Modelling spread of Bluetongue in Denmark: The code.

Græsbøll, Kaare

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Modelling spread of Bluetongue in Denmark: The code.

Kaare Græsbøll

Kongens Lyngby 2012
IMM-TECHNICAL REPORT
# Contents

1 Introduction ......................................................... 1
   1.1 The model ............................................... 1

2 Structure ......................................................... 5
   2.1 dk.f90 ................................................ 6
   2.2 gdata.f90 (A.2) ....................................... 7
   2.3 initialize.f90 (A.3) .................................. 10
   2.4 functions.f90 (A.4) ................................... 10
   2.5 iohosts.f90 (A.5) ................................... 11
   2.6 iomodule.f90 (A.6) ................................... 12
   2.7 host.f90 (A.7) ......................................... 12
   2.8 temperatures.f90 (A.8) ............................... 13
   2.9 makemap.f90 (A.9) ................................... 14
   2.10 viraemia.f90 (A.10) ................................. 14
   2.11 windy.f90 (A.11) .................................... 15
   2.12 midge.f90 (A.12) .................................... 16
   2.13 bite.f90 (A.13) ....................................... 16
   2.14 errorhandl.f90 (A.14) .............................. 17

A Code .......................................................... 19
   A.1 dk.f90 (2.1) ........................................... 20
   A.2 gdata.f90 (2.2) ...................................... 24
   A.3 initialize.f90 (2.3) ................................. 30
   A.4 functions.f90 (2.4) ................................. 33
   A.5 iohosts.f90 (2.5) .................................... 39
   A.6 iomodule.f90 (2.6) ................................... 54
   A.7 host.f90 (2.7) ......................................... 63
   A.8 temperatures.f90 (2.8) ............................ 67
| A.9 makemap.f90 (2.9) | 71 |
| A.10 viraemia.f90 (2.10) | 80 |
| A.11 windy.f90 (2.11) | 84 |
| A.12 midge.f90 (2.12) | 85 |
| A.13 bite.f90 (2.13) | 90 |
| A.14 errorhandl.f90 (2.14) | 96 |
| A.15 Subroutine table | 101 |

**Bibliography**

103
This technical report was produced to make public the code produced as the main project of the PhD project by Kaare Græsbøll, with the title: "Modelling spread of Bluetongue and other vector borne diseases in Denmark and evaluation of intervention strategies".

1.1 The model

The main aim of the PhD thesis the code presented in this Technical report refer to, was to create a simulation model to predict the spread of Bluetongue and other vector borne diseases should they be introduced in Denmark. This model is presented in a publication with the title: "Simulating spread of Bluetongue Virus by flying vectors between hosts on pasture." [1] Which has been preliminary accepted for publication in Scientific Reports, and it is recommended to read this report in conjunction with the article. Mathematical expression and parameter values are not included in this report, but are to be found in the article.

The program code presented is for the case of Bluetongue Virus with cattle as hosts and biting midges as vectors. Bluetongue Virus (BTV) is a non-contagious infectious disease that infects ruminants. In Denmark the primary ruminant of concern is cattle in which the disease causes relative mild symptoms, but do
reduce milk yield and increase the risk of spontaneous abortions. In certain
breeds of sheep BTV can have a very high mortality, and therefore the disease is
considered notifiable by the OIE. Bluetongue does not transmit directly between
ruminants but requires to pass through a blood-sucking vector. Inside the vector
BTV needs to replicate to a certain level and make it from the gut to saliva
glands. This process is very sensitive to temperature and cannot be completed
if the vector is at temperatures below 13°C. Therefore, we take in meteorological
data to calculate the extrinsic incubation period (EIP) a.k.a. the incubation time
in the vectors.

What especially differentiates the model presented in this report with previous
models on vector borne spread is that hosts can be distributed onto pasture
areas. We wanted to create a more process oriented approach so that the
parameters describing spread of disease relates directly to parameters describing
flight patterns of vectors. For a more detailed description of the reasons to
choose this particular model type see the PhD thesis.

