infected farm as well as from other adjacent premises that are susceptible to the disease. Such information would help epidemiology surveyors, analysts and decision makers to determine the type of action for control of FMD for individual farms in a region.

It is challenging to predict dispersal events at a farm distant from another farm with infected hosts. Such a farm could be infected after a series of infection occurred between farms. In the

simulation of FMD dispersal, the probability of disease spread based on the distance between farms is often used in stochastic simulations. Although an FMD dispersal model would have a relatively small error, the error would be accumulated along the pair-wise dispersal between farms.

FMD crisis could occur by a long-distance dispersal of the pathogen although the probability of such events would be considerably small. For example, Sørensen et al. (2000) reported that sudden outbreak of FMD could result from the virus transported through the atmosphere by wind. Thus, the long-distance transmission event of FMD has been ignored in the deterministic modeling of FMD (Kao, 2002). Keeling et al. (2001) used the transmission kernel that assigns a small value to the probability of disease spread between two distant farms to take into account a long dispersal event. The FMD-IBM also used the same kernel function suggested by Keeling et al. (2001) for a long-distance dispersal between farms. Still, the FMD-IBM was not able to identify a small number of farms distinct from the first site where the FMD epidemic started during a relatively short period of simulation time.

An approach based on the graph theory would be useful to simulate a long-distance disease dispersal. Individual farms and the event of disease dispersal between farms could be represented by vertices and edges, respectively (Gross and Yellen, 2005). In the FMD-IBM model, the probability of dispersal event between farms was equivalent to a weight value of edge. A long-distance dispersal event would occur between farms through livestock transportation system or atmospheric transport. Thus, a long dispersal event for the FMD could be predicted assessing trajectories of a pathogen contained in hosts or air. For example, trajectory data for vehicle transportation and wind direction could be incorporated into the FMD-IBM model to take into account the pathway of disease spread, which would allow reasonable prediction of long-distance dispersal events.

In the development of the FMD-IBM, the IBM approach allowed the impact assessment of individuals on the population-level disease outbreaks with a simple modeling framework. Individual farm-based models or hybrid models, which takes into account a livestock herd as an aggregated unit, provides little information on the intra-farm dynamics during the epidemic. In contrast, the effect of farm size and the heterogeneity of infected host ratio among farms were taken into account to simulate the infection progress of each farm in a region using the FMD-IBM. As a result, the FMD-IBM provided in-depth data during dispersal events, e.g., the occurrence date of FMD at an individual farm, which would be useful to design and assess farm-specific control measures. IBM approach on the field of epidemiology could also aid conceptualizing the processes that empirical models tend to ignore due to lack of reliable data (Auchincloss, 2008). In the case of the FMD-IBM, contacts between individual hosts as well as individual farms were represented using the modified form of Reed-Frost equations (Vynnycky, 2010), which requires no additional host-level observation data.

In spite of the advantage of the FMD-IBM in a simulation of disease dispersal, improvement of the FMD-IBM would be needed to take into account detailed host cycles. For example, the DADS model had separate procedures for the subclinical and clinical period of infected hosts, which allows the calculation of time lag between the timing of host infection and the declaration of host infection via epidemiological survey. In contrast, a specific time lag was applied to infected host in the FMD-IBM, which caused earlier culling events in simulations than in reality. This merits further development of the FMD-IBM for more realistic simulation of FMD dispersal.

CONCLUSION

We have developed an IBM that simulates the regional scale epidemic case based on the intra-farm and inter-farm FMD dispersal processes. Case studies involving 2010 Andong FMD epidemic and 2016 Chungcheongnam-do FMD epidemic indicated that the disease spread less likely occurred during the 2016 Chungcheongnam-do case, which was also reflected on the parameter sets of the FMD-IBM. The model could simulate the temporal pattern of regional scale epidemics as well as farm level disease spread. It could also estimate the speed of FMD spread by each farm in the region, which we consider useful in evaluating and planning disease control strategies for individual farms.

The concept of IBM would be effective in enabling the model to handle detailed dynamics of multilevel processes with fewer parameters, less complexity, and fewer data requirements. However, the model simulation could not provide information on the trail of disease spread, which is an important factor when control managements of the epidemic. Using trajectory data to reflect various disease transmission pathways would improve model simulation. Such modeling approach could not only help minimize the damage from livestock disease but also provide insight into understanding the basic processes of the field of epidemiology.

Further research could be performed to:

- Distinguish the avirulent and virulent stage of infected hosts to consider the time lag between discovery of host infection and confirmation of outbreak from the authorities
- Use trajectory data such as livestock vehicle movement and wind direction based on the graph theory to take various disease spreading pathways into account
- Consider interference or disease control measurement execution during the epidemic (e.g. standstill policies, vaccination)
- Apply the model to other host species or other diseases

REFERENCES

Abbey, H. “An examination of the Reed-Frost theory of epidemics.” *Human biology* 24, (1952): 201.

Alexandersen, S., Zhang, Z., Reid, S. M., Hutchings, G. H., & Donaldson, A. I. “Quantities of infectious virus and viral RNA recovered from sheep and cattle experimentally infected with foot-and-mouth disease virus O UK 2001.” *Journal of General Virology* 83, (2002): 1915-1923.

Alexandersen, S., & Mowat, N. “Foot-and-mouth disease: host range and pathogenesis.” In *Foot-and-Mouth Disease Virus*, pp. 9-42, Springer, (2005).

Auchincloss, A. H., & Roux, A. V. D. “A new tool for epidemiology: the usefulness of dynamic-agent models in understanding place effects on health.” *American journal of epidemiology* 168, (2008): 1-8.

Bates T. W., Thurmond M. C., & Carpenter T. E. “Description of an epidemic simulation model for use in evaluating strategies to control an outbreak of foot-and-mouth disease.” *American Journal of Veterinary Research* 64, (2003): 195-204.

Bicknell, K. B., Wilen, J. E., & Howitt, R. E. “Public policy and private incentives for livestock disease control.” *Australian Journal of Agricultural and Resource Economics* 43, (1999): 501-521.

Bobashev, G. V., Goedecke, D. M., Yu, F., & Epstein, J. M. “A hybrid epidemic model: combining the advantages of agent-based and equation-based approaches.” In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, pp. 1532-1537, (2007).

Bradhurst, R. A., Roche, S. E., East, I. J., Kwan, P., & Garner, M. G. “A hybrid modeling approach to simulating foot-and-mouth disease outbreaks in Australian livestock.” *Frontiers in Environmental Science* 3, (2015): 17.

Carpenter, T. E., Thurmond, M. C., & Bates, T. W. “A simulation model of intraherd transmission of foot and mouth disease with reference to disease spread before and after clinical diagnosis.” *Journal of Veterinary Diagnostic Investigation* 16, (2004): 11-16.

Charles, S., Subtil, F., Kielbassa, J., & Pont, D. “An individual-based model to describe a bullhead population dynamics including temperature variations.” *Ecological modelling* 215, (2008): 377-392.

DeAngelis, D. L., & Mooij, W. M. “Individual-based modeling of ecological and evolutionary processes.” *Annual Review of Ecology, Evolution, and Systematics*, (2005): 147-168.

Domingo, E., Baranowski, E., Escarmís, C., & Sobrino, F. “Foot-and-mouth disease virus.” *Comparative immunology, microbiology and infectious diseases* 25, (2002): 297-308.

East, M. “Regional status and approaches to control and eradication of foot and mouth disease in the Middle East and North Africa.” *Revue Scientifique Et Technique - Office International Des Epizooties* 21, (2002): 451-458.

Ferguson, N. M., Donnelly, C. A., & Anderson, R. M. “Transmission intensity and impact of control policies on the foot and mouth epidemic in Great Britain.” *Nature* 413, (2001): 542-548.

Garner, M. G., Fisher, B. S., & Murray, J. G. “Economic aspects of foot and mouth disease: perspectives of a free country, Australia.” *Revue scientifique et technique - Office International Des Epizooties* 21, (2002): 625-635.

Garner, M. G., & Beckett, S. D. “Modelling the spread of foot-and-mouth disease in Australia.” *Australian Veterinary Journal* 83, (2005): 758-766.

Gloster, J., Sellers, R. F., & Donaldson, A. I. “Long distance transport of foot-and-mouth disease virus over the sea.” *The Veterinary Record* 110, (1982): 47-52.

Gloster, J., Williams, P., Doel, C., Esteves, I., Coe, H., & Valarcher, J. F. “Foot-and-mouth disease–Quantification and size distribution of airborne particles emitted by healthy and infected pigs.” *The veterinary journal* 174, (2007): 42-53.

Green, D. M., Kiss, I. Z., & Kao, R. R. “Modelling the initial spread of foot-and-mouth disease through animal movements.” *Proceedings of the Royal Society of London B: Biological Sciences* 273, (2006): 2729-2735.

Grimm, V., & Railsback, S. F. “Pattern-oriented modelling: a ‘multi-scope’ for predictive systems ecology.” *Philosophical Transactions of the Royal Society of London B* 367, (2012): 298-310.

Gross, J. L., & Yellen, J. *Graph theory and its applications*. CRC press. p. 1, (2005).

Hare, M., & Deadman, P. “Further towards a taxonomy of agent-based simulation models in environmental management.” *Mathematics and computers in simulation* 64, (2004): 25-40.

Hogeweg, P., & Hesper, B. “Individual-oriented modelling in ecology.” *Mathematical and Computer Modelling* 13, (1990): 83-90.

Huston, M., DeAngelis, D., & Post, W. “New computer models unify ecological theory.” *BioScience*, (1988): 682-691.

Joo, Y. S., An, S. H., Kim, O. K., Lubroth, J., & Sur, J. H. “Foot-and-mouth disease eradication efforts in the Republic of Korea.” *Canadian journal of veterinary research* 66, (2002): 122-124.

Kao, R. R. “The role of mathematical modelling in the control of the 2001 FMD epidemic in the UK.” *Trends in microbiology* 10, (2002): 279-286.

Keeling, M. J., Woolhouse, M. E., Shaw, D. J., Matthews, L., Chase-Topping, M., Haydon, D. T., Cornell, S. J., Kappey, J., Wilesmith J. & Grenfell, B. T. "Dynamics of the 2001 UK foot and mouth epidemic: stochastic dispersal in a heterogeneous landscape." *Science* 294, (2001): 813-817.

Keeling, M. J. "Models of foot-and-mouth disease." *Proceedings of the Royal Society of London B: Biological Sciences* 272, (2005): 1195-1202.

Kiss, I. Z., Green, D. M., & Kao, R. R. "The network of sheep movements within Great Britain: network properties and their implications for infectious disease spread." *Journal of the Royal Society Interfac*e 3, (2006): 669-677.

Kostova-Vassilevska, T., Consultants, L., Bates, T., Thurmond, M., & Carpenter, T. "On the use of models to assess foot-and-mouth disease transmission and control." *United States. Department of Energy*. (2004).

Morris, R. S., Wilesmith, J. W., Stern, M. W., Sanson, R. L., & Stevenson, M. A. "Predictive spatial modelling of alternative control strategies for the foot-and-mouth disease epidemic in Great Britain, 2001." In *II International Symposium on Application of Modelling as an Innovative Technology in the Agri-Food Chain; MODEL-IT* 566, pp. 337-347, (2001).

Park, J. H., Lee, K. N., Ko, Y. J., Kim, S. M., Lee, H. S., Shin, Y. K., Sohn, H. J., Park, J. Y., Yeh, J. Y., Lee, Y. H., Kim, M. J., Joo, Y. S., Yoon, H, Yoon, S. S., Cho, I. S., & Kim, B. "Control of foot-and-mouth disease during 2010-2011 epidemic, South Korea." *Emerging infectious diseases* 19, (2013): 655-660.

Park, M. E., Lee, S. Y., Kim, R. H., Ko, M. K., Park, J. N., Lee, K. N., Kim, S. M., Choi, J. H., You, S. H., Kim, B., Lee, J. S., & Park, J. H. "Altered adjuvant of foot-and-mouth disease vaccine improves immune response and protection from virus challenge." *Trials in Vaccinology* 5, (2016): 97-104.

Paton, D. J., King, D. P., Knowles, N. J., Hammond, J. "Recent spread of foot-and-mouth disease in the Far East." *Veterinary Record* 166, (2010): 569-570.

Reeves, A., Talbert, M., Salman, M.D. and Hill, A.E. "Development of a stochastic, individual-based modeling framework for within-unit transmission of highly infectious animal diseases." *Preventive Veterinary Medicine*, Submitted, http://www.naadsm.org/wh, (2013).

Sanson, R. L., Morris, R. S., & Stern, M. W. "EpiMAN-FMD: a decision support system for managing epidemics of vesicular disease." *Revue scientifique et technique - Office International Des Epizooties* 18, (1999): 593-605.

Sørensen, J. H., Mackay, D. K. J., Jensen, C. Ø., & Donaldson, A. I. "An integrated model to predict the atmospheric spread of foot-and-mouth disease virus." *Epidemiology & Infection* 124, (2000): 577-590.

Sørensen, J. H., Donaldson, A. I., Alexandersen, S., & Mikkelsen, T. "Relative risks of the uncontrollable (airborne) spread of FMD by different species." *The Veterinary Record* 148, (2001): 602-604.

Vosloo, W., Bastos, A. D. S., Sangare, O., Hargreaves, S. K., & Thomson, G. R. "Review of the status and control of foot and mouth disease in sub-Saharan Africa." *Revue Scientifique Et Technique - Office International Des Epizooties* 21, (2002): 437-445.

Vynnycky, E., & White, R. *An introduction to infectious disease modelling*. Oxford University Press. p. 152, (2010).

Zurell, D., Eggers, U., Kaatz, M., Rotics, S., Sapir, N., Wikelski, M., Nathan, R., & Jeltsch, F. "Individual-based modelling of resource competition to predict density-dependent population dynamics: a case study with white storks." *Oikos* 124, (2015): 319-330.

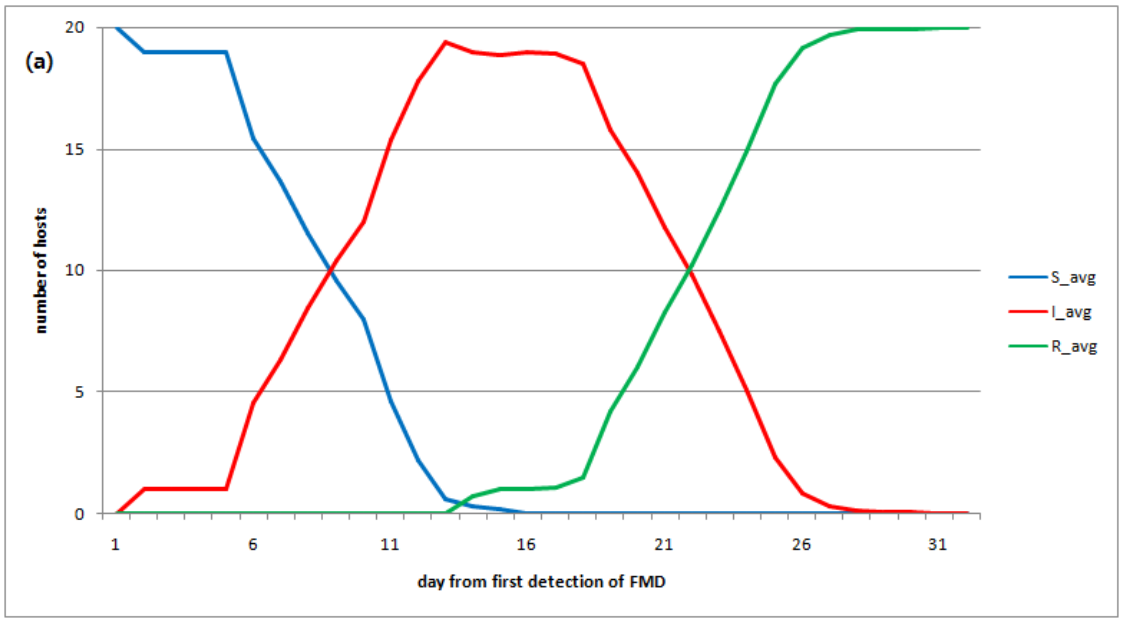
APPENDIX A

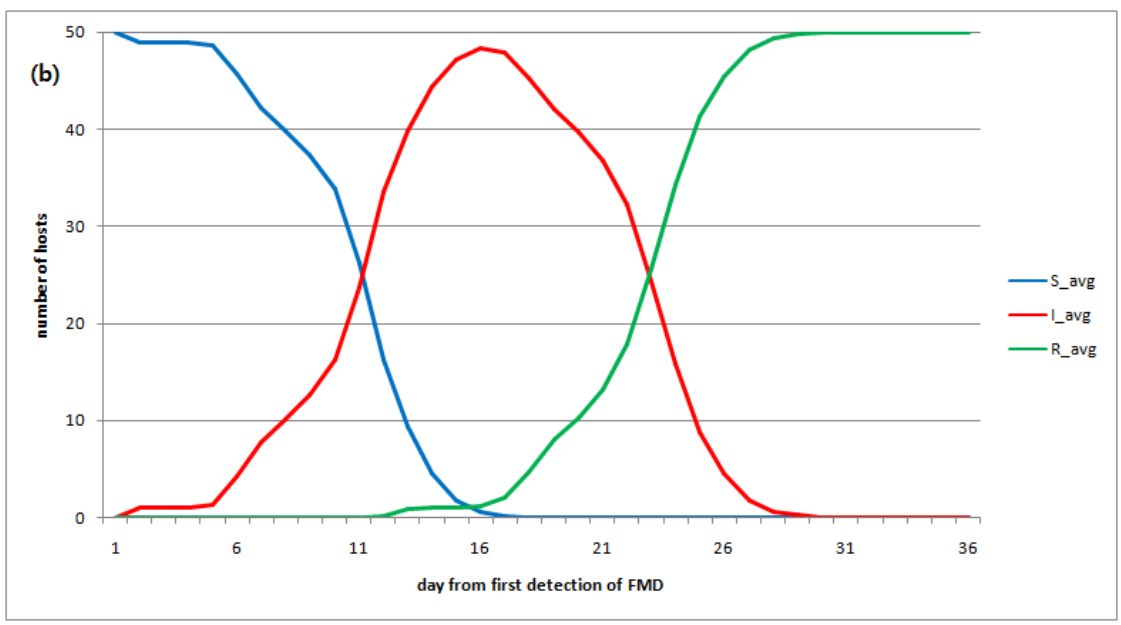
Sensitivity analysis of intra-farm disease spread process

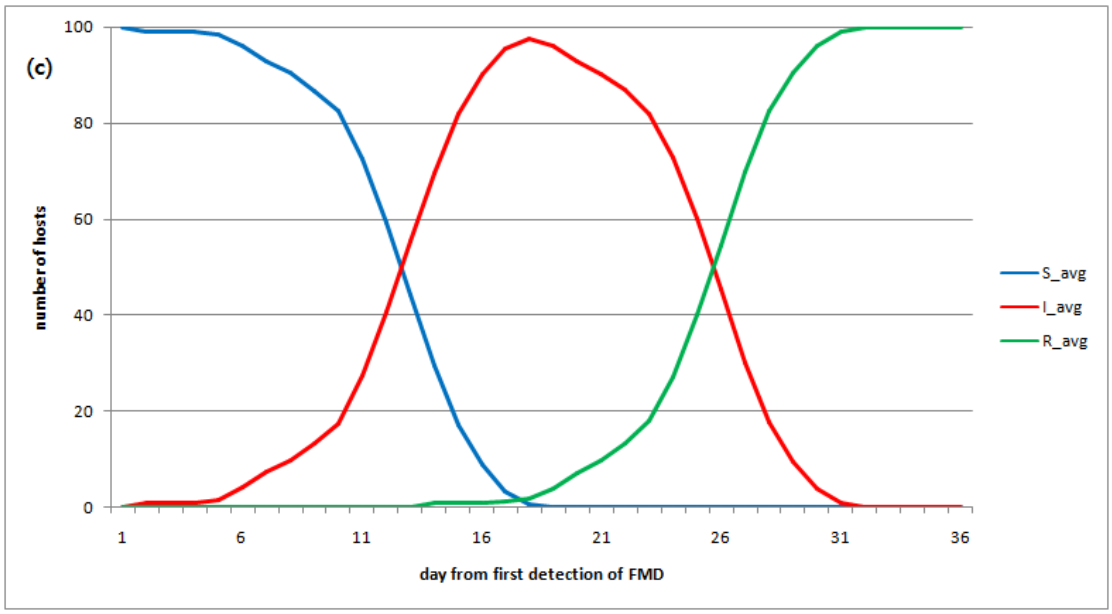
The size of a livestock farm would affect the intra-farm level disease spread during the epidemic. We conducted a sensitivity analysis on the intra-farm disease spread process to examine the impact of the farm size, which is determined by the number of the host it contains. The initial number of hosts in farms with different sizes were assigned as shown in Table Al. The within-farm disease spread dynamics of each farm were simulated by the FMD-IBM, and the number of susceptible, infected, and recovered hosts through time were retrieved (Fig. A1). The values were averaged from 50 repeated simulations for each farm.