The model share in-herd dynamics described in Gubbins (2008) [2] and Sz-
maragd (2009) [3] (Figure 1.1), while the between herd model was presented
in Græsbøll (2012) [1] (Figure 1.2). Vaccination was modeled on UK data in
a collaboration with the Pirbright Institute presented in Græsbøll (2012b) [4]
using vaccination as described in Szmaragd (2010) [5].
1.1 The model

Figure 1.1: The viraemia of the hosts is described by an extended Susceptible Exposed Infectious Recovered (SEIR) model (left), and the vectors are described by an extended SEI model (right). All movements between the stages in the model are governed by the probabilities listed in Græsbøll (2012) [1], in the code most parameters are defined in \texttt{gdata} (section 2.2) or read in from the \texttt{input.txt} file (section 2.6.1).
Figure 1.2: Movement of vectors between herds are described either as active flight or passively carried by wind. This can be interpreted by 2D density distributions. (Left) probability density when vectors are moving by active flight to neighboring areas (arbitrary length scale, darker colors are equivalent to higher probability). (Right) probability density when vectors are carried by wind (arbitrary length scale). Mathematical expression can be found in Græsbøll (2012) [1].
The code was generated with an eye for speed, and alone for this reason the program was twice rebuild from scratch. The main issue with speed relates to the number of vectors: The number of vectors in high seasons is as many as 5,000 vectors per host. Given that there are approximately 1.7 million cattle in Denmark, this round up to almost 10 billion vectors, each need potentially be modeled for flight. This high number is never realistically reached, and optimizing for speed is therefore mainly a problem of best identifying relevant vectors to move. This is primarily achieved by keeping a list of all vectors and hosts with the disease, and only perform relevant operations: viraemia, movement, death, etc. on these animals. In the program this is handled by two lists that cross-reference to each other and to the map. Using a list proved much faster (at least a factor of ten) than just having vectors and host located in a grid matching the map and scanning across areas looking for animals with disease.

The code is structured with a control file `dk.f90` which then call 13 other modules to do as explained in the following sections as listed below (In parenthesis the section in appendix where to locate the code):

2.1 `dk.f90` (A.1) Determines the overall structure of the simulation.

2.2 `gdata.f90` (A.2) Defines the global data available to all modules.
2.3 *initialize.f90* (A.3) Initialises many of the variables and arrays.

2.4 *functions.f90* (A.4) Collates repeated functions.

2.5 *iohosts.f90* (A.5) Reads in pasture maps.

2.6 *iomodule.f90* (A.6) Initialization of pasture, farms, etc.

2.7 *host.f90* (A.7) Reads input from the `input.txt` file.

2.8 *temperatures.f90* (A.8) Handles temperature data.

2.9 *makemap.f90* (A.9) Distributes hosts onto pasture.

2.10 *viraemia.f90* (A.10) Emulates the viraemia of hosts and vectors.

2.11 *windy.f90* (A.11) Handles wind data.

2.12 *midge.f90* (A.12) Does the active and passive movements of vectors.

2.13 *bite.f90* (A.13) Transmission of disease between vectors and hosts.

2.14 *errorhandl.f90* (A.14) Various checks of the code.

Please note that section A.15 contains a full list of all modules and subroutines. Some parts of the code have been omitted to increase the ease of the read, these parts are mainly lines of codes that determines events when running in certain test modes of the program. However there are also omitted reallocation statements that handles sizes of arrays, these were omitted mostly due to space considerations. For a fully functional copy of the code please contact the author. The version number of the code presented is 9.7.4.

2.1 *dk.f90* (A.1)

The main control file calls the subroutines that handles everything. The program is structured in this way because it makes it easy to turn subroutines on and off, which is very convenient in the test phase, and when testing different scenarios with regards to e.g. vaccinations or different countries. The code here presented contains two loops in *dk.f90*, one ’repeat’ which is simple replicates of the simulations, and one ’day’ that goes from day one to the designated ’end-day’. When running sensitivity analysis extra loops can be added to control parameters outside these two loops. Furthermore the control file writes much of the output to file. Often it can be desirable to handle output in a separate module, I chose otherwise because the day-counter is in *dk.f90* it is very convenient to write output from this loop.
2.2 gdata.f90 (A.2)