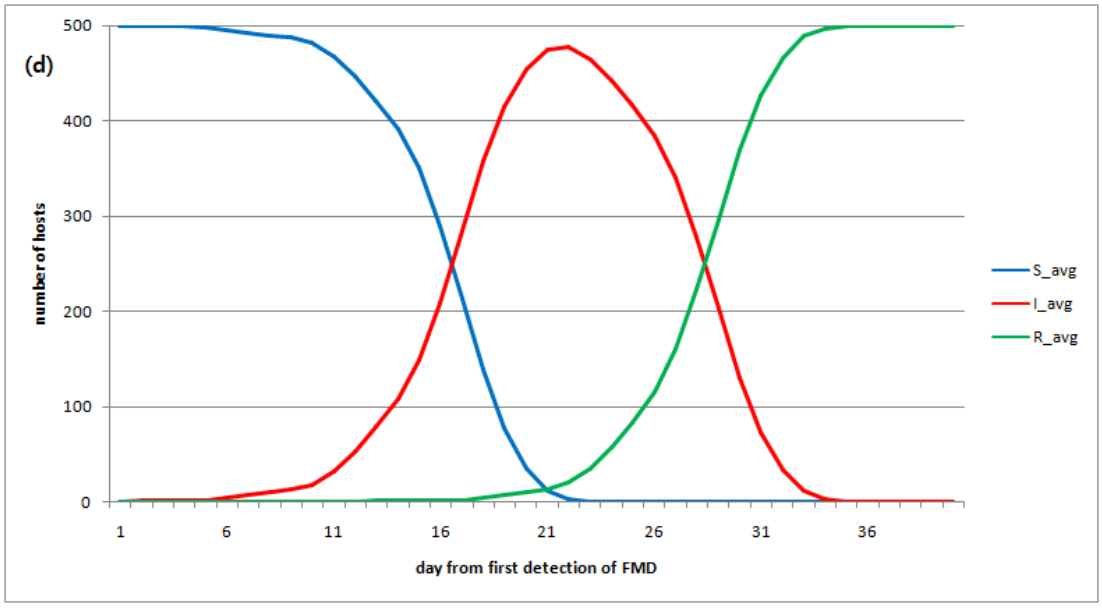
Table A1. Initial number of hosts in farms with different sizes

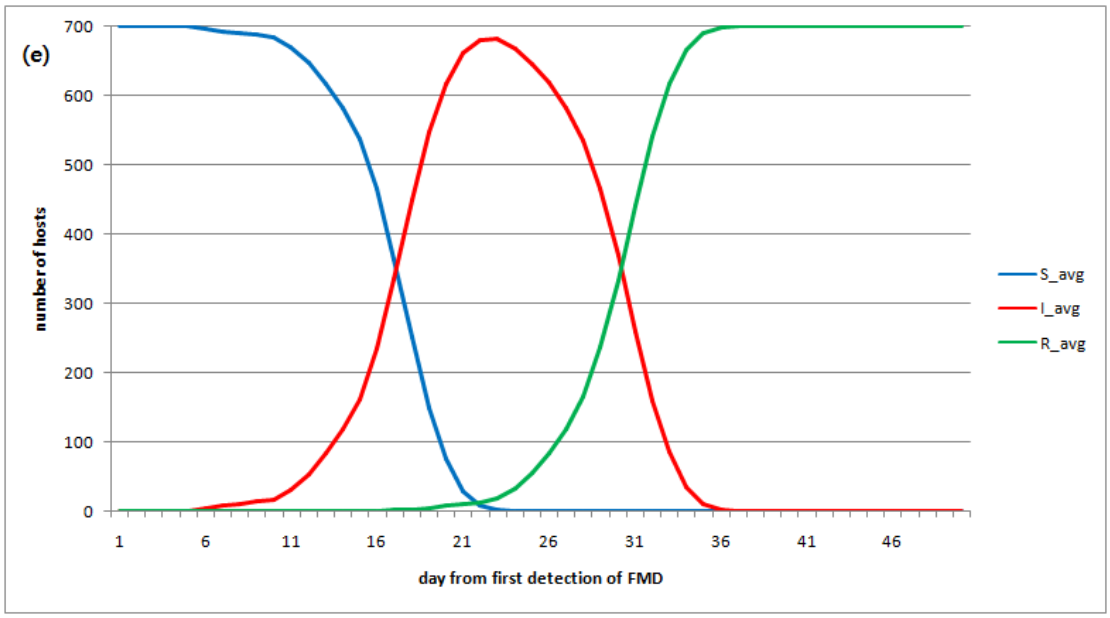
	Cattle farm	Pig farm
Small-sized	20	50
Mediocre-sized	100	500
Large-sized	700	7000


(a)
20
15
10
5
0
number of hosts
1
6
11
16
21
26
31
day from first detection of FMD
S_avg
I_avg
R_avg


(b)
50
40
30
20
10
0
number of hosts
1
6
11
16
21
26
31
36
day from first detection of FMD
S_avg
I_avg
R_avg


(c)
100
80
60
40
20
0
number of hosts
1
6
11
16
21
26
31
36
day from first detection of FMD
S_avg
I_avg
R_avg


(d)
500
400
300
200
100
0
number of hosts
1
6
11
16
21
26
31
36
day from first detection of FMD
S_avg
I_avg
R_avg



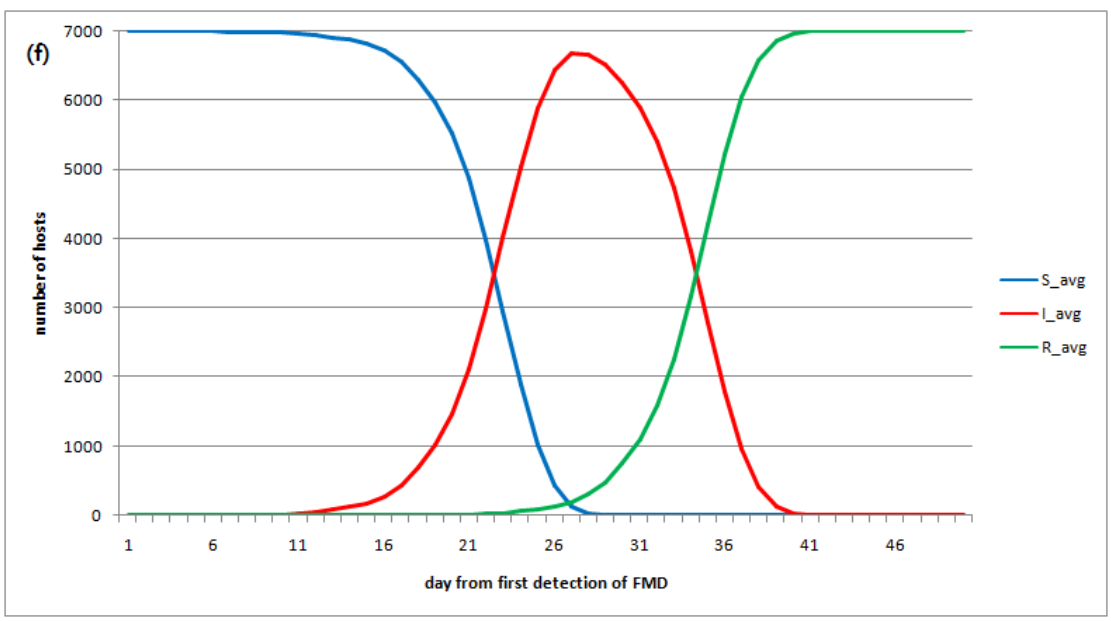
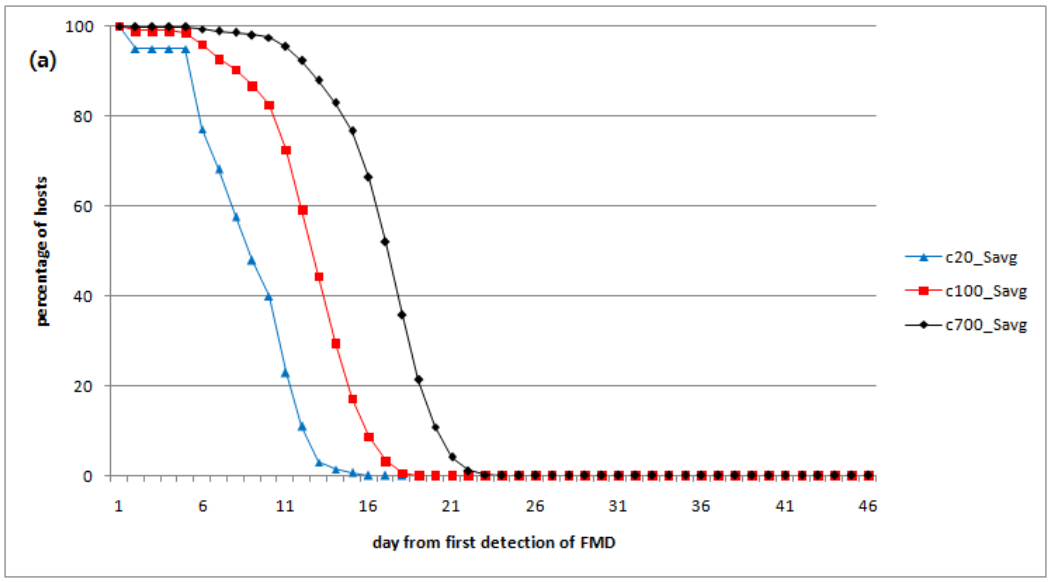
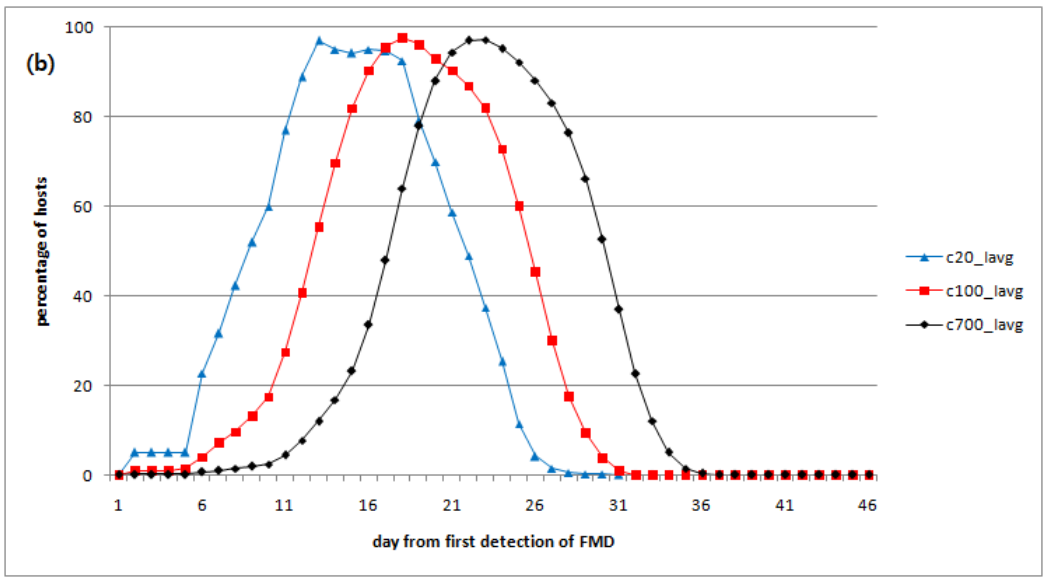
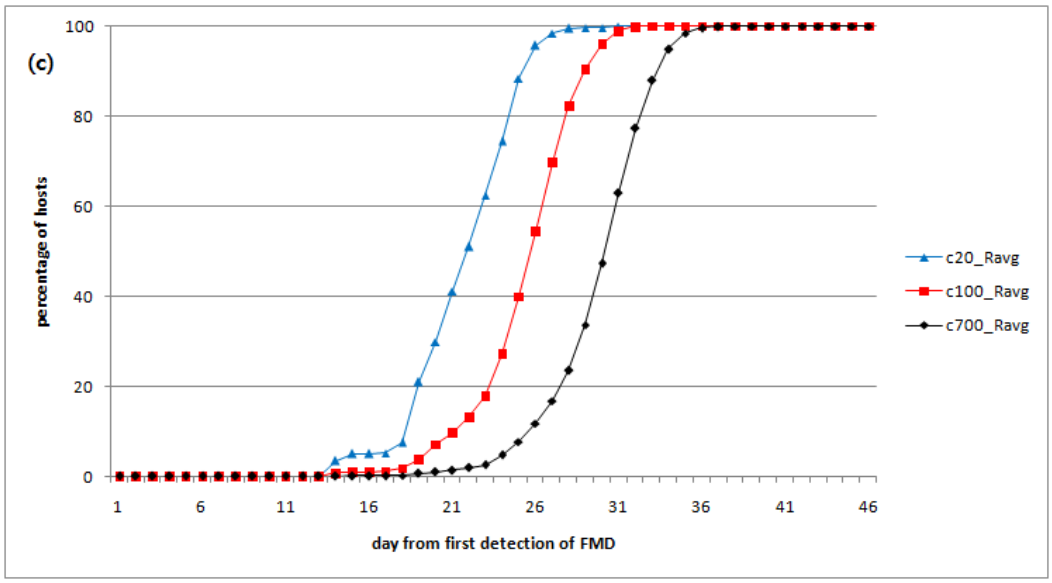


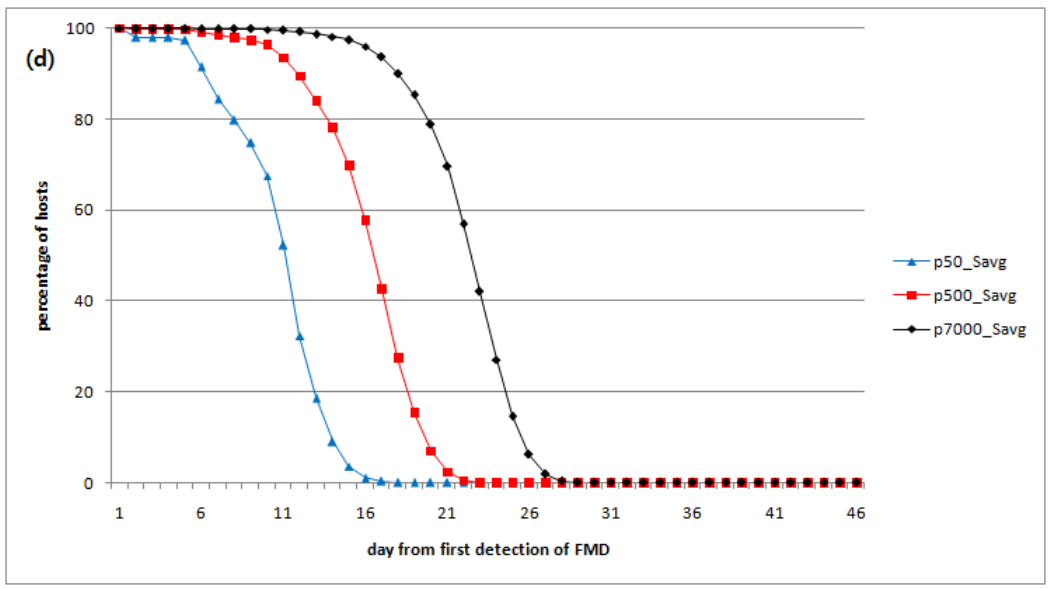
Fig. A1. The average number of susceptible, infected, and recovered hosts through time in various farms. The corresponding farms contained the following hosts at the beginning of the simulation: (a) 20 cattle, (b) 50 pigs, (c) 100 cattle, (d) 500 pigs, (e) 700 cattle, and (f) 7000 pigs.

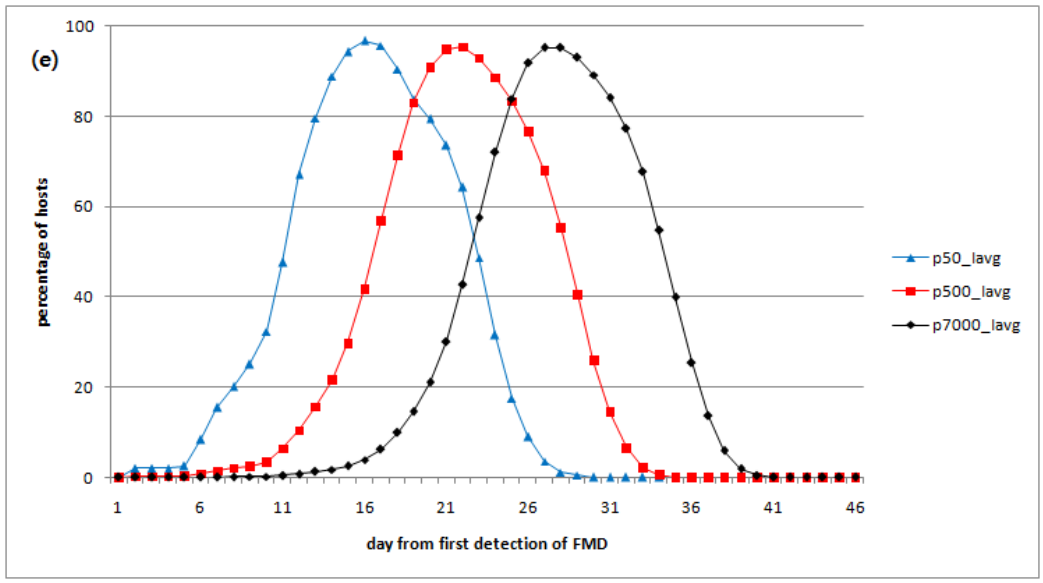
The results showed that the event emergence is likely to occur late in farms with a large number of hosts. There was a gap of about 5 days between the event dates of small, mediocre, and large-sized farms (Fig. A2). However, the pattern of increase and decrease in host number was similar regardless of the farm size when the values are scaled into percentages. In this study, we calibrated the intra-farm disease spread process of the model to fit the epidemic dynamics of a large-sized farm, which was the same condition as the calibration sets were generated.