As the program is build using the module functionality of Fortran 90 it is advantageous to define the global data arrays in a module which is then called by all other modules. Structuring data in this way ensures that this global data is permanently in the computer memory and only in one copy. Using a module to state global data also eliminates the need to move data between subroutines, or call it explicitly. This is very nice if much data is to be moved. However, this do comes on the expense of the detailed knowledge of what goes in and out of the specific subroutines. Thus it sometimes becomes tricky to keep track of all variable names, I sincerely recommend using the \texttt{implicit none}, and also keep a separate file which contains names of the most important variables and arrays and their structure. Here I present parts of my own "variable list", both as an example, but also to ease the read of the code. The number of parenthesis after a variable indicates the dimensionality of the array, so that none = scalar, () = vector, ()() = matrix, and so on. Variable inside [ ] indicates optional variables. the [rand] indicates that variables are only use when generating fields at random.

--- List of variables ================

hosts(y)(x)(#C,C_id,[#S,S_id,]M_id) the spatial map linking to the list
csick_loc (C_id)(y,x,M_id,#H,SEIR) list of infected cattle
ssick_loc (S_id)(y,x,M_id,#H,SEIR) list of infected sheep
msick_loc(M_id)(y,x,C_id,#C,[S_id,#S,]#H,T_id,SEIR) list of infected midges

#C, #S, #M number of cattle, sheep, midges at a given location
#H = #C + #S total number of hosts at a given location

C_id, S_id, M_id the ID number to cross reference lists

--- gdata.f90 ====

-integer-

xsize number of cells in x direction
ysize number of cells in y direction
mstages number of boxes used to emulate midge stages
ios input/output error status
xstart x value in cells of start of epidemic
ystart y value in cells of start of epidemic
msick count of locations of sick midges
n_farms number of cells with hosts [rand]
nmax maximum number of cells in a cluster to represent fields [rand]
nmean mean number of cells in a cluster to represent fields [rand]
input_type is data generated or read from file
area_sel area select between predefined or own definition
infherds count of infected herds
fly_c count of flying events per time unit
day current day of simulation
m_var number of additional variables in the msick_loc array
mboxes number of stages in the midge EIP
mbij integer value of number of boxes to jump when emulate mboxes
h_var number of additional variables in the csick_loc2 array
a_fly the allocated size of the fly array
a_msick the allocated size of the msick array
host_types number of hosts types (cattle, sheep, etc)
fly() array of fly to locations
msick_loc() array with all infected midges
hstages() number of hosts stages (no need to emulate <- temp indep.)
mth() midges per host
hsick() count of locations of sick hosts
n_hosts() number of hosts
hosts() array with hosts location
msick_loc() locations of sick cattle
msick_loc_com() history of infected cattle
ssick_loc() locations of sick sheep
ssick_loc_com() history of infected sheep
hsick_com() counter for csick_loc_com+ssick_loc_com
a_hsic() allocated size of hsick
a_hsic_c() allocated size of hsick_loc_com
mk parameter determining the precision of variables
pi pi=3.1415....

loclen local length scale for the random walk of midges
mrec 1 / EIP for midges
wile width to length scale in wind spread cone
mmort midge daily mortality
sizefac cell side length in km
temperature mean daily temperature
biterate daily bite probability of midges
flytime how long do midges stay in the wind [hours]
midgefly | fraction of midges that fly away from hosts
midgestay | fraction of midges that stay where no hosts
ave_dist | average distance between cells with hosts
meaninf | mean number of infected hosts per affected cell
vac_cov | fraction of hosts immune from vaccination
wind_s | wind speed in current cell
wind_a | angle of the wind in current cell
mbjr | real value of number of boxes to jump when emulate mbxes
hrec() | 1/days for hosts to recover from infection
probtrans_mh() | probability of transmission midge to host
probtrans_hm() | probability of transmission host to midge
hosts_outside() | percentage hosts outside

gdata.f90 also contains statements to turn host movement, movement restriction and a debugging mode (debugging statements have though mostly been removed from the displayed code) on and off.