(a)
percentage of hosts
day from first detection of FMD
c20_Savg
c100_Savg
c700_Savg


(b)
percentage of hosts
day from first detection of FMD
c20_Iavg
c100_Iavg
c700_Iavg


(c)
100
80
60
40
20
0
percentage of hosts
1
6
11
16
21
26
31
36
41
46
day from first detection of FMD
c20_Ravg
c100_Ravg
c700_Ravg


(d)
100
80
60
40
20
0
percentage of hosts
1
6
11
16
21
26
31
36
41
46
day from first detection of FMD
p50_Savg
p500_Savg
p7000_Savg



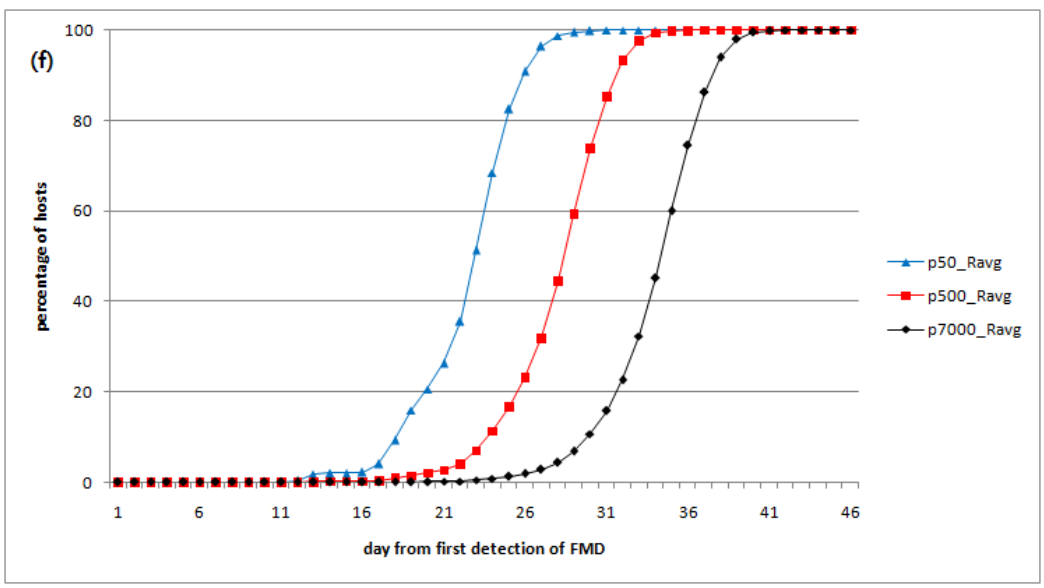
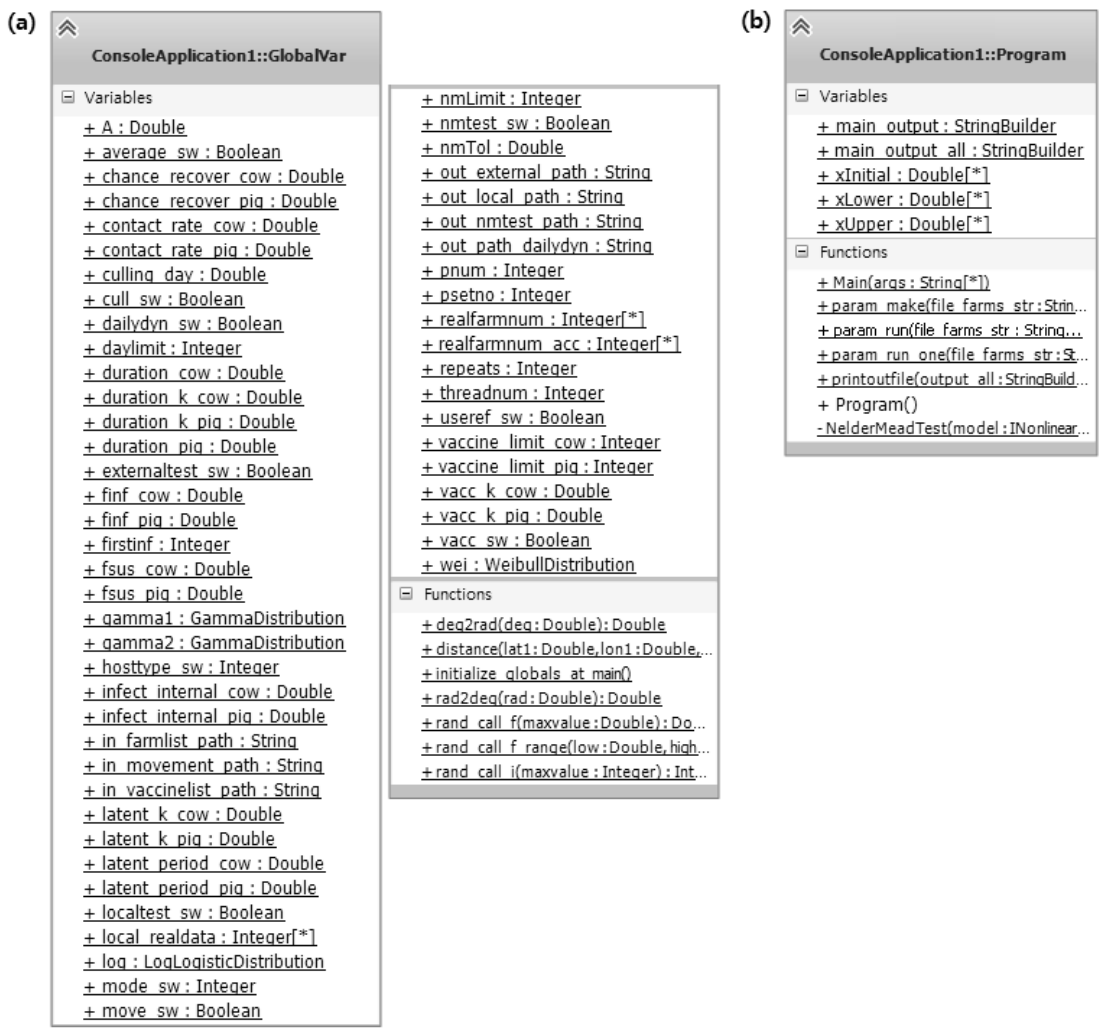


Fig. A2. Comparison of the average percentage of susceptible, infected, and recovered hosts through time between farms with different sizes. All the values were scaled as percentages of the total host number regardless of the farm size. (a), (b), and (c) are describing the average number of susceptible, infected, and recovered hosts through time in cattle farms, respectively, while (d), (e), and (f) are describing the counterpart in pig farms. The blue, red, and black lines correspond to the values from small, mediocre, and large-sized farms, respectively.

APPENDIX B

Classes of FMD-IBM

Some components in the model are implemented but not used in this study, e.g. option for applying farm vaccination on a designated schedule.



(c)


ConsoleApplication1::FMD_IBM
Variables
+ A : Double
+ average_sw : Boolean
+ chance_recover_cow : Double
+ chance_recover_pig : Double
+ contact_rate_cow : Double
+ contact_rate_pig : Double
+ culling_day : Double
+ cull_sw : Boolean
+ dailydyn_sw : Boolean
+ daylimit : Integer
+ duration_cow : Double
+ duration_k_cow : Double
+ duration_k_pig : Double
+ duration_pig : Double
+ externaltest_sw : Boolean
+ finf_cow : Double
+ finf_pig : Double
+ firstinf : Integer
+ fsus_cow : Double
+ fsus_pig : Double
+ gamma1 : GammaDistribution
+ gamma2 : GammaDistribution
+ hosttype_sw : Integer
+ infect_internal_cow : Double
+ infect_internal_pig : Double
+ latent_k_cow : Double
+ latent_k_pig : Double
+ latent_period_cow : Double
+ latent_period_pig : Double
+ localtest_sw : Boolean
+ local_realdata : Integer[*]
+ log : LogLogisticDistribution
+ mode_sw : Integer
+ move_sw : Boolean
+ out_path_dailydyn : String
+ realfarmnum_acc : Integer[*]
+ repeats : Integer
+ useref_sw : Boolean
+ vaccine_limit_cow : Integer
+ vaccine_limit_pig : Integer
+ vacc_k_cow : Double
+ vacc_k_pig : Double
+ vacc_sw : Boolean
+ wei : WeibullDistribution
+ _disposed : Boolean
- cowpop : Integer[*]
- farms : farm[*]
- hostpop : Integer[*]
- movements : traj[*]
- pigpop : Integer[*]
- trackers : Integer[*]
- vaccines : vaccinedata[*]
Functions
+ CORE(iter : Integer, pset : Doub...
+ culling(day : Integer)
+ deg2rad(deg : Double) : Double
+ Dispose()
+ distance(lat1 : Double, lon1 : Double,...
+ FMD_IBM()
+ give_vaccine(day : Integer)
+ go(d : Integer[*], trackers : Inte...
+ loadInput(file_farms_str : String...
+ model_run(pset : Double[*], pse...
+ model_run_neldermead(pset : D...
+ move()
+ rad2deg(rad : Double) : Double
+ rand_call_f(maxvalue : Double) : Do...
+ rand_call_f_range(low : Double, high...
+ rand_call_i(maxvalue : Integer) : Int...
+ renew()
+ save_results_infdate(iter : Integ...
+ save_results_local(iter : Integer,...
+ save_results_new(iter : Integer,...
+ update(day : Integer, d : Intege...
- ~FMD_IBM()
Dispose(disposing : Boolean)

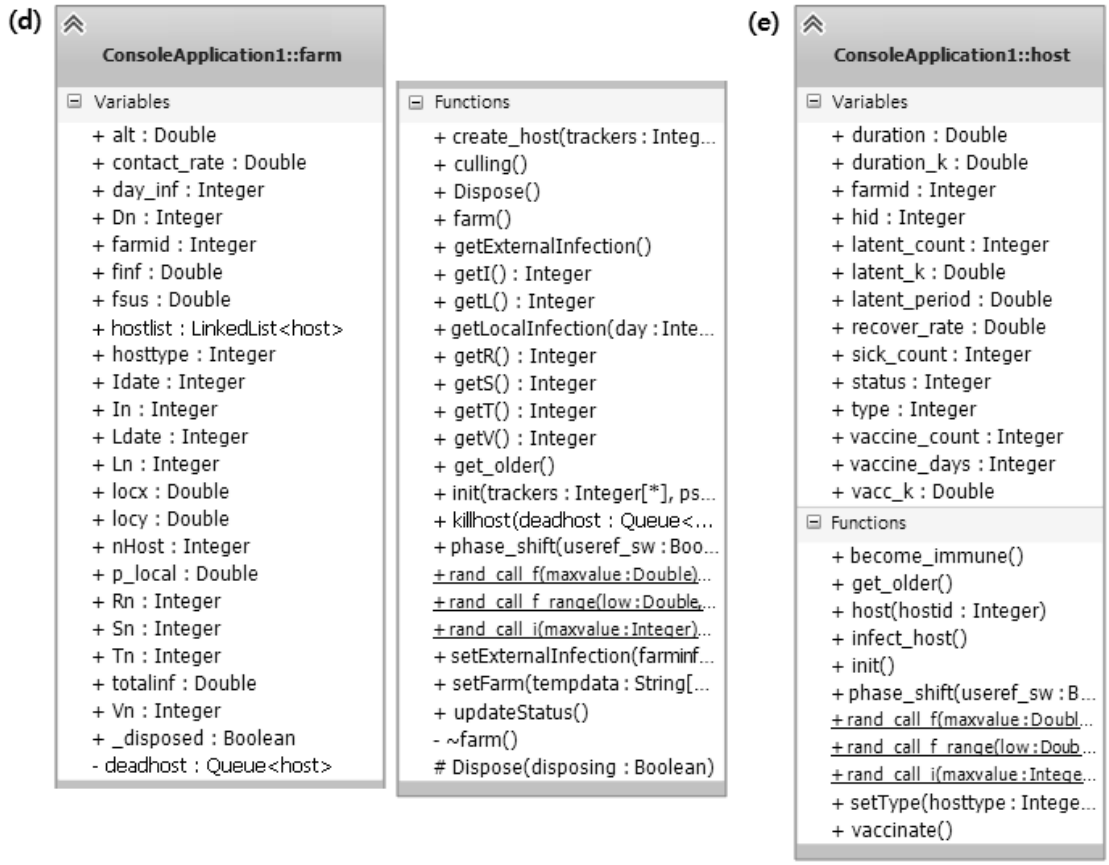


Fig. A3. Full list of class properties implemented in the code

APPENDIX C

Source codes of FMD-IBM

FMD-IBM consists of 5 files (GlobalVar.cs, Program.cs, FMD_IBM.cs, farm.cs, host.cs) each containing different modules and functions for the corresponding classes it manages.

[GlobalVar.cs]

```
using System;
using System.IO;

using Accord.Statistics.Distributions.Univariate;

namespace ConsoleApplication1
{
  public static class GlobalVar
  {
    /* real data for comparison ... from UNIST model */

    public static int[] local_realdata;         //local infection graph emergence data (0, 1, 2 : cow / 3, 4, 5 :
pig)
    public static int[] realfarmnum;            //real data of daily infected farm number ... NOT USED
    public static int[] realfarmnum_acc;        //real data of daily ACCUMULATED infected farm number

    /* statistical distribution for random numbers used in phase shift ... from reference */

    public static WeibullDistribution wei;      //cumulative probability distribution for phase shift (L -> I) of
cow
    public static GammaDistribution gamma1;     //cumulative probability distribution for phase shift (L -> I)
of pig
    public static GammaDistribution gamma2;     //cumulative probability distribution for phase shift (I -> R
or D) of cow
    public static LogLogisticDistribution log;  //cumulative probability distribution for phase shift (I -> R or
D) of pig
```

```
	/* variable used for NelderMead test */

	public static bool nmtest_sw;		//switch variable whether to use Nelder-Mead test
	public static int nmLimit;		//iteration limit of NelderMead test
	public static double nmTol;		//tolerance rate of NelderMead test
	public static int pnum;		//number of parameters for fitting through Nelder-Mead test

	public static int psetno;		//id of parameter set being tested in NelderMead test mode ... NOT AN INPUT

	/* simulation options (fixed value) */

	public static int threadnum;		//core number of computer (used to define the number of plist in pdb for parallel computing)
	public static int repeats;		//simulation repeating number
	public static int firstinf;		//number of farm(s) infected on day 0

	public static string in_farmlist_path;	//input file (farmlist) path
	public static string in_movement_path;	//input file (movement) path
	public static string in_vaccinelist_path;	//input file (vaccinelist) path
	public static string out_path_dailydyn;	//output file (result) path for recording daily dynamics ... NOT USED
	public static string out_local_path;	//output file (result) path for recording local test graph emergence date ... used to validate local infection
	public static string out_external_path;	//output file (result) path for recording farm infection date ... used to validate external infection
	public static string out_nmtest_path;	//output file (result) path for recording result of Nelder-Mead test (parameter fitting)
	public static int daylimit;		//simulation day limit (day)
	public static bool move_sw;		//switch variable whether to use movement data ... THIS OPTION IS NOT SUPPORTED
	public static bool vacc_sw;		//switch variable whether to apply vaccination
	public static bool cull_sw;		//switch variable whether to apply culling ... put this ON during external test, OFF during local test

	public static bool localtest_sw;		//switch variable whether to test local infection
	public static bool dailydyn_sw;		//switch variable whether to print out daily dynamics data ... only works with 1 repeat simulation
	public static bool average_sw;		//switch variable whether to print out average results of all simulation ... only works with localtest
	public static bool useref_sw;		//switch variable whether to use reference data for latent period and duration ... pnum should be 2 if this is ON, 5 if this is OFF
	public static int hosttype_sw;		//switch variable determining the type of host in local test (1 : cow / 2 : pig)

	public static bool externaltest_sw;		//switch variable whether to test external infection
	public static int mode_sw;		//switch variable of mode
				//0 : print out number of infected farms
				//1 : print out difference of infection date of each farms ... only works when dailydyn_sw = FALSE (?)
```

```
//2 : print out infection date of each farms

    /* global parameters (fixed value) */

    //host factors
    public static int vaccine_limit_cow;        //duration of vaccine effect to cow (day)
    public static int vaccine_limit_pig;        //duration of vaccine effect to pig (day)

    public static double infect_internal_cow;   //internal infect rate between cow to cow (%) ..... becomes
p_local value of cow farms
    public static double infect_internal_pig;   //internal infect rate between pig to pig (%) ..... becomes
p_local value of pig farms
    public static double contact_rate_cow;      //parameter for contact rate in a cow farm
    public static double contact_rate_pig;      //parameter for contact rate in a pig farm
    public static double latent_period_cow;     //average latent period of cow (day) ..... using incubation
period from review note
    public static double latent_period_pig;     //average latent period of pig (day) ..... using incubation
period from review note
    public static double duration_cow;          //average duration until cow dies or recovers after infection
(day) ..... not important if culling option is on
    public static double duration_pig;          //average duration until pig dies or recovers after infection
(day) ..... not important if culling option is on
    public static double chance_recover_cow;    //recover rate of cow (%)
    public static double chance_recover_pig;    //recover rate of pig (%)

    public static double latent_k_cow;          //parameter for L->I of cow (steepness of sigmoid curve)
    public static double latent_k_pig;          //parameter for L->I of pig (steepness of sigmoid curve)
    public static double duration_k_cow;        //parameter for I->RorD of cow (steepness of sigmoid curve)
    public static double duration_k_pig;        //parameter for I->RorD of pig (steepness of sigmoid curve)
    public static double vacc_k_cow;            //parameter for V->S of cow (steepness of sigmoid curve)
    public static double vacc_k_pig;            //parameter for V->S of pig (steepness of sigmoid curve)

    public static double finf_cow;              //external infectibility of cow farm (%)
    public static double finf_pig;              //external infectibility of pig farm (%)
    public static double fsus_cow;              //external susceptibility of cow farm (%)
    public static double fsus_pig;              //external susceptibility of pig farm (%)

    //others
    public static double A;                     //parameter for external infection
    public static double culling_day;           //average days when infected farms get culled after first
occurence of L host (day) ..... this is important instead of duration

    /* tracking variables (not fixed value) */
    /*
    //counters
    public static int uhid;                     //universal id of hosts

    //statistics
    public static int nFarm;                    //number of farms (n)
    public static int Lfarms;                   //number of farms with L (n)
    public static int Ifarms;                   //number of farms with I (n)

    public static int cownum_init;              //initial number of cows (n)
```

```
    public static int pignum_init;            //initial number of pigs (n)
    public static int hostnum_init;           //initial number of hosts (n)
    public static int cownum;                 //number of cows (n)
    public static int pignum;                 //number of pigs (n)
    public static int hostnum;                //total number of hosts (n)


    public static double Scow;                //healthy cows (n)
    public static double Vcow;                //vaccinated cows (n)
    public static double Lcow;                //latent cows (n)
    public static double Icow;                //infected cows (n)
    public static double Rcow;                //immune cows (n)
    public static double Dcow;                //dead cows (n)
    public static double Spig;                //healthy pigs (n)
    public static double Vpig;                //vaccinated pigs (n)
    public static double Lpig;                //latent pigs (n)
    public static double Ipig;                //infected pigs (n)
    public static double Rpig;                //immune pigs (n)
    public static double Dpig;                //dead pigs (n)
    public static double Shost;               //healthy pigs (n)
    public static double Vhost;               //vaccinated pigs (n)
    public static double Lhost;               //latent pigs (n)
    public static double Ihost;               //infected pigs (n)
    public static double Rhost;               //immune pigs (n)
    public static double Dhost;               //dead hosts (n)
    */


    public static void initialize_globals_at_main() //only initialize once at start
    {
        GlobalVar.local_realdata = new int[6] { 10, 19, 25, 16, 24, 31 };                    //local infection
graph emergence data (0, 1, 2 : cow / 3, 4, 5 : pig)


        GlobalVar.firstinf = 1;


        //2010 Andong Data
        /*
        GlobalVar.realfarmnum = new int[20] { 1, 0, 1, 1, 0, 3, 12, 10, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0 };              //real data of daily infected farm number ... not used
        GlobalVar.realfarmnum_acc = new int[20] { 1, 1, 2, 3, 3, 6, 18, 28, 30, 30, 30, 30, 30, 30, 30, 30,
30, 30, 30 };    //real data of daily infected farm number (accumulated)
        */


        //2016 Chungnam Data
        GlobalVar.realfarmnum = new int[50]
        {
            2, 0, 0, 0, 0, 0, 0, 1, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
            0, 0, 1, 5, 0, 0, 2, 0, 0, 1,
            0, 1, 0, 3, 0, 0, 1, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        };
        GlobalVar.realfarmnum_acc = new int[50]
        {
            2, 2, 2, 2, 2, 2, 2, 3, 3, 3,
            3, 3, 3, 3, 3, 3, 3, 3, 3, 4,
            4, 4, 5, 10, 10, 10, 12, 12, 12, 13,
```

```
            13, 14, 14, 17, 17, 17, 18, 18, 18, 18,
            18, 18, 18, 18, 18, 18, 18, 18, 18, 18
        };

        GlobalVar.wei = new WeibullDistribution(scale: 3.974, shape: 1.782);     //cow - latent
        GlobalVar.gamma1 = new GammaDistribution(theta: 1/1.914, k: 1.617);      //pig - latent
        GlobalVar.gamma2 = new GammaDistribution(theta: 1/1.107, k: 3.969);      //cow - infectious
        GlobalVar.log = new LogLogisticDistribution(5.474, 5.39);                //pig - infectious


        GlobalVar.nmtest_sw = false;
        GlobalVar.nmLimit = 10000;       //should be bigger than 1000
        GlobalVar.nmTol = 0.1;           //should be smaller than 1.0
        GlobalVar.pnum = 6;              //2 for localtest with ref, 6 for localtest without ref, 6 for externaltest


        GlobalVar.psetno = 0;            //NOT AN INPUT (id for every parameter sets, also used to check
number of loops in NelderMead test)


        GlobalVar.threadnum = Environment.ProcessorCount;
        GlobalVar.repeats = 100;         //should be at least 100~500
        GlobalVar.in_farmlist_path = Directory.GetCurrentDirectory() + "/input_farmlist.txt";
        GlobalVar.in_movement_path = Directory.GetCurrentDirectory() + "/input_movement.txt";
        GlobalVar.in_vaccinelist_path = Directory.GetCurrentDirectory() + "/input_vaccinelist.txt";
        GlobalVar.out_path_dailydyn = Directory.GetCurrentDirectory() + "/result_dailydyn.txt";
        GlobalVar.out_local_path = Directory.GetCurrentDirectory() + "/result_local.txt";
        GlobalVar.out_external_path = Directory.GetCurrentDirectory() + "/result_external.txt";
        GlobalVar.out_nmtest_path = Directory.GetCurrentDirectory() + "/result_nmtest.txt";
        GlobalVar.daylimit = 70;         //70 for local test (7000 pig farm & 700 cow farm), 20 for external test
(only Andong farms)
        GlobalVar.move_sw = false;
        GlobalVar.vacc_sw = false;
        GlobalVar.cull_sw = false;       //put this OFF during local test, ON during external test


        GlobalVar.localtest_sw = true;
        GlobalVar.dailydyn_sw = false;
        GlobalVar.average_sw = true;
        GlobalVar.hosttype_sw = 2;       //1 for cow, 2 for pig (only works for local test)


        GlobalVar.externaltest_sw = false;
        GlobalVar.mode_sw = 0;


        GlobalVar.vaccine_limit_cow = 7;
        GlobalVar.vaccine_limit_pig = 7;


        /*
        GlobalVar.infect_internal_cow = 0.07;    //fixed
        GlobalVar.contact_rate_cow = 9.9;        //fixed
        GlobalVar.latent_period_cow = 4.8;       //fixed
        GlobalVar.duration_cow = 9.0;            //fixed
        GlobalVar.latent_k_cow = 5.7;            //fixed
        GlobalVar.duration_k_cow = 4.1;          //fixed
        */
        GlobalVar.infect_internal_cow = 0.13;    //newly fixed
        GlobalVar.contact_rate_cow = 23.5;       //newly fixed
```

```
    GlobalVar.latent_period_cow = 3.4;       //newly fixed
    GlobalVar.duration_cow = 8.65;           //newly fixed
    GlobalVar.latent_k_cow = 5.9;            //newly fixed
    GlobalVar.duration_k_cow = 4.4;          //newly fixed

    /*
    GlobalVar.infect_internal_pig = 0.05;    //fixed
    GlobalVar.contact_rate_pig = 14.1;       //fixed
    GlobalVar.latent_period_pig = 6.7;       //fixed
    GlobalVar.duration_pig = 7.8;            //fixed
    GlobalVar.latent_k_pig = 5.4;            //fixed
    GlobalVar.duration_k_pig = 5.5;          //fixed
    */
    GlobalVar.infect_internal_pig = 0.11;    //newly fixed
    GlobalVar.contact_rate_pig = 27.0;       //newly fixed
    GlobalVar.latent_period_pig = 3.4;       //newly fixed
    GlobalVar.duration_pig = 7.4;            //newly fixed
    GlobalVar.latent_k_pig = 5.0;            //newly fixed
    GlobalVar.duration_k_pig = 5.6;          //newly fixed

    GlobalVar.vacc_k_cow = 4.0;              //not used
    GlobalVar.vacc_k_pig = 4.0;              //not used

    GlobalVar.chance_recover_cow = 0.95;     //not important
    GlobalVar.chance_recover_pig = 0.65;     //not important

    GlobalVar.finf_cow = 0.40;
    GlobalVar.finf_pig = 0.25;
    GlobalVar.fsus_cow = 0.20;
    GlobalVar.fsus_pig = 0.20;

    GlobalVar.A = 9999;
    GlobalVar.culling_day = 2.0;
}

/*
public static void initialize_globals() //initialize for every simulation repeats
{
    GlobalVar.uhid = 0;

    GlobalVar.nFarm = 0;    //get value by entries in input file
    GlobalVar.Lfarms = 0;   //number of farms with L (n)
    GlobalVar.Ifarms = 0;   //number of farms with I (n)

    GlobalVar.cownum_init = 0;    //get value by entries in input file
    GlobalVar.pignum_init = 0;    //get value by entries in input file
    GlobalVar.hostnum_init = 0;   //get value by entries in input file
    GlobalVar.cownum = 0;
    GlobalVar.pignum = 0;
    GlobalVar.hostnum = 0;

    GlobalVar.Scow = 0;
    GlobalVar.Vcow = 0;
```

```
        GlobalVar.Lcow = 0;
        GlobalVar.Icow = 0;
        GlobalVar.Rcow = 0;
        GlobalVar.Dcow = 0;
        GlobalVar.Spig = 0;
        GlobalVar.Vpig = 0;
        GlobalVar.Lpig = 0;
        GlobalVar.Ipig = 0;
        GlobalVar.Rpig = 0;
        GlobalVar.Dpig = 0;
        GlobalVar.Shost = 0;
        GlobalVar.Vhost = 0;
        GlobalVar.Lhost = 0;
        GlobalVar.Ihost = 0;
        GlobalVar.Rhost = 0;
        GlobalVar.Dhost = 0;
    }
    */

    public static int rand_call_i(int maxvalue)                    //new function for throwing random int values
    {
        Random rand = new Random(Guid.NewGuid().GetHashCode());      //reuse this if generating many
random values
        return (rand.Next(0, maxvalue));                           //returns a random int value ranging (0 <= ? <
maxvalue)
    } //end of rand_call_i

    public static double rand_call_f(double maxvalue)              //new function for throwing random double
values
    {
        Random rand = new Random(Guid.NewGuid().GetHashCode());      //reuse this if generating many
random values
        return (rand.NextDouble() * maxvalue);                     //returns a random double value ranging (0 <= ?
< maxvalue)
    } //end of rand_call_f

    public static double rand_call_f_range(double low, double high)
    {
        double rand_scaled = 0.0;

        while (rand_scaled == 0.0)        //prevent from selecting 0.0
        {
            Random rand = new Random(Guid.NewGuid().GetHashCode());      //reuse this if generating
many random values
            rand_scaled = low + (rand.NextDouble() * (high - low));
        }

        return (rand_scaled);             //returns a random double value ranging (low < ? < high)
    } //end of rand_call_f_range

    public static double distance(double lat1, double lon1, double lat2, double lon2)    //kilometer
    {
        double theta = lon1 - lon2;
        double dist = Math.Sin(deg2rad(lat1)) * Math.Sin(deg2rad(lat2)) + Math.Cos(deg2rad(lat1)) *
Math.Cos(deg2rad(lat2)) * Math.Cos(deg2rad(theta));
```

```
            dist = Math.Acos(dist);
            dist = rad2deg(dist);
            dist = dist * 60 * 1.1515 * 1.609344;

            return (dist);
        }

        public static double deg2rad(double deg) { return (deg * Math.PI / 180.0); }

        public static double rad2deg(double rad) { return (rad / Math.PI * 180.0); }

    } //end of class GlobalVar
}
```

[Program.cs]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Threading;
using System.Threading.Tasks;


using Microsoft.SolverFoundation.Common;
using Microsoft.SolverFoundation.Solvers;
using Microsoft.SolverFoundation.Services;


namespace ConsoleApplication1
{
  class Program
  {
    /* Nelder-Mead parameter test */


    public static double[] xInitial;
    public static double[] xLower;
    public static double[] xUpper;
    public static StringBuilder main_output;
    public static StringBuilder main_output_all;


    //public static void Main()
    public static void Main(string[] args)
    {
      Console.WriteLine("   !!!   START OF MAIN   !!!   ");


      GlobalVar.initialize_globals_at_main();


      main_output = new StringBuilder();
      main_output_all = new StringBuilder();


      int option = 7;
      GlobalVar.repeats = 500;
      //GlobalVar.dailydyn_sw = true;
      GlobalVar.mode_sw = 0;


      if (option == 1)          //local_cow with ref
      {
        GlobalVar.in_farmlist_path = Directory.GetCurrentDirectory() + "/input_farmlist - cow 700.txt";


        GlobalVar.pnum = 2;
        GlobalVar.daylimit = 70;       //70 for local test (7000 pig farm & 700 cow farm), 20 for external
test (only Andong farms)
        GlobalVar.cull_sw = false;      //put this OFF during local test, ON during external test


        GlobalVar.localtest_sw = true;
        GlobalVar.useref_sw = true;
```

```
            GlobalVar.hosttype_sw = 1;       //1 for cow, 2 for pig (only works for local test)

            GlobalVar.externaltest_sw = false;
        }
        else if (option == 2)       //local_pig with ref
        {
            GlobalVar.in_farmlist_path = Directory.GetCurrentDirectory() + "/input_farmlist - pig 7000.txt";

            GlobalVar.pnum = 2;
            GlobalVar.daylimit = 70;        //70 for local test (7000 pig farm & 700 cow farm), 20 for external
test (only Andong farms)
            GlobalVar.cull_sw = false;      //put this OFF during local test, ON during external test

            GlobalVar.localtest_sw = true;
            GlobalVar.useref_sw = true;
            GlobalVar.hosttype_sw = 2;       //1 for cow, 2 for pig (only works for local test)

            GlobalVar.externaltest_sw = false;
        }
        else if (option == 3)       //local_cow without ref
        {
            GlobalVar.in_farmlist_path = Directory.GetCurrentDirectory() + "/input_farmlist - cow 700.txt";

            GlobalVar.pnum = 6;
            GlobalVar.daylimit = 70;        //70 for local test (7000 pig farm & 700 cow farm), 20 for external
test (only Andong farms)
            GlobalVar.cull_sw = false;      //put this OFF during local test, ON during external test

            GlobalVar.localtest_sw = true;
            GlobalVar.useref_sw = false;
            GlobalVar.hosttype_sw = 1;       //1 for cow, 2 for pig (only works for local test)

            GlobalVar.externaltest_sw = false;
        }
        else if (option == 4)       //local_pig without ref
        {
            GlobalVar.in_farmlist_path = Directory.GetCurrentDirectory() + "/input_farmlist - pig 7000.txt";

            GlobalVar.pnum = 6;
            GlobalVar.daylimit = 70;        //70 for local test (7000 pig farm & 700 cow farm), 20 for external
test (only Andong farms)
            GlobalVar.cull_sw = false;      //put this OFF during local test, ON during external test

            GlobalVar.localtest_sw = true;
            GlobalVar.useref_sw = false;
            GlobalVar.hosttype_sw = 2;       //1 for cow, 2 for pig (only works for local test)

            GlobalVar.externaltest_sw = false;
        }
        else if (option == 5)       //external for 2010 Andong
        {
```

```
            GlobalVar.in_farmlist_path = Directory.GetCurrentDirectory() + "/input_farmlist - 2010 Andong
(date shift).txt";


            GlobalVar.pnum = 6;
            GlobalVar.daylimit = 20;        //70 for local test (7000 pig farm & 700 cow farm), 20 for external
test (only Andong farms)
            GlobalVar.cull_sw = true;       //put this OFF during local test, ON during external test


            GlobalVar.localtest_sw = false;


            GlobalVar.externaltest_sw = true;


            GlobalVar.firstinf = 1;         //number of farm(s) infected on day 0
            //GlobalVar.realfarmnum = new int[20] { 1, 0, 1, 1, 0, 3, 12, 10, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0 };                //real data of daily infected farm number ... not used
            //GlobalVar.realfarmnum_acc = new int[20] { 1, 1, 2, 3, 3, 6, 18, 28, 30, 30, 30, 30, 30, 30, 30,
30, 30, 30, 30 };    //real data of daily infected farm number (accumulated)
            GlobalVar.realfarmnum = new int[10] { 1, 0, 1, 1, 0, 3, 12, 10, 2, 0 };                 //real data of
daily infected farm number ... not used
            GlobalVar.realfarmnum_acc = new int[10] { 1, 1, 2, 3, 3, 6, 18, 28, 30, 30 };    //real data of daily
infected farm number (accumulated)
        }
        else if (option == 6)       //external for 2016 Chungnam
        {
            GlobalVar.in_farmlist_path = Directory.GetCurrentDirectory() + "/input_farmlist - 2016 Korea only
chungnam.txt";


            GlobalVar.pnum = 6;
            GlobalVar.daylimit = 50;        //50 for external test (2016 Chungnam data)
            GlobalVar.cull_sw = true;       //put this OFF during local test, ON during external test


            GlobalVar.localtest_sw = false;


            GlobalVar.externaltest_sw = true;


            GlobalVar.firstinf = 2;         //number of farm(s) infected on day 0
            GlobalVar.realfarmnum = new int[50]
            {
                2, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                0, 0, 1, 5, 0, 0, 2, 0, 0, 1,
                0, 1, 0, 3, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0
            };
            GlobalVar.realfarmnum_acc = new int[50]
            {
                2, 2, 2, 2, 2, 2, 2, 3, 3, 3,
                3, 3, 3, 3, 3, 3, 3, 3, 3, 4,
                4, 4, 5, 10, 10, 10, 12, 12, 12, 13,
                13, 14, 14, 17, 17, 17, 18, 18, 18, 18,
                18, 18, 18, 18, 18, 18, 18, 18, 18, 18
            };
        }
```

```
        else if (option == 7)    //modified 2016 Chungnam (-3 initial farms)
        {
           GlobalVar.in_farmlist_path = Directory.GetCurrentDirectory() + "/input_farmlist - 2016 Korea only
chungnam (without 3 farms).txt";


           GlobalVar.pnum = 6;
           GlobalVar.daylimit = 30;         //50 for external test (2016 Chungnam data)
           GlobalVar.cull_sw = true;        //put this OFF during local test, ON during external test


           GlobalVar.localtest_sw = false;


           GlobalVar.externaltest_sw = true;


           GlobalVar.firstinf = 1;          //number of farm(s) infected on day 0
           /*
           GlobalVar.realfarmnum = new int[30]
           {
              1, 0, 0, 1, 5, 0, 0, 2, 0, 0,
              1, 0, 1, 0, 3, 0, 0, 1, 0, 0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0
           };
           GlobalVar.realfarmnum_acc = new int[30]
           {
              1, 1, 1, 2, 7, 7, 7, 9, 9, 9,
              10, 10, 11, 11, 14, 14, 14, 15, 15, 15,
              15, 15, 15, 15, 15, 15, 15, 15, 15, 15
           };
           */
           GlobalVar.realfarmnum = new int[20]
           {
              1, 0, 0, 1, 5, 0, 0, 2, 0, 0,
              1, 0, 1, 0, 3, 0, 0, 1, 0, 0
           };
           GlobalVar.realfarmnum_acc = new int[20]
           {
              1, 1, 1, 2, 7, 7, 7, 9, 9, 9,
              10, 10, 11, 11, 14, 14, 14, 15, 15, 15
           };
        }



        /* input file into stringbuilder */


        StringBuilder file_farms = new StringBuilder(File.ReadAllText(GlobalVar.in_farmlist_path));
        StringBuilder file_movements = new StringBuilder();
        StringBuilder file_vaccines = new StringBuilder();
        if (GlobalVar.move_sw)
{ file_movements.Append(File.ReadAllText(GlobalVar.in_movement_path)); }
        if (GlobalVar.vacc_sw) { file_vaccines.Append(File.ReadAllText(GlobalVar.in_vaccinelist_path)); }


        int key = 0;
        if (args.Length != 0)
```

```
        {
            key = Convert.ToInt32(args[0]);
            if (key < 1 || key > 8) { Console.WriteLine("ERROR : args[0] is incorrect (should be 1 ~ 8)");
Console.ReadLine(); }


            GlobalVar.out_path_dailydyn = Directory.GetCurrentDirectory() + "/result_dailydyn_" + args[0] +
".txt";
            GlobalVar.out_local_path = Directory.GetCurrentDirectory() + "/result_local_" + args[0] + ".txt";
            GlobalVar.out_external_path = Directory.GetCurrentDirectory() + "/result_external_" + args[0] +
".txt";
            GlobalVar.out_nmtest_path = Directory.GetCurrentDirectory() + "/result_nmtest_" + args[0] +
".txt";
        }


        /* stringbuilder for output file */


        StringBuilder output = new StringBuilder();          //output for every simulations
        StringBuilder output_all = new StringBuilder();      //output for every parameter sets


        if (GlobalVar.nmtest_sw)     //use Nelder-Mead test settings
        {
            xInitial = new double[GlobalVar.pnum];


            if (GlobalVar.localtest_sw)
            {
                if (GlobalVar.useref_sw)
                {
                    if (GlobalVar.hosttype_sw == 1)
                    {
                        //pnum should be 2
                        xLower = new double[2] { 0.10, 10.0 };
                        xUpper = new double[2] { 0.80, 50.0 };
                    }
                    else if (GlobalVar.hosttype_sw == 2)
                    {
                        //pnum should be 2
                        xLower = new double[2] { 0.10, 10.0 };
                        xUpper = new double[2] { 0.80, 50.0 };
                    }
                    else { Console.WriteLine("ERROR on hosttype_sw!!!"); Console.ReadLine(); }
                }
                else
                {
                    if (GlobalVar.hosttype_sw == 1)
                    {
                        //pnum should be 6
                        xLower = new double[6] { 0.01, 10.0, 3.0, 2.0, 4.0, 4.0 };
                        xUpper = new double[6] { 0.80, 50.0, 5.0, 10.0, 6.0, 6.0 };
                    }
                    else if (GlobalVar.hosttype_sw == 2)
                    {
                        //pnum should be 6
                        xLower = new double[6] { 0.01, 10.0, 5.0, 4.0, 4.0, 4.0 };
                        xUpper = new double[6] { 0.80, 50.0, 8.0, 14.0, 6.0, 6.0 };
                    }
```

```
                else { Console.WriteLine("ERROR on hosttype_sw!!!"); Console.ReadLine(); }
            }
        }
        if (GlobalVar.externaltest_sw)
        {
            if (GlobalVar.mode_sw == 0)
            {
                //pnum should be 6
                xLower = new double[6] { 0.05, 0.05, 0.05, 0.05, 0.1, 2.0 };    //no fitting for culling_day
                xUpper = new double[6] { 0.75, 0.75, 0.75, 0.75, 2.0, 2.0 };    //no fitting for culling_day
            }
            else if (GlobalVar.mode_sw == 2)
            {
                //pnum should be 6
                xLower = new double[6] { 0.05, 0.05, 0.05, 0.05, 0.1, 2.0 };    //no fitting for culling_day
                xUpper = new double[6] { 0.75, 0.75, 0.75, 0.75, 2.0, 2.0 };    //no fitting for culling_day
            }
        }

        var solver = new NelderMeadSolver();

        // Objective function
        int[] varids = new int[GlobalVar.pnum];
        int objid;
        solver.AddRow("obj", out objid);    //Adds a row to the model. (Object key, out int variable_index)
        solver.AddGoal(objid, 0, true);     //Marks a row as a goal. (int row_id, int goal_priority, bool
minimize_or_not)

        // Define variables
        if (GlobalVar.localtest_sw)
        {
            if (GlobalVar.useref_sw)
            {
                solver.AddVariable("infect_internal", out varids[0]);   //Ensures that a user variable with the
given key is in the model. (Object key, out int variable_index)
                solver.AddVariable("contact_rate", out varids[1]);
            }
            else
            {
                solver.AddVariable("infect_internal", out varids[0]);   //Ensures that a user variable with the
given key is in the model. (Object key, out int variable_index)
                solver.AddVariable("contact_rate", out varids[1]);
                solver.AddVariable("latent_period", out varids[2]);
                solver.AddVariable("duration", out varids[3]);
                solver.AddVariable("latent_k", out varids[4]);
                solver.AddVariable("duration_k", out varids[5]);
            }
        }
        if (GlobalVar.externaltest_sw)
        {
            solver.AddVariable("finf_cow", out varids[0]);   //Ensures that a user variable with the given
key is in the model. (Object key, out int variable_index)
            solver.AddVariable("finf_pig", out varids[1]);
            solver.AddVariable("fsus_cow", out varids[2]);
            solver.AddVariable("fsus_pig", out varids[3]);
            solver.AddVariable("A", out varids[4]);
```

```
            solver.AddVariable("culling_day", out varids[5]);
        }
        // Define bounds
        for (int i = 0; i < GlobalVar.pnum; i++) { solver.SetBounds(varids[i], xLower[i], xUpper[i]); }


        // Assign objective function delegate
        solver.FunctionEvaluator = NelderMeadTest;    //Gets or sets the value callback function that the solver invokes periodically in order to obtain function values for different variable values.


        // Solve
        var param = new NelderMeadSolverParams();
        param.IterationLimit = GlobalVar.nmLimit;    //maximum runs of NelderMead test per line
        param.Tolerance = GlobalVar.nmTol;           //Specifies the tolerance level. The solver terminates when the size of the simplex falls below the Tolerance.