Additional subroutines included are also primarily connected to the start-up of the program.

2.2.1 init (A.2.1)

Defines many of the parameters characterizing which disease and the effect on specific hosts. These statements are though somewhat obsolete as they are also read in from the input.txt file (section A.6.2). But some are needed to initialize the size of certain data arrays.

2.2.2 redistribute (A.2.2)

This subroutine reset all hosts and vectors arrays to the starting setup. This section is used for test runs when it is not desirable to resample all hosts onto pasture at every repetition.

2.2.3 seeed (A.2.3)

The seeed subroutine ensures that the random seed is indeed random. (Trivia: Seeed is also a name of one of the author’s favorite German bands.)
2.2.4 alloc_cphid (A.2.4)

This subroutine allocate cph numbers to all pasture, so that hosts can be tracked back to their owners.

2.3 initialize.f90 (A.3)

The Initialize module in its full contains initialization of epidemic for both Denmark and the UK, and also the definition of the UK vaccination roll-out plan. Here I have only included the initialization of the Danish epidemic.

2.3.1 init_inf (A.3.1)

It is possible to initiate using infected hosts or infected vectors. Here we also determine the area of first infection.

2.4 functions.f90 (A.4)

The functions module contains functions used throughout the program, but also subroutines to collect data and distribute flying vectors. From the original code some subroutines have been removed. The removed pieces of code did reallocation of arrays and error checks.

2.4.1 BINV (A.4.1)

This function approximates a draw from a binomial distribution. Fortran 90 has only a call to a uniform random distribution. The simple solution would be to do Bernoulli trials, however this method is significantly faster especially when the number of draws, \( n \), is large and the probability, \( p \), low. This combination of \( n \) and \( p \) is often fulfilled in the code given that there are often many vectors but transmission probability is very low. The algorithm was taken from Kachitvichyanukul and Schmeiser (1988) [6]. In this implementation the approximation crashes somewhere above \( np = 500 \) (probably due to rounding to zero), therefore for \( np > 500 \) the algorithm returns \( np \). When running within
reasonable disease parameter and vector abundance this cutoff is almost never in use.

2.4.2 viraspread (A.4.2)

viraspread simply calculates the maximum distance virus has traveled from the starting point.

2.4.3 herd_inf (A.4.3)

This subroutine reports the number of currently infected herds and the prevalence in the herds.

2.4.4 flysort/flysortm (A.4.4)

When implementing dispersal of vectors, dispersing vectors was assigned to a temporary array so that the same vector was not able to fly more than once (see section 2.12). The two modes (active and passive) of vector dispersal can run with the temporary array being either a list or a matrix mode depending on the expected number of flying vectors. The subroutines flysort and flysortm distributes vectors from the temporary array (list and matrix respectively) and makes sure that all indexing arrays are updated.

2.5 iohosts.f90 (A.5)

Until version 9.6 of the program pasture distribution was read in from pre-made files using the read_mark_blocks subroutine. These pre-maps were very large, and it was not feasible to store the number needed to ensure proper sampling. However they did load faster than generating new maps for every run, so when doing test runs it was often used. The o part of iohosts.f90 is represented by subroutines find_herds and find_farms designed to make maps of the spread.
2.5.1 read_mark_blocks (A.5.1)

Reads in pasture data from files and distribute hosts onto these pasture areas. Pasture data for Denmark comes with a percentage of grass, therefore the program tries to distribute hosts onto pasture with high grass percentage first. It is also defined how many cattle per hectare that is maximum allowed.

2.5.2 find_herds/find_farms (A.5.2)

Outputs infected herd/farm locations to file.

2.6 iomodule.f90 (A.6)

This module was prematurely named as an io module, considering that it only does input.

2.6.1 read_inputfile (A.6.1)

To avoid numerous recompiles and to ease use for other users an input.txt was constructed which contains most of the parameters the program needs to run. read_inputfile reads and checks that input file. It also generate some parameters if the input file specified some random distribution. An example of the input file is included in section A.6.2.

2.7 host.f90 (A.7)

This module contains the subroutine to handle host movements.