        // Run Nelder-Mead test
        for (int i = 0; i < GlobalVar.pnum; i++)
        {
            xInitial[i] = (xLower[i] + xUpper[i]) * 0.5;
            solver.SetValue(varids[i], xInitial[i]);
        }


        var solution = solver.Solve(param);


        Console.WriteLine("   Iteration Count : {0} / {1}", solver.IterationCount, param.IterationLimit);
        Console.WriteLine("   Tolerance : {0}", param.Tolerance);
        Console.WriteLine("   Result : {0}", solution.Result);
        Console.WriteLine("   Minimium Objective Value : {0}", solution.GetValue(objid));
        for (int i = 0; i < GlobalVar.pnum; i++)
        {
            Console.WriteLine("   Parameter Values : {0}\t", solution.GetValue(varids[i]));
        }


        StreamWriter fpnm = File.AppendText(GlobalVar.out_nmtest_path);                  //file open (append)


        fpnm.WriteLine("   Iteration Count : {0} / {1}", solver.IterationCount, param.IterationLimit);
        fpnm.WriteLine("   Tolerance : {0}", param.Tolerance);
        fpnm.WriteLine("   Result : {0}", solution.Result);
        fpnm.WriteLine("   Minimium Objective Value : {0}", solution.GetValue(objid));
        for (int i = 0; i < GlobalVar.pnum; i++)
        {
            fpnm.WriteLine("   Parameter Values : {0}\t", solution.GetValue(varids[i]));
        }


        fpnm.Close();
    }
    else    //not using Nelder-Mead test
    {
        printoutfile(output_all, 0);    //write header at file


        param_make(file_farms.ToString(), file_movements.ToString(), file_vaccines.ToString(), key, option);   //make parameter and run model
```

```
            printoutfile(output_all, 1);     //write output at file
        }

        Console.WriteLine("   !!!   END OF MAIN   !!!   ");
    } //end of Main()

    public static void printoutfile(StringBuilder output_all, int mode)
    {
        /* stringbuilder to output file */

        //StreamWriter fp = new StreamWriter(GlobalVar.out_path);                    //file open (overwrite)
        //StreamWriter fpext = File.AppendText(GlobalVar.out_external_path);         //file open (append)

        StreamWriter fpout;

        if (mode == 0)       //start : write header info
        {
            if (GlobalVar.localtest_sw)      //print local test data
            {
                fpout = File.AppendText(GlobalVar.out_local_path);       //file open (append)

                if (GlobalVar.average_sw)    //header - average result of all repeats
                {
                    if (GlobalVar.useref_sw) { fpout.WriteLine("hosttype" + "\t" + "INF" + "\t" + "CR" + "\t" +
"avgerr" + "\t"); }
                    else { fpout.WriteLine("hosttype" + "\t" + "INF" + "\t" + "CR" + "\t" + "LP" + "\t" + "DUR" + "\t"
+ "LPk" + "\t" + "DURk" + "\t" + "avgerr" + "\t"); }
                }
                else                         //header - result of each repeats
                {
                    fpout.WriteLine("hosttype" + "\t" + "simno" + "\t" + "INF" + "\t" + "CR" + "\t"
                            + "d[0]" + "\t" + "d[1]" + "\t" + "d[2]" + "\t" + "d[3]" + "\t" + "d[4]" + "\t" + "d[5]" + "\t");
                }

                fpout.Close();       //file close (result_local.txt)
            }

            if (GlobalVar.externaltest_sw)   //print external test data
            {
                fpout = File.AppendText(GlobalVar.out_external_path);    //file open (append)

                if (GlobalVar.mode_sw == 0)
                {
                    fpout.Write("psetno" + "\t"
                            + "FINFc" + "\t" + "FINFp" + "\t" + "FSUSc" + "\t" + "FSUSp" + "\t" + "A" + "\t" +
"cullday" + "\t"
                            + "avgerr" + "\t" + "badsim1" + "\t" + "badsim2" + "\t" + "badsim" + "\t" + "totsim" +
"\t"
                            + "simno" + "\t" + "psimday" + "\t" + "errflag" + "\t");
                    for (int d = 0; d < GlobalVar.daylimit; d++)
                    {
```

```
                    fpout.Write("day" + d.ToString() + "\t");
                }
                fpout.WriteLine("err" + "\t");

                //format : (psetno) (p1~p6) (avgerr) (badsim1) (badsim2) (badsim) (totsim) / (simno)
(presimday) (errflag) (ACCUMULATED daily number of infected farms) (err)
            }
            else if (GlobalVar.mode_sw == 2)
            {
                fpout.Write("psetno" + "\t"
                        + "FINFc" + "\t" + "FINFp" + "\t" + "FSUSc" + "\t" + "FSUSp" + "\t" + "A" + "\t" +
"cullday" + "\t"
                        + "avgerr" + "\t" + "badsim1" + "\t" + "badsim2" + "\t" + "badsim" + "\t" + "totsim" +
"\t"
                        + "simno" + "\t" + "psimday" + "\t" + "errflag" + "\t");

                // (this header is written in FMD_IBM.cs)
                /*
                for (int d = 0; d < number_of_total_farms_in_input_file; d++)
                {
                    fpout.Write("farm" + d.ToString() + "\t");
                }
                fpout.WriteLine(" ");
                */

                //format : (psetno) (p1~p6) (avgerr) (badsim1) (badsim2) (badsim) (totsim) / (simno)
(presimday) (errflag) / (infection date of each farms)
            }

            fpout.Close();      //file close (result_external.txt)
        }
    }
    else if (mode == 1)     //finish : write output at file
    {
        if (GlobalVar.localtest_sw)     //print local test data
        {
            fpout = File.AppendText(GlobalVar.out_local_path);      //file open (append)

            if (GlobalVar.average_sw)   //header - average result of all repeats
            {
                fpout.WriteLine(main_output_all.ToString());
                fpout.WriteLine("*** END ***");
            }
            else                        //header - result of each repeats
            {
                fpout.WriteLine(main_output_all.ToString());
                fpout.WriteLine("*** END ***");
            }

            fpout.Close();      //file close (result_local.txt)
        }

        if (GlobalVar.externaltest_sw)  //print external test data
        {
```

```
            fpout = File.AppendText(GlobalVar.out_external_path);    //file open (append)

            if (GlobalVar.mode_sw == 0)
            {
               fpout.WriteLine(main_output_all.ToString());
               fpout.WriteLine("*** END ***");
            }
            else if (GlobalVar.mode_sw == 2)
            {
               fpout.WriteLine(main_output_all.ToString());
               fpout.WriteLine("*** END ***");
            }

            fpout.Close();       //file close (result_external.txt)
         }
      }
      else if (mode == 2)      //midcheck
      {
         if (GlobalVar.localtest_sw)      //print local test data
         {
            fpout = File.AppendText(GlobalVar.out_local_path);       //file open (append)

            if (GlobalVar.average_sw)    //header - average result of all repeats
            {
               fpout.WriteLine(main_output_all.ToString());
            }
            else                         //header - result of each repeats
            {
               fpout.WriteLine(main_output_all.ToString());
            }

            fpout.Close();       //file close (result_local.txt)
         }

         if (GlobalVar.externaltest_sw)   //print external test data
         {
            fpout = File.AppendText(GlobalVar.out_external_path);    //file open (append)

            if (GlobalVar.mode_sw == 0)
            {
               fpout.WriteLine(main_output_all.ToString());
            }
            else if (GlobalVar.mode_sw == 2)
            {
               fpout.WriteLine(main_output_all.ToString());
            }

            fpout.Close();       //file close (result_external.txt)
         }
      }
   } //end of printoutfile()
```

```
    public static void param_make(string file_farms_str, string file_movements_str, string file_vaccines_str,
int key, int option)
    {
        ParamSetDB pdb = new ParamSetDB();                  //new version of pdb (db after allocation)
        ParamSetList plist_ori = new ParamSetList();        //new version of plist (list before allocation)

        if (GlobalVar.localtest_sw)
        {
            if (GlobalVar.useref_sw)
            {
                int start = 0;
                int end = 80;
                if (key != 0)
                {
                    start = (key - 1) * 10;   //1,2,3,4,5,6,7,8 : 0,10,20,30,40,50,60,70
                    end = start + 10;         //1,2,3,4,5,6,7,8 : 10,20,30,40,50,60,70,80
                }

                for (int x0 = 0; x0 < 60; x0++)
                {
                    for (int x1 = start; x1 < end; x1++)
                    {
                        double[] pset = new double[GlobalVar.pnum];

                        pset[0] = 0.15 + (0.01 * x0);    //infect_internal = 0.15 ~ 0.75 (0.60 = 0.01 x
60)
                        pset[1] = 10.0 + (0.5 * x1);     //contact_rate = 10.0 ~ 50.0 (40.0 = 0.5 x 80)

                        plist_ori.Add(pset);
                    }
                }
            }
            else
            {
                if (GlobalVar.hosttype_sw == 1)
                {
                    double[] pset = new double[GlobalVar.pnum];

                    pset[0] = 0.13; //INF
                    pset[1] = 23.5; //CR
                    pset[2] = 3.4;   //LP
                    pset[3] = 8.65; //DUR
                    pset[4] = 5.9;   //LPk
                    pset[5] = 4.4;   //DURk

                    plist_ori.Add(pset);
                }
                else if (GlobalVar.hosttype_sw == 2)
                {
                    double[] pset = new double[GlobalVar.pnum];

                    pset[0] = 0.11; //INF
                    pset[1] = 27.0; //CR
                    pset[2] = 3.4;   //LP
                    pset[3] = 7.4;   //DUR
```

```
            pset[4] = 5.0;   //LPk
            pset[5] = 5.6;   //DURk

            plist_ori.Add(pset);
        }
    }
}

if (GlobalVar.externaltest_sw)
{
    int start = 0;
    int end = 16;
    if (key != 0)
    {
        start = (key - 1) * 2;       //1,2,3,4,5,6,7,8 : 0,2,4,6,8,10,12,14
        end = start + 2;             //1,2,3,4,5,6,7,8 : 2,4,6,8,10,12,14,16
    }

    for (int x0 = 0; x0 <= 0; x0++)
    {
        for (int x1 = 0; x1 <= 0; x1++)
        {
            for (int x2 = 0; x2 <= 0; x2++)
            {
                for (int x3 = 0; x3 <= 0; x3++)
                {
                    for (int x4 = 0; x4 <= 0; x4++)
                    {
                        for (int x5 = 0; x5 <= 0; x5++)
                        {
                            double[] pset = new double[GlobalVar.pnum];

                            //PARAM TESTING HERE

                            if (option == 5)         //andong - day 10
                            {
                                pset[0] = 0.56;
                                pset[1] = 0.16;
                                pset[2] = 0.13;
                                pset[3] = 0.85;
                            }
                            else if (option == 7)    //chungnam -3 - day 20
                            {
                                pset[0] = 0.064;
                                pset[1] = 0.007;
                                pset[2] = 0.023;
                                pset[3] = 0.064;
                            }

                            pset[4] = 9999;
                            pset[5] = 5.0;

                            //GlobalVar.finf_cow
                            //GlobalVar.finf_pig
```

```
                        //GlobalVar.fsus_cow
                        //GlobalVar.fsus_pig
                        //GlobalVar.A
                        //GlobalVar.cullday

                        plist_ori.Add(pset);
                    }
                }
            }
        }
    }
}

//making pdb
//   make (GlobalVar.threadnum) of plists
//   allocate (plist_ori.Count() / GlobalVar.threadnum) psets to each plist
//   leftovers which are smaller than (GlobalVar.threadnum) are added in the last plist
int div = plist_ori.Count() / GlobalVar.threadnum;
int j = 0;

for (int i = 0; i < GlobalVar.threadnum; i++)
{
    ParamSetList plist = new ParamSetList();        //new version of plist (members of pdb)

    for (j = i * div; j < (i + 1) * div; j++)
    {
        plist.Add(plist_ori[j]);
    }

    pdb.Add(plist);
}
for (int l = j; l < plist_ori.Count; l++)
{
    pdb[GlobalVar.threadnum - 1].Add(plist_ori[j]);
}

Console.WriteLine("finished making pdb (number of plist : {0})", pdb.Count());
for (int i = 0; i < pdb.Count(); i++) { Console.WriteLine("   plist No. {0} (number of pset : {1})", i,
pdb[i].Count()); }
Console.WriteLine("total pset : {0}", plist_ori.Count());
Console.WriteLine("repeat {0} iterations for each parameter set", GlobalVar.repeats);

/*
for (int i = 0; i < pdb.Count(); i++)
{
    for (int j = 0; j < pdb[i].Count(); j++)
    {
        for (int k = 0; k < GlobalVar.pnum; k++)
        {
            Console.Write("{0} ", pdb[i][j][k]);
        }
        Console.WriteLine();
    }
```

```
        }
        */


        if (GlobalVar.dailydyn_sw)
        {
            param_run_one(file_farms_str, file_movements_str, file_vaccines_str, key, pdb[pdb.Count() -
1][0]);   //do not use parallel when testing with only one parameter set to check daily dynamics
        }
        else
        {
            param_run(file_farms_str, file_movements_str, file_vaccines_str, key, pdb);
        }


        main_output = new StringBuilder();
    } //end of param_make()


    public static void param_run(string file_farms_str, string file_movements_str, string file_vaccines_str,
int key, List<ParamSetList> pdb)
    {
        int psetno = 0;          //dummy value
        int threadnum = GlobalVar.threadnum;


        StringBuilder total_output = new StringBuilder();
        StringBuilder total_output_all = new StringBuilder();
        StringBuilder[] output = new StringBuilder[threadnum];
        StringBuilder[] output_all = new StringBuilder[threadnum];
        string[] para_result = new string[threadnum];

        for (int c = 0; c < threadnum; c++)
        {
            output[c] = new StringBuilder();
            output_all[c] = new StringBuilder();
            para_result[c] = "";
        }


        Console.WriteLine("This computer has {0} cores", Environment.ProcessorCount);
        Console.WriteLine("Decided to use maximum {0} threads", threadnum);


        Parallel.For(0, threadnum, new ParallelOptions() { MaxDegreeOfParallelism = threadnum }, core
=>    //use all cores for parallelism with Parallel.For();
            {
                //Thread for tasks are automatically allocated
                //During task, thread number can be checked by Thread.CurrentThread.ManagedThreadId
                //To check core number of the machine, use Environment.ProcessorCount


                Console.WriteLine("   --- core {0}/{1} : Thread {2} started param_run() ---", core, threadnum,
Thread.CurrentThread.ManagedThreadId);


                FMD_IBM fmd_ibm = new FMD_IBM();


                fmd_ibm.loadInput(file_farms_str, file_movements_str, file_vaccines_str);
```

```
            for (int i = 0; i < pdb[core].Count(); i++)
            {
                //Console.WriteLine("   Thread {0} : start {1} loop", Thread.CurrentThread.ManagedThreadId,
i);

                para_result[core] = fmd_ibm.model_run(pdb[core][i], psetno);   //run model

                output[core].Append(para_result[core].Split('@')[0]);
                output_all[core].Append(para_result[core].Split('@')[1]);
                //para_output_all format for local test
                //   if error : (error message with errflag/repeats)
                //   if not error : (hosttype) (params) (avgerr)
                //para_output_all format for external test
                //   common : (DUMMY psetno) (params) (avgerr) (badsim1) (badsim2) (badsim) (totsim)
(simno) (psimday) (errflag)
                //   if error (errflag = 1 or 2) : nothing after errflag
                //   if not error (errflag = 0) : (accumulated number of daily infected farms for 20 days) (err)

                if (i % 10 == 0)
                {
                    Console.WriteLine("   ... core {0} : loop {1}", core, i);
                }
            }

            Console.WriteLine("   --- core {0} finished param_run() ---", core);
        }
    );

    for (int c = 0; c < threadnum; c++)
    {
        main_output.Append(output[c].ToString());
        main_output_all.Append(output_all[c].ToString());
    }
} //end of param_run()

public static void param_run_one(string file_farms_str, string file_movements_str, string
file_vaccines_str, int key, double[] pset)
{
    int psetno = 0;         //dummy value

    StringBuilder total_output = new StringBuilder();
    StringBuilder total_output_all = new StringBuilder();

    Console.WriteLine("Daily Dynamics Test Output");
    Console.Write("using pset : ");
    for (int i = 0; i < GlobalVar.pnum; i++)
    {
        Console.Write("{0} ", pset[i]);
    }
    Console.WriteLine(" ");

    FMD_IBM fmd_ibm = new FMD_IBM();
```

```
        string para_result;

        fmd_ibm.loadInput(file_farms_str, file_movements_str, file_vaccines_str);

        para_result = fmd_ibm.model_run(pset, psetno);   //run model

        total_output.Append(para_result.Split('@')[0]);
        total_output_all.Append(para_result.Split('@')[1]);
        //para_output_all format for local test
        //   if error : (error message with errflag/repeats)
        //   if not error : (hosttype) (params) (avgerr)
        //para_output_all format for external test
        //   common : (DUMMY psetno) (params) (avgerr) (badsim1) (badsim2) (badsim) (totsim) (simno)
(psimday) (errflag)
        //   if error (errflag = 1 or 2) : nothing after errflag
        //   if not error (errflag = 0) : (accumulated number of daily infected farms for 20 days) (err)

        Console.WriteLine("   !!! finished dailydyn test");
    } //end of param_run_one()

    /*
================================================================================
============================================== */

    private static double NelderMeadTest(INonlinearModel model, int rowVid, ValuesByIndex values, bool
newValues)
    {
        /*
        You must set the value callback function before you try to solve the model.
        The value callback function has the following arguments:
        The first argument is the INonlinearModel that is being solved.
        The second argument is a Int32 that represents the row index to be evaluated, which is a goal or a
constraint.
        The third argument is a ValuesByIndex that contains the current variable values that are accessible
by index.
        The fourth argument is a Boolean that indicates whether the current call is the first evaluator call
with the current variable values.
        The return value is a Double that is the value of the row for the current variable values.
        */

        double obj = 0;
        double[] pset = new double[GlobalVar.pnum];
        string outputstr = "";
        string output_allstr = "";

        StringBuilder file_farms = new StringBuilder(File.ReadAllText(GlobalVar.in_farmlist_path));
        StringBuilder file_movements = new StringBuilder();
        StringBuilder file_vaccines = new StringBuilder();

        if (GlobalVar.localtest_sw)
        {
            if (GlobalVar.useref_sw)
            {
```

```
            pset[0] = values[model.GetIndexFromKey("infect_internal")];
            pset[1] = values[model.GetIndexFromKey("contact_rate")];
        }
        else
        {
            pset[0] = values[model.GetIndexFromKey("infect_internal")];
            pset[1] = values[model.GetIndexFromKey("contact_rate")];
            pset[2] = values[model.GetIndexFromKey("latent_period")];
            pset[3] = values[model.GetIndexFromKey("duration")];
            pset[4] = values[model.GetIndexFromKey("latent_k")];
            pset[5] = values[model.GetIndexFromKey("duration_k")];
        }
    }
    if (GlobalVar.externaltest_sw)
    {
        pset[0] = values[model.GetIndexFromKey("finf_cow")];
        pset[1] = values[model.GetIndexFromKey("finf_pig")];
        pset[2] = values[model.GetIndexFromKey("fsus_cow")];
        pset[3] = values[model.GetIndexFromKey("fsus_pig")];
        pset[4] = values[model.GetIndexFromKey("A")];
        pset[5] = values[model.GetIndexFromKey("culling_day")];
    }

    GlobalVar.psetno++;


    FMD_IBM fmd_ibm = new FMD_IBM();

    fmd_ibm.loadInput(file_farms.ToString(), file_movements.ToString(), file_vaccines.ToString());

    obj = fmd_ibm.model_run_neldermead(pset, outputstr, output_allstr, GlobalVar.psetno);   //run
model

    if (GlobalVar.psetno % 100 == 0)   //print checking point every 100 loops
    {
        StreamWriter fpnm = File.AppendText(GlobalVar.out_nmtest_path);                    //file open
(append)

        Console.WriteLine("model_run() loop {0} ... result : {1}", GlobalVar.psetno, obj);
        Console.Write("   pset :\t");
        fpnm.WriteLine("model_run() loop {0} ... result : {1}", GlobalVar.psetno, obj);
        fpnm.Write("   pset :\t");
        for (int i = 0; i < GlobalVar.pnum; i++)
        {
            Console.Write("{0}\t", pset[i]);
            fpnm.Write("{0}\t", pset[i]);
        }
        Console.WriteLine(" ");
        fpnm.WriteLine(" ");


        fpnm.Close();
    }
```

```
            return obj;
        }
    } //end of class Program

    public class ParamSetDB : List<ParamSetList> { }

    public class ParamSetList : List<double[]> { }
}
```

[FMD_IBM.cs]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

using Microsoft.SolverFoundation.Common;
using Microsoft.SolverFoundation.Solvers;
using Accord.Statistics.Distributions.Univariate;

namespace ConsoleApplication1
{
  public class FMD_IBM : IDisposable
  {
    /* define variables */

    public bool _disposed;

    int[] hostpop;            //number of S, V, L, I, R, D (all hosts)
    int[] cowpop;             //number of S, V, L, I, R, D (cows)
    int[] pigpop;             //number of S, V, L, I, R, D (pigs)

    int[] trackers;
      //0 : uhid
      //1 : nFarm
      //2 : Lfarms
      //3 : Ifarms
      //4 : cownum_init
      //5 : pignum_init
      //6 : hostnum_init
      //7 : cownum
      //8 : pignum
      //9 : hostnum

    farm[] farms;
    traj[] movements;
    vaccinedata[] vaccines;

    public int[] realfarmnum_acc;      //real data of daily ACCUMULATED infected farm number

    public WeibullDistribution wei;    //cumulative probability distribution for phase shift (L -> I) of cow
    public GammaDistribution gamma1;   //cumulative probability distribution for phase shift (L -> I) of pig
    public GammaDistribution gamma2;   //cumulative probability distribution for phase shift (I -> R or D) of cow
    public LogLogisticDistribution log; //cumulative probability distribution for phase shift (I -> R or D) of pig

    public string out_path_dailydyn;   //output file (result) path for recording daily dynamics ... NOT USED
```

```
    public int daylimit;              //simulation day limit (day)
    public bool move_sw;              //switch variable whether to use movement data ... THIS OPTION IS
NOT SUPPORTED
    public bool vacc_sw;              //switch variable whether to apply vaccination
    public bool cull_sw;              //switch variable whether to apply culling ... put this ON during external
test, OFF during local test


    public bool localtest_sw;         //switch variable whether to test local infection
    public bool dailydyn_sw;          //switch variable whether to print out daily dynamics data ... only
works with 1 parameter set
    public bool average_sw;           //switch variable whether to print out average results of all
simulation ... only works with localtest
    public bool useref_sw;            //switch variable whether to use reference data for latent period and
duration ... pnum should be 2 if this is ON, 5 if this is OFF
    public int hosttype_sw;           //switch variable determining the type of host in local test (1 : cow / 2 :
pig)


    public bool externaltest_sw;      //switch variable whether to test external infection
    public int mode_sw;               //switch variable of mode
                                      //0 : print out number of infected farms
                                      //1 : print out difference of infection date of each farms ... only works when
dailydyn_sw = FALSE (?)
                                      //2 : print out infection date of each farms


    public int vaccine_limit_cow;     //duration of vaccine effect to cow (day)
    public int vaccine_limit_pig;     //duration of vaccine effect to pig (day)


    public double infect_internal_cow;  //internal infect rate between cow to cow (%) ..... becomes
p_local value of cow farms
    public double infect_internal_pig;  //internal infect rate between pig to pig (%) ..... becomes p_local
value of pig farms
    public double contact_rate_cow;     //parameter for contact rate in a cow farm
    public double contact_rate_pig;     //parameter for contact rate in a pig farm
    public double latent_period_cow;    //average latent period of cow (day) ..... using incubation period
from review note
    public double latent_period_pig;    //average latent period of pig (day) ..... using incubation period
from review note
    public double duration_cow;         //average duration until cow dies or recovers after infection
(day) ..... not important if culling option is on
    public double duration_pig;         //average duration until pig dies or recovers after infection (day) .....
not important if culling option is on
    public double chance_recover_cow;   //recover rate of cow (%)
    public double chance_recover_pig;   //recover rate of pig (%)


    public double latent_k_cow;         //parameter for L->I of cow (steepness of sigmoid curve)
    public double latent_k_pig;         //parameter for L->I of pig (steepness of sigmoid curve)
    public double duration_k_cow;       //parameter for I->RorD of cow (steepness of sigmoid curve)
    public double duration_k_pig;       //parameter for I->RorD of pig (steepness of sigmoid curve)
    public double vacc_k_cow;           //parameter for V->S of cow (steepness of sigmoid curve)
    public double vacc_k_pig;           //parameter for V->S of pig (steepness of sigmoid curve)


    public double finf_cow;             //external infectibility of cow farm (%)
    public double finf_pig;             //external infectibility of pig farm (%)
```

```
    public double fsus_cow;              //external susceptibility of cow farm (%)
    public double fsus_pig;              //external susceptibility of pig farm (%)

    public double A;                     //parameter for external infection
    public double culling_day;           //average days when infected farms get culled after first occurence of
L host (day) ..... this is important instead of duration

    public int[] local_realdata;         //local infection graph emergence data (0, 1, 2 : cow / 3, 4, 5 : pig)
    public int repeats;                  //simulation repeating number

    public int firstinf;                 //number of farm(s) infected on day 0

    public FMD_IBM()
    {
       /* initialize variables */

       hostpop = new int[6];
       cowpop = new int[6];
       pigpop = new int[6];
       trackers = new int[10];

       realfarmnum_acc = new int[GlobalVar.daylimit];
       if (GlobalVar.externaltest_sw)
       {
          for (int i = 0; i < GlobalVar.daylimit; i++)
          {
             realfarmnum_acc[i] = GlobalVar.realfarmnum_acc[i];
          }
       }

       wei = GlobalVar.wei;              //cow - latent
       gamma1 = GlobalVar.gamma1;        //pig - latent
       gamma2 = GlobalVar.gamma2;        //cow - infectious
       log = GlobalVar.log;              //pig - infectious

       out_path_dailydyn = GlobalVar.out_path_dailydyn;

       daylimit = GlobalVar.daylimit;
       move_sw = GlobalVar.move_sw;
       vacc_sw = GlobalVar.vacc_sw;
       cull_sw = GlobalVar.cull_sw;

       localtest_sw = GlobalVar.localtest_sw;
       dailydyn_sw = GlobalVar.dailydyn_sw;
       average_sw = GlobalVar.average_sw;
       hosttype_sw = GlobalVar.hosttype_sw;

       externaltest_sw = GlobalVar.externaltest_sw;
       mode_sw = GlobalVar.mode_sw;

       vaccine_limit_cow = GlobalVar.vaccine_limit_cow;
```

```
    vaccine_limit_pig = GlobalVar.vaccine_limit_pig;

    infect_internal_cow = GlobalVar.infect_internal_cow;
    contact_rate_cow = GlobalVar.contact_rate_cow;
    latent_period_cow = GlobalVar.latent_period_cow;
    duration_cow = GlobalVar.duration_cow;
    latent_k_cow = GlobalVar.latent_k_cow;
    duration_k_cow = GlobalVar.duration_k_cow;

    infect_internal_pig = GlobalVar.infect_internal_pig;
    contact_rate_pig = GlobalVar.contact_rate_pig;
    latent_period_pig = GlobalVar.latent_period_pig;
    duration_pig = GlobalVar.duration_pig;
    latent_k_pig = GlobalVar.latent_k_pig;
    duration_k_pig = GlobalVar.duration_k_pig;

    vacc_k_cow = GlobalVar.vacc_k_cow;
    vacc_k_pig = GlobalVar.vacc_k_pig;

    chance_recover_cow = GlobalVar.chance_recover_cow;
    chance_recover_pig = GlobalVar.chance_recover_pig;

    finf_cow = GlobalVar.finf_cow;
    finf_pig = GlobalVar.finf_pig;
    fsus_cow = GlobalVar.fsus_cow;
    fsus_pig = GlobalVar.fsus_pig;

    A = GlobalVar.A;
    culling_day = GlobalVar.culling_day;

    local_realdata = new int[6];
    for (int i = 0; i < 6; i++)
    {
        local_realdata[i] = GlobalVar.local_realdata[i];
    }
    repeats = GlobalVar.repeats;

    firstinf = GlobalVar.firstinf;
}

~FMD_IBM()
{
    Dispose(false);
}

public void Dispose()
{
    Dispose(true);
    // any other managed resource cleanups you can do here
    GC.SuppressFinalize(this);
}
```

```
    protected virtual void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            _disposed = true;
        }
    }

    public void loadInput(string file_farms_str, string file_movements_str, string
file_vaccines_str)       //load input file data into farm, traj, vaccinedata
    {
        trackers[7] = 0;    //cownum
        trackers[8] = 0;    //pignum
        trackers[9] = 0;    //hostnum
        trackers[2] = 0;    //Lfarms
        trackers[3] = 0;    //Ifarms

        /* construct data from string */

        //!!! WARNING !!! ... error occurs if input files have any additional blank lines at the end due to
(string).Length method

        string[] templine_farms = file_farms_str.Split('\n');                    //seperate buffer by lines
        string[] templine_movements = file_movements_str.Split('\n');            //seperate buffer by lines
        string[] templine_vaccines = file_vaccines_str.Split('\n');              //seperate buffer by lines

        trackers[1] = templine_farms.Length - 1;                                 //initialize number of farms here
(subtract 1 line becasue it is header)

        farms = new farm[trackers[1]];                                           //subtract 1 line becasue it is header
        movements = new traj[templine_movements.Length - 1];                     //subtract 1 line becasue it
is header
        vaccines = new vaccinedata[templine_vaccines.Length - 1];                //subtract 1 line becasue it
is header

        for (int lnum = 0; lnum < trackers[1]; lnum++)                           //skip first line (header)
        {
            farms[lnum] = new farm();

            string[] tempstr = templine_farms[lnum + 1].Split('\t');             //input file seperator is TAB

            farms[lnum].setFarm(tempstr, localtest_sw, externaltest_sw,
                        finf_cow, fsus_cow, infect_internal_cow, contact_rate_cow,
                        finf_pig, fsus_pig, infect_internal_pig, contact_rate_pig);      //create farms
        }

        if (move_sw)
        {
            for (int lnum = 0; lnum < templine_movements.Length - 1; lnum++)     //skip first line (header)
            {
                movements[lnum] = new traj();
```

```
            string[] tempstr = templine_movements[lnum + 1].Split('\t');        //input file seperator is TAB

            movements[lnum].date = int.Parse(tempstr[0]);
            movements[lnum].start_farmid = int.Parse(tempstr[1]);
            movements[lnum].goal_farmid = int.Parse(tempstr[2]);
        }
    }

    if (vacc_sw)
    {
        for (int lnum = 0; lnum < templine_vaccines.Length - 1; lnum++)         //skip first line (header)
        {
            vaccines[lnum] = new vaccinedata();

            string[] tempstr = templine_vaccines[lnum + 1].Split('\t');         //input file seperator is TAB

            vaccines[lnum].farmid = int.Parse(tempstr[0]);
            vaccines[lnum].date = int.Parse(tempstr[1]);
        }
    }
}

public void renew()         //used to reset FMD_IBM instance for next simulation
{
    trackers[7] = 0;    //cownum
    trackers[8] = 0;    //pignum
    trackers[9] = 0;    //hostnum
    trackers[2] = 0;    //Lfarms
    trackers[3] = 0;    //Ifarms

    for (int i = 0; i < 6; i++)
    {
        hostpop[i] = 0;
        cowpop[i] = 0;
        pigpop[i] = 0;
    }
}

public string CORE(int iter, double[] pset, int[,] local_d, double[] extsave, StringBuilder[] dailydyn_res,
int firstinf)
{
    //Console.WriteLine("   ===   START OF CORE {0} / {1}   ===   ", iter, repeats);

    string tempres = "DEFAULT\n";
    int presimdays = 0;

    int[] d = new int[6];
    for (int a = 0; a < 6; a++)
    {
        d[a] = -9999;
    }
```

```
        extsave[0] = 0;
        extsave[1] = 0;


        //checking farm and host (only 1st~10th of each farms) list
        /*
        Console.WriteLine("checking farmdata : {0} lines", linenum);
        for (int i = 0; i < 10; i++)
        {
           Console.WriteLine("   farm {0} : {1} {2} {3} {4}", farms[i].farmid, farms[i].locx, farms[i].locy,
farms[i].hosttype, farms[i].nHost);
           for (int j = 0; j < 3; j++)
           {
              Console.WriteLine("     host {0} in farm {1} : {2} {3}", farms[i].hostlist.ElementAt(j).hid,
farms[i].hostlist.ElementAt(j).farmid, farms[i].hostlist.ElementAt(j).type, farms[i].hostlist.ElementAt(j).status);
           }
        }
        */


        /* setup(Initialize) part */


        for (int i = 0; i < trackers[1]; i++)
        {
           farms[i].init(trackers, pset, localtest_sw, externaltest_sw, useref_sw,
                 latent_period_cow,  duration_cow,  chance_recover_cow,  vaccine_limit_cow,
latent_k_cow, duration_k_cow, vacc_k_cow,
                 latent_period_pig,  duration_pig,  chance_recover_pig,  vaccine_limit_pig,  latent_k_pig,
duration_k_pig, vacc_k_pig);
        }


        if (externaltest_sw && presimdays == -9999)
        {
           /*
           Console.WriteLine("   simulation {0} / {1} is invalid", iter, repeats);
           */
           extsave[0] = 1;


           return (iter + "\t" + presimdays + "\t" + 1 + "\t");    //BADSIM ERROR ... return empty line
        }


        if (localtest_sw)
        {
           for (int i = 0; i < 6; i++)
           {
              if (d[i] == -9999)   //if any of the event is invalid (emergence is not within simulation period)
              {
                 return ("ERROR : INVALID EVENT");
              }
              local_d[iter - 1, i] = d[i];
           }


           if (!average_sw)
           {
              tempres = save_results_local(iter, d, pset);
           }
```

```
        }

        if (externaltest_sw)
        {
            if (mode_sw == 0)
            {
                tempres = save_results_new(iter, presimdays, extsave);     //save_results_new() ... save simulation results into a string after a simulation ended at final day
            }
            else if (mode_sw == 1)
            {
                //tempres = save_results(iter, presimdays, extsave);        //save_results() ... save simulation results into a string after a simulation ended at final day
            }
            else if (mode_sw == 2)
            {
                tempres = save_results_infdate(iter, presimdays, extsave);
            }
        }

        //Console.WriteLine(" ===  END OF CORE {0} / {1} : {2} presimdays  === ", iter, repeats, presimdays);

        return (tempres);
    } //end of CORE()

    public void move()      //infected farm[i] transmit disease to nearby susceptible farm[j]
    {
        for (int i = 0; i < farms.Length; i++)
        {
            if (farms[i].Idate >= 0)     //infected farm[i]
            {
                for (int j = 0; j < farms.Length; j++)
                {
                    farms[j].totalinf = 1.0;     //initialize totalinf value here and calculate at setExternalInfection()

                    if (dist < 0.01) { dist = 0.01; }     //minimum dist between farms set to 10m

                    if (i != j && farms[i].hostlist.Count() != 0)   //do not calculate from culled farms
                    //if (i != j && farms[i].hostlist.Count() != 0 && farms[j].Ldate < 0)   //use this if already infected farms are not targeted for external infection
                    {
                        int infectious_hosts = 0;
                        double p = 0;
                        double farminfp = 0;

                        foreach (host each in farms[i].hostlist)
                        {
                            if (each.status == 1 || each.status == 2) { infectious_hosts++; }    //count L+A+I in farm[i]
                            //if (each.status == 2) { infectious_hosts++; }    //count A+I in farm[i]
                        }
```

```
                    //p = farms[i].p_local * Math.Exp(-1 * A * dist);                    //infectibility of one infected
host in farm[i]
                    //farminfp = (1 - Math.Pow(1 - p, infectious_hosts)) * farms[i].finf;    //infectibility of farm[i]



                    //farminfp = p * farms[i].finf;                                      //using simple method
                    if (farminfp > 1.0) { farminfp = 1.0; }


                    farms[j].setExternalInfection(farminfp);
                    //[!!!]
                    //Console.WriteLine("move : farm{0} -> farm{1} ... {2}", farms[i].farmid, farms[j].farmid,
farminfp);
                    //Console.WriteLine("   {0}/{1}, {2}km : {3}, {4}", infectious_hosts, farms[i].hostlist.Count(),
dist, p, farms[i].finf);


                    if (Double.IsNaN(farminfp) || Double.IsInfinity(farminfp))
                    {
                        Console.WriteLine("ERROR! some farminfp values are invalid!");
                        Console.WriteLine("   p = {0}, farminfp = {1}", p, farminfp);
                    }
                }


                // [!]
                //Console.WriteLine("move : farmALL -> farm{0} / {1}", farms[j].farmid, farms[j].totalinf);
            }
        }
    }
} //end of move()

public void give_vaccine(int day)
{
    for (int i = 0; i < vaccines.Length; i++)
    {
        if (day == vaccines[i].date)    //only act on vaccinating day
        {
            bool check_vaccinate = false;

            for (int j = 0; j < farms.Length; j++)
            {
                if (vaccines[i].farmid == farms[j].farmid)
                {
                    foreach (host each in farms[j].hostlist)
                    {
                        each.vaccinate();
                    }
                    check_vaccinate = true;
                    break;
                }
            }

            if (!check_vaccinate)
            {
                Console.WriteLine("ERROR : give_vaccine() ... wrong farmid {0}", vaccines[i].farmid);
```

```
                Console.WriteLine("   use only farmid in the farmlist input file");
                Console.ReadLine();
            }
        }
    }
} //end of give_vaccine()

public void culling(int day)
{
    foreach (farm each in farms)
    {
        if (each.Idate > 0 && day >= each.Idate + culling_day)    //cull the infected farm after specific days have passed since first occurence of L host
        {
            each.culling();
        }
    }
}

public void update(int day, int[] d, int[] trackers, int iter, StringBuilder[] dailydyn_res)
{
    //cownum = 0;
    //pignum = 0;
    //hostnum = 0;
    //Lfarms = 0;
    //Ifarms = 0;

    for (int i = 0; i < 6; i++)
    {
        cowpop[i] = 0;
        pigpop[i] = 0;
        hostpop[i] = 0;
    }

    trackers[2] = 0;
    trackers[3] = 0;

    //manually check total population
    for (int i = 0; i < trackers[1]; i++)
    {
        //count hosts in each farm
        farms[i].updateStatus();

        {
            Console.WriteLine("ERROR : update() ... wrong host type {0}", farms[i].hosttype);
            Console.WriteLine("   use only 1(cow) or 2(pig) for host type");
            Console.ReadLine();
        }

        //count farms with L or I
        if (farms[i].Ldate > 0) { trackers[2]++; }   //Lfarms
        if (farms[i].Idate > 0) { trackers[3]++; }   //Iframs
    }
```

```
        trackers[9] = trackers[7] + trackers[8];     //hostnum = cownum + pignum



        if (day == 0)    //only works at first update call
        {
           trackers[4] = trackers[7];              //cownum_init = cownum;
           trackers[5] = trackers[8];              //pignum_init = pignum;
           trackers[6] = trackers[9];              //hostnum_init = hostnum;
        }


        cowpop[5] = trackers[4] - trackers[7];       //D cows = cownum_init - cownum
        pigpop[5] = trackers[5] - trackers[8];       //D pigs = pignum_init - pignum




        /*
        Console.WriteLine(" ");
        Console.WriteLine("(in update) Day : {0} / {1}", day, daylimit);
        Console.WriteLine("infected farms : {0} / {1}", infected_farms, farms.Length);
        Console.WriteLine("type\tT\tS\tV\tL\tI\tR\tD");
        //Console.WriteLine("COWS\t{0}\t{1}\t{2}\t{3}\t{4}\t{5}", cownum, cowpop[0], cowpop[1], cowpop[2],
cowpop[3], cowpop[4], cowpop[5]);
        //Console.WriteLine("PIGS\t{0}\t{1}\t{2}\t{3}\t{4}\t{5}", pignum, pigpop[0], pigpop[1], pigpop[2],
pigpop[3], pigpop[4], pigpop[5]);
        Console.WriteLine("HOSTS\t{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}", hostnum, hostpop[0], hostpop[1],
hostpop[2], hostpop[3], hostpop[4], hostpop[5]);
        Console.WriteLine(" ");
        //Console.ReadLine();
        */


        //Console.WriteLine("(in update) Day : {0} / {1}", day, daylimit);




        //save event occurence data
        if (d[0] == -9999 && hostpop[3] > trackers[6] * 0.01) { d[0] = day; }                  //record d[0] (I > 1%
of hostnum_init)
        if (d[1] == -9999 && (hostpop[4] + hostpop[5]) > trackers[6] * 0.01) { d[1] = day; }      //record d[1]
(R+D > 1% of hostnum_init)
        if (d[2] == -9999 && (hostpop[0] + hostpop[2]) < trackers[6] * 0.01) { d[2] = day; }      //record d[2]
(S+L < 1% of hostnum_init)
        if (d[0] != -9999 && d[3] == -9999 && hostpop[3] < trackers[6] * 0.01) { d[3] = day; }    //record d[3] (I
< 1% of hostnum_init)


        if (d[5] < hostpop[3])
        {
           d[4] = day;                                                    //record d[4] (day of maxI)
           d[5] = hostpop[3];                                               //record d[5] (maxI)
        }
```

```
        //dailydyn printout
        if (dailydyn_sw)
        {
            if (day == 0)    //print header at first call of update()
            {
                dailydyn_res[iter - 1].AppendLine("day\tT\tS\tV\tL\tI\tR\tD\toutbreak\n");       //write header on
first day
            }

            dailydyn_res[iter - 1].Append(day + "\t" + trackers[9] + "\t" + hostpop[0] + "\t" + hostpop[1] + "\t" +
hostpop[2] + "\t" + hostpop[3] + "\t" + hostpop[4] + "\t" + hostpop[5] + "\t");


            /*
            foreach (farm each in farms)
            {
                if (each.Ldate == day)
                {
                    dailydyn_res[iter - 1] += each.farmid;
                    dailydyn_res[iter - 1] += "\t";
                }
            } //if any, print out farmid that have L at current date
            */


            dailydyn_res[iter - 1].AppendLine(" ");
        }
    } //end of update()


//      public string save_results(int iter, int presimdays, double[] extsave)      //save infection date of
farms (subtract presimdays from raw results)
//      {
//          string tempres = iter + "\t" + presimdays + "\t" + 0 + "\t";
//          int sdiff = 0;


//          for (int fcount = 0; fcount < farms.Length; fcount++)
//          {
            /* [EXCEPTION] error flag 2 disabled */
//              if (farms[fcount].Idate - presimdays < 0)   //if there are any uninfected farms at last day of
simulation
//              {
                /*
                Console.WriteLine("BADSIM : save_results() ... uninfected farm {0} at final day : {1}",
farms[fcount].farmid, farms[fcount].Idate);
                Console.WriteLine("   all farms should be have I hosts at simulation day {0}", daylimit);

                Console.WriteLine("   simulation {0} / {1} is invalid", iter, repeats);
                */
                //extsave[0] = 2;


                //return (iter + "\t" + presimdays + "\t" + 2 + "\t");    //return empty string due to BADSIM ERROR
//              }


            /******* IMPORTANT : (farms[].Idate - presimdays) is the actual Idate! *******/
```

```
            //tempres += (farms[fcount].Idate - presimdays) + "\t";                    //choose this to print sim
(simulated infection date)
//              tempres += (farms[fcount].Idate - presimdays - farms[fcount].day_inf) + "\t";       //choose this
to print diff (sim - obs)


            //[!]
            //Console.WriteLine("save_results(), {0} = {1}", presimdays, farms[0].Idate);


//              sdiff  +=  Math.Abs(farms[fcount].Idate  -  presimdays  -  farms[fcount].day_inf);        //calculate
total diff
//          }


//          extsave[1] = sdiff / (double)farms.Length;                                  //simavgdiff
//          tempres += extsave[1] + "\t";                                             //add avg_diff to result too


//          return (tempres);    //string format is (simno) (presimday) (errflag) (sim of diff of farms) (avg_diff)
//      } //end of save_results()


    public string save_results_new(int iter, int presimdays, double[] extsave)       //save daily number of
infected farms (subtract presimdays from raw results)
    {
        string tempres_acc = iter + "\t" + presimdays + "\t" + 0 + "\t";


        int[] Ifarmnum = new int[daylimit];
        int farm_acc = 0;


        int simerr = 0;


        for (int i = 0; i < daylimit; i++) { Ifarmnum[i] = 0; }


        for (int fcount = 0; fcount < farms.Length; fcount++)
        {
            /* [EXCEPTION] error flag 2 disabled */
            if (farms[fcount].Idate - presimdays < 0)   //if there are any uninfected farms at last day of
simulation
            {
                /*
                Console.WriteLine("BADSIM : save_results() ... uninfected farm {0} at final day : {1}",
farms[fcount].farmid, farms[fcount].Idate);
                Console.WriteLine("   all farms should be have I hosts at simulation day {0}", daylimit);


                Console.WriteLine("   simulation {0} / {1} is invalid", iter, repeats);
                */
                //extsave[0] = 2;


                //return (iter + "\t" + presimdays + "\t" + 2 + "\t");    //return empty string due to BADSIM ERROR
            }
            else
            {
                /******* IMPORTANT : (farms[].Idate - presimdays) is the actual Idate! *******/
```

```
            //Console.WriteLine("presimdays value is? {0} {1}", farms[fcount].Idate, presimdays);

            Ifarmnum[farms[fcount].Idate - presimdays]++;
          }
        }

        extsave[1] = simerr / (double)daylimit;                    //simavgerr - sum of error is averaged by dividing into 20 days
        tempres_acc += extsave[1] + "\t";

        return (tempres_acc);  //string format is (simno) (presimday) (errflag) (ACCUMULATED daily number of infected farms) (simavgerr)
    } //end of save_results_new()

    public string save_results_infdate(int iter, int presimdays, double[] extsave)       //save infected dates of farms (subtract presimdays from raw results)
    {
        string tempres_acc = iter + "\t" + presimdays + "\t" + 0 + "\t";

        int[] Idate_of_farms = new int[farms.Count()];



        return (tempres_acc);   //string format is (simno) (presimday) (errflag) (infected date of each farms)
    } //end of save_results_infdate()

    public string save_results_local(int iter, int[] d, double[] pset)       //save dates of event occurence
    {
        //printout format
        //   type : COW or PIG
        //   simulation number : iter
        //   parameters : infect_internal, latent_period, duration, contact_rate, latent_k, duration_k
        //   results (event dates) : I > 1%, R > 1%, S < 1%
        //   optional results : I < 1%, d(maxI), maxI

        string tempres;

        if (useref_sw) { tempres = farms[0].hosttype + "\t" + iter + "\t" + pset[0] + "\t" + pset[1] + "\t"; }
        else { tempres = farms[0].hosttype + "\t" + iter + "\t" + pset[0] + "\t" + pset[1] + "\t" + pset[2] + "\t" + pset[3] + "\t" + pset[4] + "\t" + pset[5] + "\t"; }

        tempres += d[0] + "\t" + d[1] + "\t" + d[2] + "\t" + d[3] + "\t" + d[4] + "\t" + d[5] + "\t";    //do not subtract presimdays

        return (tempres);
    } //end of save_results_local

    /*
============================================================================== */

    public string model_run(double[] pset, int psetno)
    {
```

```
        double nmgoal = 0;
        StringBuilder output = new StringBuilder();
        StringBuilder output_all = new StringBuilder();
        int[,] local_d = new int[repeats, 6];                    //array for saving local test results
        double[] extsave = new double[2];                        //array for saving external test error flag and simavgerr
        StringBuilder[] dailydyn_res = new StringBuilder[repeats];        //array of strings for saving daily dynamics simulation results
        for (int i = 0; i < repeats; i++) { dailydyn_res[i] = new StringBuilder(); }


        /* variable used in local test */
        int errflag = -9999;     //used to check whether event date contains -9999 or not


        /* variable used in external test */
        double avgerr_ext = 0;  //variable that contains the total err of all simulations for a specific parameter set
        int badsim = 0;          //number of invalid simulations
        int badsim1 = 0;         //number of invalid simulations (badsim1 : other farm is infected before first farm) ..... not used in data printout since every data contains its own error flag
        int badsim2 = 0;         //number of invalid simulations (badsim2 : there are uninfected farm at final day) ..... not used in data printout   since every data contains its own error flag
        string[] temps = new string[repeats];            //temporary string to save results


        if (externaltest_sw)
        {
            A = pset[4];
            culling_day = pset[5];
        }


        for (int iter = 1; iter <= repeats; iter++)
        {
            //Console.WriteLine("PART AAA : repeat no. {0}", iter);


            string results;


            renew();


            //output format
            //   internal test : (hosttype) (simno) (parameters) (results)
            //   external test (mode_sw = TRUE) : (psetno) (p1~p6) (avgerr) (badsim1) (badsim2) (badsim) (totsim) / (simno) (presimday) (errflag) (ACCUMULATED daily number of infected farms) (simavgerr)
            //   external test (mode_sw = FALSE) : (presimday) (sim or diff of farms) (avg_diff)


            //check invalid simulation for localtest
            if (localtest_sw)
            {
                if (results == "ERROR : INVALID EVENT")     //this simulation is invalid due to lack of emergence detection in the graph
                {
                    errflag = iter;
                    break;
                }
                output.Append(results);      //used if average_sw is off
```

```
            }

            //check invalid simulation for external_test
            if (externaltest_sw)
            {
               if (extsave[0] == 0) { avgerr_ext += extsave[1]; }
               else if (extsave[0] == 1) { badsim1++; }
               else if (extsave[0] == 2) { badsim2++; }
               else { Console.WriteLine("ERROR : EXT_ERRFLAG (extsave[0]) IS INVALID"); }


               if (mode_sw == 0)
               {
                  temps[iter - 1] = results;       //use temps[] instead of output
               }
               else if (mode_sw == 1)
               {
                  output.AppendLine(results);       //(simno)  (presimday)  (errflag)  (ACCUMULATED  daily
number of infected farms)
               }
               else if (mode_sw == 2)
               {
                  temps[iter - 1] = results;       //use temps[] instead of output
               }
            }
         }

         badsim = badsim1 + badsim2;
         avgerr_ext = avgerr_ext / (double)(repeats - badsim);

         /* after all repeats ended, save results in stringbuilder */

         if (localtest_sw)      //print local test data
         {
            if (average_sw)    //print average result of all repeats
            {
               if (useref_sw) { output_all.Append(hosttype_sw + "\t" + pset[0] + "\t" + pset[1] + "\t"); }
               else { output_all.Append(hosttype_sw + "\t" + pset[0] + "\t" + pset[1] + "\t" + pset[2] + "\t" +
pset[3] + "\t" + pset[4] + "\t" + pset[5] + "\t"); }

               if (errflag == -9999)     //if there was no error, add avgerr to print list
               {
                  //calculate average of results
                  double err = 0;
                  double toterr = 0;
                  double avgerr = 0;


                  double ra = 0;
                  double rb = 0;
                  double rc = 0;
                  if (hosttype_sw == 1)         //cow realdata
                  {
                     ra = local_realdata[0];
                     rb = local_realdata[1];
                     rc = local_realdata[2];
```

```
                }
                else if (hosttype_sw == 2)    //pig realdata
                {
                    ra = local_realdata[3];
                    rb = local_realdata[4];
                    rc = local_realdata[5];
                }
                else
                {
                    Console.WriteLine("ERROR on hosttype_sw!!!");
                    Console.ReadLine();
                }

                output_all.AppendLine(avgerr + "\t");

                nmgoal = avgerr;
            }
            else     //if there was error, do not calculate avgerr and just put error message with iteration
            {
                output_all.AppendLine("ERROR at iteration " + errflag + " of " + repeats + "\t");

                nmgoal = 9999;
            }
        }
        else                        //print result of each repeats
        {
            output_all = output;        //write file
        }
        //output_all format for localtest
        //   if error : (error message with errflag/repeats)
        //   if not error : (hosttype) (params) (avgerr)

        if (dailydyn_sw)
        {
            for (int i = 0; i < repeats; i++)
            {
                StreamWriter fpdaily = new StreamWriter(out_path_dailydyn + i + ".txt");     //file overwrite
                //StreamWriter fpdaily = File.AppendText(out_path_dailydyn + i + ".txt");      //file append

                fpdaily.WriteLine(dailydyn_res[i]);

                fpdaily.Close();
            }
        }
    }

    if (externaltest_sw)       //print external test data
    {
        if (mode_sw == 0)
        {
            //final printout format for external test : (psetno) (p1~p6) (avgerr) (badsim1) (badsim2) (badsim)
(totsim) / (simno) (presimday) (errflag) (ACCUMULATED daily number of infected farms) (err)
```

```
            for (int i = 0; i < repeats; i++)
            {
                output_all.Append(psetno + "\t");                                   //(psetno) ... this is DUMMY if NelderMead test is off
                output_all.Append(pset[0] + "\t" + pset[1] + "\t" + pset[2] + "\t" + pset[3] + "\t" + pset[4] + "\t" + pset[5] + "\t");     //(p1~p6)
                output_all.Append(avgerr_ext + "\t");                               //(avgerr) ... average of all errs
                output_all.Append(badsim1 + "\t" + badsim2 + "\t" + badsim + "\t");                 //(badsim1) (badsim2) (badsim)
                output_all.Append(repeats + "\t");                                  //(totsim)

                output_all.AppendLine(temps[i]);    //(simno) (presimday) (errflag) (ACCUMULATED daily number of infected farms) (err)
            }

            nmgoal = avgerr_ext;
        }
        else if (mode_sw == 2)
        {
            //final printout format for external test : (psetno) (p1~p6) (avgerr) (badsim1) (badsim2) (badsim) (totsim) / (simno) (presimday) (errflag) (infected date of each farms)

            // !!! print header for mode_sw == 2
            double[] dist_of_farms = new double[farms.Count()];    //distance between farm no. 1 and each farms

            for (int i = 0; i < farms.Count(); i++)
            {
                dist_of_farms[i] = distance(farms[i].locx, farms[i].locy, farms[0].locx, farms[0].locy);
                if (dist_of_farms[i] < 0.01) { dist_of_farms[i] = 0.01; }     //minimum dist between farms set to 10m
                output_all.Append(dist_of_farms[i] + "\t");

                //Console.WriteLine("{0} : {1}", i, dist_of_farms[i]);
                //Console.WriteLine("    {0}, {1} to {2}, {3}", farms[i].locx, farms[i].locy, farms[0].locx, farms[0].locy);
            }

            output_all.AppendLine(" ");

            for (int i = 0; i < repeats; i++)
            {
                output_all.Append(psetno + "\t");                                   //(psetno) ... this is DUMMY if NelderMead test is off
                output_all.Append(pset[0] + "\t" + pset[1] + "\t" + pset[2] + "\t" + pset[3] + "\t" + pset[4] + "\t" + pset[5] + "\t");     //(p1~p6)
                output_all.Append(avgerr_ext + "\t");                               //(avgerr) ... average of all errs
                output_all.Append(badsim1 + "\t" + badsim2 + "\t" + badsim + "\t");                 //(badsim1) (badsim2) (badsim)
```

```
                output_all.Append(repeats                                                    +
"\t");                                                    //(totsim)

                output_all.AppendLine(temps[i]);    //(simno) (presimday) (errflag) (infected date of each
farms)
            }

            nmgoal = avgerr_ext;
        }

        /*
        //file write farm infection date (simulation results)
        if (mode_sw) { output_all.AppendLine("sim No." + "\t" + "Psimday" + "\t" + "daily number of
infected farms ... "); }    //header for mode_sw = TRUE
        else { output_all.AppendLine("sim No." + "\t" + "Psimday" + "\t" + "diff of farms ...
avg_diff"); }                    //header for mode_sw = FALSE
        output_all = output;

        //file write parameters of internal infection
        output_all.AppendLine("P_int" + "\t" + "LPc" + "\t" + "LPp" + "\t" + "DURc" + "\t" + "DURp" + "\t" +
"infc" + "\t" + "infp" + "\t");
        output_all.AppendLine("\t" + (GlobalVar.latent_period_cow) + "\t" + (GlobalVar.latent_period_pig)
+ "\t"
                        + (GlobalVar.duration_cow) + "\t" + (GlobalVar.duration_pig) + "\t"
                        + (GlobalVar.infect_internal_cow) + "\t" + (GlobalVar.infect_internal_pig) + "\t");

        //file write parameters of external infection
        output_all.AppendLine("P_ext" + "\t" + "A" + "\t" + "cullday" + "\t" + "finfc" + "\t" + "finfp" + "\t" +
"fsusc" + "\t" + "fsusp" + "\t");
        output_all.AppendLine("\t" + (GlobalVar.A) + "\t" + (GlobalVar.culling_day) + "\t"
                        + (GlobalVar.finf_cow) + "\t" + (GlobalVar.finf_pig) + "\t"
                        + (GlobalVar.fsus_cow) + "\t" + (GlobalVar.fsus_pig) + "\t");

        //file write simulation validity check
        output_all.AppendLine(" ");
        output_all.AppendLine("badsim" + "\t" + badsim + "\t" + "out of" + "\t" + (repeats) + "\t");
        */
    }

    return (output.ToString() + "@" + output_all.ToString());
} //end of model_run()

public double model_run_neldermead(double[] pset, string outputstr, string output_allstr, int psetno)
{
    double nmgoal = 0;                          //goal result for Nelder-Mead test
    StringBuilder output = new StringBuilder();
    StringBuilder output_all = new StringBuilder();
    int[,] local_d = new int[repeats, 6];           //array for saving local test results
    double[] extsave = new double[2];               //array for saving external test error flag and
simavgerr
    StringBuilder[] dailydyn_res = new StringBuilder[repeats];      //array of strings for saving daily
dynamics simulation results
    for (int i = 0; i < repeats; i++) { dailydyn_res[i] = new StringBuilder(); }
```

```
        int firstinf = 1;                       //number of farm(s) that are infected on day 0

        /* variable used in local test */
        int errflag = -9999;    //used to check whether event date contains -9999 or not

        /* variable used in external test */
        double avgerr_ext = 0;  //variable that contains the total err of all simulations for a specific parameter set
        int badsim = 0;         //number of invalid simulations
        int badsim1 = 0;        //number of invalid simulations (badsim1 : other farm is infected before first farm) ..... not used in data printout since every data contains its own error flag
        int badsim2 = 0;        //number of invalid simulations (badsim2 : there are uninfected farm at final day) ..... not used in data printout   since every data contains its own error flag
        string[] temps = new string[repeats];         //temporary string to save results

        if (externaltest_sw)
        {
            //asdf
            Console.WriteLine("in FMD_IBM.cs, model_run_neldermead()");
            Console.WriteLine("A (before) : {0}", A);

            A = pset[4];
            culling_day = pset[5];

            //asdf
            Console.WriteLine("A (after) : {0}", A);
        }

        for (int iter = 1; iter <= repeats; iter++)
        {
            string results;

            renew();

            //output format
            //   internal test : (hosttype) (simno) (parameters) (results)
            //   external test (mode_sw = TRUE) : (psetno) (p1~p6) (avgerr) (badsim1) (badsim2) (badsim) (totsim) / (simno) (presimday) (errflag) (ACCUMULATED daily number of infected farms) (simavgerr)
            //   external test (mode_sw = FALSE) : (presimday) (sim or diff of farms) (avg_diff)

            //check invalid simulation for localtest
            if (localtest_sw)
            {
                if (results == "ERROR : INVALID EVENT")     //this simulation is invalid due to lack of emergence detection in the graph
                {
                    errflag = iter;
                    break;
                }
                output.Append(results);      //used if average_sw is off
            }
```

```
            //check invalid simulation for external_test
            if (externaltest_sw)
            {
                if (extsave[0] == 0) { avgerr_ext += extsave[1]; }
                else if (extsave[0] == 1) { badsim1++; }
                else if (extsave[0] == 2) { badsim2++; }
                else { Console.WriteLine("ERROR : EXT_ERRFLAG (extsave[0]) IS INVALID"); }
            }
        }

        badsim = badsim1 + badsim2;
        avgerr_ext = avgerr_ext / (double)(repeats - badsim);

        /* after all repeats ended, save results in stringbuilder */

        if (localtest_sw)      //print local test data
        {
            if (average_sw)    //print average result of all repeats
            {
                if (errflag == -9999)     //if there was no error, add avgerr to print list
                {
                    //calculate average of results
                    double err = 0;
                    double toterr = 0;
                    double avgerr = 0;

                    double ra = 0;
                    double rb = 0;
                    double rc = 0;
                    if (hosttype_sw == 1)         //cow realdata
                    {
                        ra = local_realdata[0];
                        rb = local_realdata[1];
                        rc = local_realdata[2];
                    }
                    else if (hosttype_sw == 2)    //pig realdata
                    {
                        ra = local_realdata[3];
                        rb = local_realdata[4];
                        rc = local_realdata[5];
                    }
                    else
                    {
                        Console.WriteLine("ERROR on hosttype_sw!!!");
                        Console.ReadLine();
                    }

                    for (int a = 0; a < repeats; a++)
                    {
                        err = Math.Abs(local_d[a, 0] - ra) + Math.Abs(local_d[a, 1] - rb) + Math.Abs(local_d[a, 2] -
rc);
                        toterr += err;
                    }
                    avgerr = toterr / (double)repeats;
```

```
                nmgoal = avgerr;
            }
            else    //if there was error, do not calculate avgerr and just put error message with iteration
            {
                nmgoal = 9999;
            }
        }
    }

    return nmgoal;
} //end of model_run_neldermead()

public static int rand_call_i(int maxvalue)                 //new function for throwing random int values
{
    Random rand = new Random(Guid.NewGuid().GetHashCode());     //reuse this if generating many random values
    return (rand.Next(0, maxvalue));                        //returns a random int value ranging (0 <= ? < maxvalue)
} //end of rand_call_i

public static double rand_call_f(double maxvalue)           //new function for throwing random double values
{
    Random rand = new Random(Guid.NewGuid().GetHashCode());     //reuse this if generating many random values
    return (rand.NextDouble() * maxvalue);                  //returns a random double value ranging (0 <= ? < maxvalue)
} //end of rand_call_f

public static double rand_call_f_range(double low, double high)
{
    double rand_scaled = 0.0;

    while (rand_scaled == 0.0)      //prevent from selecting 0.0
    {
        Random rand = new Random(Guid.NewGuid().GetHashCode());     //reuse this if generating many random values
        rand_scaled = low + (rand.NextDouble() * (high - low));
    }

    return (rand_scaled);           //returns a random double value ranging (low < ? < high)
} //end of rand_call_f_range

public static double distance(double lat1, double lon1, double lat2, double lon2)
{
    double theta = lon1 - lon2;
    double dist = Math.Sin(deg2rad(lat1)) * Math.Sin(deg2rad(lat2)) + Math.Cos(deg2rad(lat1)) * Math.Cos(deg2rad(lat2)) * Math.Cos(deg2rad(theta));
    dist = Math.Acos(dist);
    dist = rad2deg(dist);
    dist = dist * 60 * 1.1515 * 1.609344;
```

```
        return (dist);
      }

      public static double deg2rad(double deg) { return (deg * Math.PI / 180.0); }

      public static double rad2deg(double rad) { return (rad / Math.PI * 180.0); }
   } //end of class FMD_IBM
}
```