2.7.1 host_move (A.7.1)

Host movements in the model are modeled by use of a transmission kernel. The kernel is fitted to data from "The Knowledgecenter of Agriculture, Dairy &
Cattle farming" who tracks all movements of cattle in Denmark. Ideally a more process oriented network analysis should form the basis of movements in future models. However, using this simple kernel approach showed very little influence of cattle movement on the output of the model, so I do not believe that this approximation does much difference to the model.

2.8 temperatures.f90 (A.8)

This module handles temperature related subroutines. Specifically the interpolation from daily to hourly temperature and mapping temperature stations so that herds share temperatures with the spatially nearest data point. From the original code was deleted subroutines which handles temperatures for UK, these where similar in function, and mostly differed because the structure of input data differs.

2.8.1 temp_var_h (A.8.1)

Many of the parameters in this model are nonlinear with relation to temperature and most have cutoffs whereunder values go to zero. E.g. the Extrinsic Incubation Period (EIP) have a cutoff at 13.5°C so if the daily mean temperature is below this value virus cannot replicate inside vectors. However it is possible that a part of the day has temperatures above cutoff, and in these periods virus should be able to replicate. To emulate this we interpolate from daily minimum and maximum temperatures using a sinusoidal function to approximate hourly temperature data.

2.8.2 read_temp_id (A.8.2)

The temperature data was provided by the Danish Meteorological Institute (www.dmi.dk) from 39 stations across Denmark. Data was processed for another project regarding Bluetongue (www.nordrisk.dk) where data was interpolated to a grid of 25 by 25 km. To increase the speed of the program a map of same resolution as the pasture map was made which indicate which temperature grid point the pasture should take temperature data from. This subroutine reads this pre-made map.
2.9  makemap.f90 (A.9)

This module contains the subroutine to make the grid cells (mark blocks) that represent the range vectors can locate hosts within.

2.9.1  make_mark_blocks (A.9.1)

While the size of the grid cells in the simulation represents the vectors ability to locate hosts, these are also pasture areas. This subroutine collate the input data with regards to which pasture areas are owned by which farmers, and how much cattle are owned by the individual farmer. Cattle are then randomly distributed onto pasture owned by the farmers according to criteria given in the input.txt file.

2.10  viraemia.f90 (A.10)

The viraemia.f90 module deals with the viraemia of hosts and vectors.

2.10.1  host_viraemia (A.10.1)

This subsections moves the infected hosts through the stages in the SEIR model (figure 1.1).

2.10.2  midge_viraemia (A.10.2)

This subsections moves the exposed vectors through the exposed stages (figure 1.1). The code starts by handling the cases where all exposed and infectious vectors have moved away. Then runs through the list of exposed/infected midges updating the parameters according to temperature, and moving them through the states or killing them. In the end the code handles the case where all exposed and infectious vectors have died. I have left in some out-commented code that claims to ensure constant amount of midges, this is a remnant from when the code ran with a fixed amount of midges, so all midges having died were replaced in the susceptible stage. However, when number of midges became variable on a daily and seasonal basis, the interpretation of number of midges became
equivalent to the number of susceptible midges. This was done because adjusting the total number of midges would be more computationally demanding than the out-commented line, and would also require further assumptions. An example: we sample daily between 1 and 5000 midges, day one there is 5000 midges, and we get 10 exposed midges, the next day we sample that there should be 1 midge. If this number is not assumed to be the susceptible number, we need to consider how to handle the 10 exposed midges from yesterday.

The observant reader might also have noted that the code seems to have 40 stages for the vectors, which is a lot different from the 11 as described in the theory section of Græsbøll (2012) [1]. From hereon the 40 partitions will be referred to as boxes to differentiate from the stages described in the theory. When this code was initially written the number of stages was only given as an interval, and while e.g. Gubbins (2008) [2] handled this by assigning the number of stages for the EIP to each farm, this could not be achieved in this model because vectors move around, and how to handle a vector in stage four out of seven when it arrives at an area with 11 stages? This was solved by emulating stages using the 40 boxes. To exemplify: if there are ten stages at a given location (and for a given temperature) the vectors will jump four boxes forward each time we evaluate the viraemia. In this implementation we can think of the boxes as each representing 2.5% of the completion of EIP. This implementation allows for vectors to move between areas with different number of stages, and number of stages can also vary with temperature if needed. If the number of boxes divided by the number of stages is not an integer, the vectors will be distributed to the nearest boxes in proportions according to the decimal.