[farm.cs]

```
using System;
using System.Collections.Generic;
using System.Linq;

using Accord.Statistics.Distributions.Univariate;

namespace ConsoleApplication1
{
  public class farm : IDisposable
  {
    //farm properties (get from input)
    public int farmid;
    public double locx;
    public double locy;
    public double alt;
    public int hosttype;
    public int nHost;
    public double finf;          //farm infectibility (to be used in external infection)
    public double fsus;          //farm susceptibility (to be used in external infection)
    public double totalinf;      //1 - sum of farm infectibility (to be used in external infection)
    public double contact_rate; //contact rate of the farm

    public int Ldate;           //date when first latent host occured, -1 at default (simulated value) ... subtract presimdays to get true Ldate
    public int Idate;           //date when first infected host occured, -1 at default (simulated value) ... subtract presimdays to get true Idate
    public int day_inf;         //date when first infected host actually occured (from input, observed value)

    public double p_local;      //possibility of local infection (%)

    public LinkedList<host> hostlist;

    //farm statistics (used when printing out detailed farm population info)
    public int Tn;
    public int Sn;
    public int Vn;
    public int Ln;
    public int In;
    public int Rn;
    public int Dn;

    //used to delete dead hosts in recover()
    Queue<host> deadhost = new Queue<host>();

    public bool _disposed;    //used for disposing the class ..... this function cannot be used now

    public farm()            //for object
    {
      hostlist = new LinkedList<host>();
```

```
    }

    public void setFarm(string[] tempdata, bool localtest_sw, bool externaltest_sw,
                double finf_cow, double fsus_cow, double infect_internal_cow, double contact_rate_cow,
                double finf_pig, double fsus_pig, double infect_internal_pig, double
contact_rate_pig)          //initialize farm
    {
        //some other farm variables were already set in farminit

        this.farmid = int.Parse(tempdata[0]);
        this.locx = double.Parse(tempdata[1]);
        this.locy = double.Parse(tempdata[2]);
        this.alt = double.Parse(tempdata[3]);
        this.hosttype = int.Parse(tempdata[4]);
        this.nHost = int.Parse(tempdata[5]);
        this.day_inf = int.Parse(tempdata[6]);

        if (this.hosttype == 1)
        {
            this.p_local = infect_internal_cow;
            this.contact_rate = contact_rate_cow;

            this.finf = finf_cow;
            this.fsus = fsus_cow;
        }
        else if (this.hosttype == 2)
        {
            this.p_local = infect_internal_pig;
            this.contact_rate = contact_rate_pig;

            this.finf = finf_pig;
            this.fsus = fsus_pig;
        }

        this.totalinf = 1.0;

        this.Ldate = -1;
        this.Idate = -1;
    } //end of setFarm()

    public void init(int[] trackers, double[] pset, bool localtest_sw, bool externaltest_sw, bool useref_sw,
                double latent_period_cow, double duration_cow, double chance_recover_cow, int
vaccine_limit_cow, double latent_k_cow, double duration_k_cow, double vacc_k_cow,
                double latent_period_pig, double duration_pig, double chance_recover_pig, int
vaccine_limit_pig, double latent_k_pig, double duration_k_pig, double vacc_k_pig)
    {
        if (localtest_sw)
        {
            //get value from pset[] regardless of useref_sw or farm type
            this.p_local = pset[0];
            this.contact_rate = pset[1];
        }
```

```
        if (externaltest_sw)
        {
            if (this.hosttype == 1)
            {
                this.finf = pset[0];     //pset[0] : finf_cow
                this.fsus = pset[2];     //pset[2] : fsus_cow
            }
            else if (this.hosttype == 2)
            {
                this.finf = pset[1];     //pset[1] : finf_pig
                this.fsus = pset[3];     //pset[3] : fsus_pig
            }
        }

        this.totalinf = 1.0;

        this.Ldate = -1;
        this.Idate = -1;

        hostlist = new LinkedList<host>();
        create_host(trackers, pset, localtest_sw, externaltest_sw, useref_sw,
                latent_period_cow, duration_cow, chance_recover_cow, vaccine_limit_cow, latent_k_cow,
duration_k_cow, vacc_k_cow,
                latent_period_pig, duration_pig, chance_recover_pig, vaccine_limit_pig, latent_k_pig,
duration_k_pig, vacc_k_pig);
    } //end of init()

    public void create_host(int[] trackers, double[] pset, bool localtest_sw, bool externaltest_sw, bool
useref_sw,
                    double latent_period_cow, double duration_cow, double chance_recover_cow, int
vaccine_limit_cow, double latent_k_cow, double duration_k_cow, double vacc_k_cow,
                    double latent_period_pig, double duration_pig, double chance_recover_pig, int
vaccine_limit_pig, double latent_k_pig, double duration_k_pig, double vacc_k_pig)
    {
        for (int id = trackers[0]; id < trackers[0] + nHost; id++)     //uhid
        {
            host ind = new host(id);

            ind.farmid = farmid;
            ind.setType(this.hosttype, pset, localtest_sw, externaltest_sw, useref_sw,
                    latent_period_cow, duration_cow, chance_recover_cow, vaccine_limit_cow, latent_k_cow,
duration_k_cow, vacc_k_cow,
                    latent_period_pig, duration_pig, chance_recover_pig, vaccine_limit_pig, latent_k_pig,
duration_k_pig, vacc_k_pig);
            ind.init();    //initialize hosts after creation

            hostlist.AddLast(ind);       //add initialized host to hostlist
        }
        trackers[0] += nHost;        //uhid

        if (hosttype == 1)
        {
            trackers[7] += nHost;    //cownum
        }
```

```
        else if (hosttype == 2)
        {
            trackers[8] += nHost;    //pignum
        }
        else
        {
            Console.WriteLine("ERROR : create_hosts() ... wrong host type {0}", hosttype);
            Console.WriteLine("   use only 1(cow) or 2(pig) for host type");
            Console.ReadLine();
        }
        trackers[9] = trackers[7] + trackers[8];     //hostnum = cownum + pignum
    } //end of create_host()

    public void get_older()
    {
        foreach (host each in hostlist)
        {
            each.get_older();
        }
    } //end of get_older()

    public void phase_shift(bool useref_sw, bool vacc_sw, WeibullDistribution wei, GammaDistribution gamma1, GammaDistribution gamma2, LogLogisticDistribution log)
    {
        foreach (host each in hostlist)
        {
            each.phase_shift(useref_sw, vacc_sw, wei, gamma1, gamma2, log);

            if (each.status == 4)           //if the host is stage D
            {
                deadhost.Enqueue(each);
                //hostlist.Remove(each);     //can't use this inside foreach loop, use Queue instead
            }
        }

        killhost(deadhost, hostlist);
    } //end of phase_shift()

    public void setExternalInfection(double farminfp)
    {
        this.totalinf *= (1.0 - farminfp);   //therefore, totalinf = 1 - infection_prob = 1 - (1 - farminfp_A)(1 - farminfp_B)(1 - farminfp_C) ...
        if (this.totalinf > 1.0) { this.totalinf = 1.0; }
        if (this.totalinf < 0.0) { this.totalinf = 0.0; }
        // [!]
        //Console.WriteLine("farm.cs, setExternalInfection() : {0} {1}", this.totalinf, farminfp);
    } //end of setExternalInfection()

    public void getExternalInfection()
    {
        foreach (host each in hostlist)
        {
            if (each.status == 0)
            {
                // [!]
```

```
            //Console.WriteLine("farm.cs, getExternalInfection() : {0} {1}", totalinf, fsus);
            if (rand_call_f(1.00) < (1.0 - totalinf) * fsus)
            {
               each.infect_host();
            }
         }
      }
   } //end of getExternalInfection()

   public void getLocalInfection(int day, bool localtest_sw, bool externaltest_sw)      //if some cow or pig in a farm is sick, other cow or pig in that farm may be also infected, ALSO CHECK LDATE AND IDATE
   {
      int Lhosts = 0;
      int Ihosts = 0;
      int infectious_hosts = 0;

      foreach (host each in hostlist)
      {
         if (each.status == 1) { Lhosts++; }
         if (each.status == 2) { Ihosts++; }
      }

      //Ldate and Idate are determined and saved here
      if (Lhosts > 0 && Ldate < 0)
      {
         Ldate = day;
      }
      if (Ihosts > 0 && Idate < 0)
      {
         Idate = day;
      }

      if (Lhosts + Ihosts > 0)
      {
         foreach (host each in hostlist)
         {
            if (each.status == 0)
            {
               if (localtest_sw) { infectious_hosts = Ihosts; }    //use logic from UNIST model when comparing the test result with UNIST model
               if (externaltest_sw) { infectious_hosts = Lhosts + Ihosts; }

            }
         }
      }
   } //end of getLocalInfection()

   public void culling()
   {
      foreach (host each in hostlist)
      {
         deadhost.Enqueue(each); //kill every hosts in that farm
```

```
    }

    killhost(deadhost, hostlist);
} //end of culling()

public void killhost(Queue<host> deadhost, LinkedList<host> hostlist)
{
    while (deadhost.Count > 0)
    {
        host deadind = deadhost.Dequeue();
        hostlist.Remove(deadind);
    }
} //end of killhost()

public void updateStatus()       //count population of host pools in a farm
{
    this.Tn = 0;
    this.Sn = 0;
    this.Vn = 0;
    this.Ln = 0;
    this.In = 0;
    this.Rn = 0;
    //this.Dn = 0;

    foreach (host each in hostlist)
    {
        this.Tn++;

        if (each.status == 0) { this.Sn++; }
        else if (each.status == -1) { this.Vn++; }
        else if (each.status == 1) { this.Ln++; }
        else if (each.status == 2) { this.In++; }
        else if (each.status == 3) { this.Rn++; }
    }
}

public int getT() { return (this.Tn); }
public int getS() { return (this.Sn); }
public int getV() { return (this.Vn); }
public int getL() { return (this.Ln); }
public int getI() { return (this.In); }
public int getR() { return (this.Rn); }

/* class disposal */

~farm()
{
    Dispose(false);
}

public void Dispose()
{
    Dispose(true);
```

```
        //any other managed resource cleanups you can do here
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            //farmlist.Clear();   //CANNOT USE DISPOSE FUNCTION BECAUSE LIST ITSELF CANNOT DISPOSE ITSELF, AND MAIN CANNOT HAVE DISPOSAL OPTION
            _disposed = true;
        }
    }

    public static int rand_call_i(int maxvalue)                 //new function for throwing random int values
    {
        Random rand = new Random(Guid.NewGuid().GetHashCode());     //reuse this if generating many random values
        return (rand.Next(0, maxvalue));                        //returns a random int value ranging (0 <= ? < maxvalue)
    } //end of rand_call_i

    public static double rand_call_f(double maxvalue)               //new function for throwing random double values
    {
        Random rand = new Random(Guid.NewGuid().GetHashCode());     //reuse this if generating many random values
        return (rand.NextDouble() * maxvalue);                  //returns a random double value ranging (0 <= ? < maxvalue)
    } //end of rand_call_f

    public static double rand_call_f_range(double low, double high)
    {
        double rand_scaled = 0.0;

        return (rand_scaled);           //returns a random double value ranging (low < ? < high)
    } //end of rand_call_f_range
  } //end of class farm

  public class traj          //used for getting and storing trajectory info from input file
  {
    public int date;
    public int start_farmid;
    public int goal_farmid;
  }
  public class vaccinedata    //used for getting and storing vaccine info from input file
  {
    public int farmid;
    public int date;
  }
}
```

[host.cs]

```
using System;

using Accord.Statistics.Distributions.Univariate;

namespace ConsoleApplication1
{
  public class host
  {
    public int farmid;              //farm id, also points to certain location (n)
    public int type;               //host type (0 : unidentified, 1 : cow, 2 : pig)

    public int status;             //host status (-1, 0, 1, 2, 3, 4 : V, S, L, I, R, D ... D is removed immediately)

    public int latent_count;        //days after latent (day)
    public int sick_count;          //days after infection (day)
    public int vaccine_count;       //days after vaccinated (day)

    public double latent_period;    //average latent period of host (day)
    public double duration;         //average duration of host illness (day)
    public double recover_rate;     //recovery rate of host illness (%)

    public double latent_k;         //parameter for L->I
    public double duration_k;       //parameter for I->RorD
    public double vacc_k;           //parameter for V->S

    public int vaccine_days;        //limit of vaccine effect on host (day)

    public int hid;                //id of hosts

    public host(int hostid)         //give id to host
    {
      this.hid = hostid;
    }

    public void setType(int hosttype, double[] pset, bool localtest_sw, bool externaltest_sw, bool useref_sw,
                        double latent_period_cow, double duration_cow, double chance_recover_cow, int vaccine_limit_cow, double latent_k_cow, double duration_k_cow, double vacc_k_cow,
                        double latent_period_pig, double duration_pig, double chance_recover_pig, int vaccine_limit_pig, double latent_k_pig, double duration_k_pig, double vacc_k_pig)  //initialize host type with parameter values
    {
      this.type = hosttype;

      if (localtest_sw)
      {
        if (useref_sw)
        {
          if (this.type == 1)
```

```
    {
        this.latent_period = latent_period_cow;
        this.duration = duration_cow;
        this.recover_rate = chance_recover_cow;
        this.vaccine_days = vaccine_limit_cow;


        this.latent_k = latent_k_cow;
        this.duration_k = duration_k_cow;
        this.vacc_k = vacc_k_cow;
    }
    else if (this.type == 2)
    {
        this.latent_period = latent_period_pig;
        this.duration = duration_pig;
        this.recover_rate = chance_recover_pig;
        this.vaccine_days = vaccine_limit_pig;


        this.latent_k = latent_k_pig;
        this.duration_k = duration_k_pig;
        this.vacc_k = vacc_k_pig;
    }
    else
    {
        Console.WriteLine("ERROR : setType() ... wrong host type {0}", this.type);
        Console.WriteLine("   use only 1(cow) or 2(pig) for host type");
        Console.ReadLine();
    }
}
else
{
    if (this.type == 1)
    {
        this.latent_period = pset[2];
        this.duration = pset[3];
        this.recover_rate = chance_recover_cow;
        this.vaccine_days = vaccine_limit_cow;


        this.latent_k = pset[4];
        this.duration_k = pset[5];
        this.vacc_k = vacc_k_cow;
    }
    else if (this.type == 2)
    {
        this.latent_period = pset[2];
        this.duration = pset[3];
        this.recover_rate = chance_recover_pig;
        this.vaccine_days = vaccine_limit_pig;


        this.latent_k = pset[4];
        this.duration_k = pset[5];
        this.vacc_k = vacc_k_pig;
    }
    else
    {
        Console.WriteLine("ERROR : setType() ... wrong host type {0}", this.type);
        Console.WriteLine("   use only 1(cow) or 2(pig) for host type");
```

```
            Console.ReadLine();
        }
    }
}

if (externaltest_sw)
{
    if (this.type == 1)
    {
        this.latent_period = latent_period_cow;
        this.duration = duration_cow;
        this.recover_rate = chance_recover_cow;
        this.vaccine_days = vaccine_limit_cow;


        this.latent_k = latent_k_cow;
        this.duration_k = duration_k_cow;
        this.vacc_k = vacc_k_cow;
    }
    else if (this.type == 2)
    {
        this.latent_period = latent_period_pig;
        this.duration = duration_pig;
        this.recover_rate = chance_recover_pig;
        this.vaccine_days = vaccine_limit_pig;


        this.latent_k = latent_k_pig;
        this.duration_k = duration_k_pig;
        this.vacc_k = vacc_k_pig;
    }
    else
    {
        Console.WriteLine("ERROR : setType() ... wrong host type {0}", this.type);
        Console.WriteLine("   use only 1(cow) or 2(pig) for host type");
        Console.ReadLine();
    }
}
}

/* basic functions */

public void init()              //recover hosts (initialize after creation or end of vaccine effect)
{
    this.status = 0;
    this.latent_count = 0;
    this.sick_count = 0;
    this.vaccine_count = 0;
}



public void infect_host()       //infect hosts (become latent)
{
    if (this.status == 0)
    {
```

```
        this.status = 1;
        this.latent_count = 0;
        this.sick_count = 0;
        this.vaccine_count = 0;
    }
}

public void vaccinate()             //vaccinate hosts
{
    if (this.status == 0)
    {
        this.status = -1;
        this.latent_count = 0;
        this.sick_count = 0;
        this.vaccine_count = 0;
    }
}

public void become_immune()         //make hosts immune
{
    if (this.status == 2)
    {
        this.status = 3;
        this.latent_count = 0;
        this.sick_count = 0;
        this.vaccine_count = 0;
    }
}

/* go procedure functions */

public void get_older()
{
    if (this.status == 1) { this.latent_count++; }
    else if (this.status == 2) { this.sick_count++; }
    else if (this.status == -1) { this.vaccine_count++; }
}

            Console.WriteLine("{0} {1} {2} {3}", wei.DistributionFunction(x: (double)this.latent_count),
                                        gamma1.DistributionFunction(x: (double)this.latent_count),
                                        gamma2.DistributionFunction(x: (double)this.sick_count),
                                        log.DistributionFunction(x: (double)this.sick_count));
    }
    else
    {
        {
            this.status = 2;              //L to I (sigmoid)
        }

        if (vacc_sw)
        {
            {
                this.init();            //V to S (sigmoid)
            }
        }
```

```
            {
                if (rand_call_f(1.00) < this.recover_rate)
                {
                    this.become_immune();       //I to R
                }
                else
                {
                    this.status = 4;            //I to D
                }
            }
        }
    } //end of phase shift

    public static int rand_call_i(int maxvalue)                 //new function for throwing random int values
    {
        Random rand = new Random(Guid.NewGuid().GetHashCode());     //reuse this if generating many
random values
        return (rand.Next(0, maxvalue));                        //returns a random int value ranging (0 <= ? <
maxvalue)
    } //end of rand_call_i

    public static double rand_call_f(double maxvalue)               //new function for throwing random double
values
    {
        Random rand = new Random(Guid.NewGuid().GetHashCode());     //reuse this if generating many
random values
        return (rand.NextDouble() * maxvalue);                  //returns a random double value ranging (0 <= ?
< maxvalue)
    } //end of rand_call_f

    public static double rand_call_f_range(double low, double high)
    {
        double rand_scaled = 0.0;

        while (rand_scaled == 0.0)      //prevent from selecting 0.0
        {
            Random rand = new Random(Guid.NewGuid().GetHashCode());     //reuse this if generating
many random values
            rand_scaled = low + (rand.NextDouble() * (high - low));
        }

        return (rand_scaled);           //returns a random double value ranging (low < ? < high)
    } //end of rand_call_f_range
  } //end of class hosts
}
```

구제역 전파 과정의 모의를 위한 개체 기반 모델의 개발

서울대학교 농림기상 협동과정

황 그 림

(지도교수 김 광 수)

초록 (ABSTRACT IN KOREAN)

전세계적으로 축산산업을 가축전염병으로부터 보호하기 위해 방역조치 및 백신투여 등 다양한 위기대응법이 시행되고 있다. 특히 전염병 전파 모의 모델들이 이러한 해결책을 마련하기 위해 여럿 개발되었으나, 농장 내 및 농장 간에서 일어나는 질병 전파과정 양쪽 모두를 비중 있게 다루는 모델의 개발 및 사용사례는 미미한 편이다. 본 연구에서는 지역단위 가축전염병의 시간에 따른 변화양상을 더욱 세밀하게 파악하기 위해 농장 내 및 농장 간 질병 전파를 추계적으로

모의하는 개체 기반 모델(IBM)을 개발하였다. 해당 모델은 Reed-Frost 역학모델식을 기반으로 개체 사이 및 농장 간의 질병 전염확률을 계산하여 구제역 전파양상을 모의하였다.

모의 모델의 농장 내 질병 전파 과정에 관여하는 모수들을 구하기 위해 또 다른 농장 내 질병 전파 IBM 인 Davis Animal Disease Simulation (DADS) 모델을 사용하였다. 다양한 모수값들을 사용하여 대규모 농장(소 700 마리 또는 돼지 7000 마리)을 대상으로 농장 내 질병 전파를 모의하였고, 그 중에서 DADS 모델의 결과값과 가장 유사한 모의결과를 보인 모수값들을 선정하였다. 농장 간 질병 전파 과정에 관여하는 모수들의 경우 국제수역사무국(OIE)의 세계동물위생정보 시스템(WAHIS) 데이터베이스에서 제공하는 역학보고서를 기반으로 그 값들을 결정하였다.

위의 모수값들을 사용하여 2010 년 말 경상북도 안동 지역 및 2016 년 초 충청남도 지역에서 발발한 구제역 사태를 모의한 결과, 대부분의 구제역 발생농가에서 하루에 약 0.3~1.3km 의 질병 전파 속도를 보였다. 또한 2016 년 충청남도 지역에서 2010 년 경상북도 안동 지역보다 구제역의 질병 전염이 더디게 일어났음을 모수값을 통하여 확인하였다. 이처럼 지역단위의 구제역 사태에 시행된 방역대책을 평가하거나, 차후 구제역 발생 시 적용할 방역대책을 마련하기 위한 농장별 우선순위를 매기는데 이와 같은 모의 모델이 도움이 될 것으로 사료된다. 또한 차후 구제역의 전염경로를 추적하는 기능을 탑재하여 본 모델의 장거리 질병 전파 모의의 정확성을 더욱 향상시킬 수 있을 것으로 기대된다.