### 2.11 windy.f90 (A.11)

The `windy` module handles wind in the code.

### 2.11.1 wind_data (A.11.1)

Wind is drawn from a random distribution. In the code there are some out-commented examples of how to generate anisotropic winds.
2.12  midge.f90 (A.12)

The module midge handles the movement of midges (vectors). Most importantly for the speed of the code, only vectors with the disease are allowed to move. Movement is designed to be handled in two different ways depending on the number of vectors to handle. When dealing with a small epidemic, every vector move is recorded into a list. When many vectors are flying all over the simulation box, the program can use a temporary matrix were all movements are subscribed to. The different modes are subsequently handled by the flysort/flysortm routines A.4.4, which places the flying vector in their landing areas. Parts of the code is out-commented this mostly different ways of handling vectors that flies outside the simulation box. The opted mode is that flights ending outside the simulation box is canceled. Out-commented modes are flights terminates at the edge, or flights outside box means instant dead.

2.12.1  midge_local (A.12.1)

Local flight is simulated by assuming a Gaussian random walk. To generate coordinates the Box-Muller algorithm is utilized [7].

2.12.2  midge.wind (A.12.2)

Midges carried by wind also have coordinates generated using the Box-Muller algorithm [7].

2.13  bite.f90 (A.13)

The bite module handles the transmission of virus between vectors and hosts. In the subroutines there are a lot of code which handles the different index arrays when bites leads to transmission. There is also different ways of handling vaccination included.
2.13.1 inf_hosts (A.13.1)

In this subroutine hosts becomes exposed to the virus by bites from infectious midges.

2.13.2 inf_midges (A.13.2)

In this subroutine susceptible vectors becomes exposed to the virus by biting infectious hosts. Notice that it is emulated that the titres of virus in blood is different in different stages of the disease, as has been seen in experimental data [8].

2.14 errorhandl.f90 (A.14)

In the end I show parts of the testing algorithm. When using index lists it becomes very important to make sure that these list cross-reference exactly so that information is not lost or misinterpreted.

2.14.1 findmidges (A.14.1)

Test if midge IDs are correctly handled in the host list.

2.14.2 findhosts (A.14.2)

Test if host IDs are correctly handled in the host and vector lists.
Following is the source code for version 9.7.4 of the program. This version is only a slight alteration from the version 9.7.2 which was used to produce data in paper [1]. It is also very little difference on this code and the code used to produce paper [4]. This print of the code have been edited from the original to make reading easier. Par example all statements relating to the parallelization of the code have been excluded. Various debugging statements have also been deleted. This code does only represent one simulation of Denmark. For real use additional loop were introduced to do sensitivity analysis.

The subroutines are tabled in section A.15. Subroutines are therefore sectioned, which is not part of the original code.
A.1  dk.f90 (2.1)

PROGRAM dk ! simulate spread of btv in dk 2008

USE gdata
USE functions
USE initialize
USE iohosts
USE iomodule
USE host
USE temperatures
USE makemap
USE viraemia
USE windy
USE midge
USE bite
USE errorhandl

IMPLICIT NONE

INTEGER, PARAMETER :: ntemp=1,nrep=1,nrepeat=1
INTEGER :: i,j,k,h,l,m,temp,myrank,ierror,nproc,n
INTEGER :: init_count,iday,hsicko
REAL(mk) :: x,newdist
CHARACTER :: date*8,hour*10

CALL init
CALL read_inputfile

myrank=0

CALL DATE_AND_TIME(date,hour)
WRITE (*,'(a15,5(a2,a),a4)') 'Starting time: ',hour(1:2), ': ',&
& hour(3:4), ': ',hour(5:6), ': ',date(7:8), '/',date(5:6), '−',date(1:4)
WRITE (*,*) 'Start'
ENDIF

CALL seeced
CALL read_temp_id
endday=400
flyminit=700
init_count=0
repeat=0

vac_cov=0.0

DO repeat=1,nrepeat

DO inf_date=1,365

! call redistribute
! call read_mark_blocks
CALL make_mark_blocks
! call chr_start

DO day=1,endday

IF (day==inf_date) CALL init_inf

!--- update variables dependent on temperature ---
CALL temp_var_h

!--- viraemia for hosts ---- hosts never dies of BT
CALL host_viraemia
!CALL checksick

!--- viraemia for midges ----
CALL midge_viraemia

IF (day==endday) CALL findhosts
CALL checkmsick
! call findmidges

!--- wind determination ---
CALL wind_data

!--- midge local spread ---

IF (day==flyminit) DEALLOCATE(fly)
IF (day>=flyminit) flym=0
CALL midge_local
IF (day<flyminit) CALL flysort

!CALL checkmsick
!--- midge wind spread ---
CALL midge_wind
IF (day<flyminit) THEN
    CALL flysort
ELSE
    CALL flysortm
ENDIF

!CALL checkmsick

!--- host movement ---
IF (hostsmove) CALL host_move

!=== BITE ===

!--- infection of hosts ---
CALL inf_hosts

!--- infection of midges ---
CALL inf_midges

!--- check whether lists contain all virus
CALL findmidges
CALL findcattle

IF (hsick(1)>0) THEN
    iday=−1
    DO n=1,hsick(1)
        IF (n>hsicko) iday=day
        WRITE (29,'(6(i5,x))') &
            & n,day,SUM(csick_loc(n,6:10)),SUM(csick_loc(n,6:11)),&
            & SUM(csick_loc(n,6:11)),iday
    ENDDO
    hsicko=hsick(1)
ENDIF

newdist=viraspread()
infcph=0
CALL herd_inf
!CALL find_farms
!CALL find_clinical_signs !— must be within the day do loop
!if (inf_date == 180) CALL spreadkernel

WRITE(23,'(i7,3(x,i6),3(x,f7.2),f5.2)') &
& SUM(hosts(:,:,1)),infeph,hsick (1),hsick (2),msick,&
& SUM(csick_loc(1:hsick(1),6:11)),&
& SUM(ssick_loc(1:hsick(2),6:13)),&
& SUM(msick_loc(1:msick,2+m_var:2+m_var+mstages)),&
& SUM(msick_loc(1:msick,2+m_var+mstages)),&
& newdist, &
& meaninf,meanpinf,vac_cov!,clincph
flush (23)

ENDDO !— day
ENDDO !— inf_date
!CALL spreadkernel

ENDDO !— repeat

CALL DATE_AND_TIME(date,hour)
WRITE (*,'(a10,5(a2,a),a4)') 'End time: ',hour(1:2),': ',&
& hour(3:4),': ',hour (5:6), ' ',date (7:8), ' / ',date (5:6), ' — ',date (1:4)
WRITE (*,*) 'errorcounter: ',errorcounter
WRITE (*,*) 'initcounter: ',init_count
ENDIF

END PROGRAM
A.2  gdata.f90 (2.2)

MODULE gdata
IMPLICIT NONE

INTEGER :: xsize, ysize, mstages, ios, xstart, ystart, msick, inf_date
INTEGER :: nmax, nmean, input_type, area_sel, infherds, endday
INTEGER :: fly_c, day, m_var, h_var, infcph, repeat, errorcounter
INTEGER :: a_fly, a_msick, host_types, mth, ht, max_id_temp, ukinit, flyinit

INTEGER :: clin, sign(125118), clineph

INTEGER, ALLOCATABLE :: fly(:,;), msick_loc(:,;), rollout(:)
INTEGER, ALLOCATABLE :: hosts(:,;), csick_loc(:,;), county()
INTEGER, ALLOCATABLE :: csick_loc_com(:,), mbboxes(:,), mbjii(:,), flym(:,;)
INTEGER, ALLOCATABLE :: ssick_loc(:,;), cattle(:,;), cphid(:,;)
INTEGER, ALLOCATABLE :: ssick_loc_com(:,;), id_temp(:,;), vac_id(

INTEGER, PARAMETER :: mk=KIND(1.0E0), mht=2

INTEGER :: hstages(mht), hsick(mht), n_hosts(mht)
INTEGER :: hsick_com(mht), n_farms(mht)
INTEGER :: a_hsick(mht), a_hsick_c(mht)
INTEGER :: LocInitO(7,2)
! integer :: mlpar1, mlpar2

REAL, PARAMETER :: pi=3.1415926535897932_mk

REAL(mk) :: loclen, wile, sizefac, temperature
REAL(mk) :: flytime, midgefly, midgestay, meanpinf
REAL(mk) :: ave_dist, meaninf, vac_cov, wind_s, wind_a
REAL(mk) :: efficacy

REAL(mk) :: hmove(mht), hmovef(mht)
REAL(mk) :: hrec(mht), probtrans_mh(mht), probtrans_km(mht), hosts_outside(mht)
REAL(mk) :: tfp(mht)

REAL(mk), ALLOCATABLE :: mrec(:,), biterate(:,), mbjr(:,), mmort(:,), tmin(:,), tmax(:)

! logical :: hostsmove=.TRUE.
LOGICAL :: hostsmove=.FALSE.

LOGICAL :: movingrestrictions=.FALSE.
! logical :: testmode=.TRUE.
LOGICAL :: testmode=.FALSE.

! logical :: testmodef=.TRUE.
LOGICAL :: testmodef=.FALSE.

LOGICAL :: oo_transkernel=.FALSE.

LOGICAL, PARAMETER :: varmth=.FALSE. ! variable amount of midges
LOGICAL, PARAMETER :: varmthdk=.TRUE. ! variable amount of midges dk style

LOGICAL :: varvac=.FALSE. ! variable vaccination cover

CONTAINS

A.2.1 init (2.2.1)

SUBROUTINE init !---- initialise parameter values
!---- mostly obsolete due to input.txt

IMPLICIT NONE

INTEGER :: mit

mstages=40
hstages(1)=5
hrec(1)=REAL(1./20.6,mk)
wile=0.5_mk
probtrans_nh=0.9_mk
probtrans_lm=0.1_mk
sizefac=0.1_mk
flytime=0.25_mk
midgefly=0.05_mk
midgestay=0.5_mk
loclen=0.3_mk/sizefac/0.675_mk
hosts_outside=0.9
vac_cov=0.
errorcounter=0
!m_var=3+2*h
h_var=4
efficacy =1.
tfp(1)=60.

!-- stages of infection to mimic Gamma–dist.
!– Cattle Recovery probabilities 1/days
!– width to length scale in wind spread cone
!– probability of transmission midge — host
!– probability of transmission host — midge
!– cell side length in km
!– length of flight time
!– fraction of midges that fly with the wind
!– fraction of midges that stay where no cattle
!– local length scale for M random walk
!– fraction of farmers who put cattle on pasture

!– must be defined in iomodule
hmove = 0.001  !-- daily probability of moving a host
hmovef = 0.2

! --- SHEEP
hstages(2)=7
hrec(2)=REAL(1./16.4,mk)  !-- Sheep Recovery probabilities 1/days
tfp(2)=14.

ios=0
a_fly =2000
a_msick=10000
a_hsick=100
a_hsick_c=1000
hsick=0
hsick_com=0
hsick_c=0
msick=0
clin_sign =0
clincph=0

OPEN (23,FILE='sick.dat',STATUS='replace',ACTION='write')
OPEN (29,FILE='afi.dat',STATUS='replace',ACTION='write')
OPEN (27,FILE='testing.dat',STATUS='replace',ACTION='write')
OPEN (28,FILE='testing2.dat',STATUS='replace',ACTION='write')
OPEN (21,FILE='/home/kagr/Desktop/Data/temperature/y2008dk.dat',&
 & STATUS='old',ACTION='read')
!open (21, file ='/home/kagr/Desktop/Data/uk/TempDataFor2007.txt',&
 ! & status='old',action='read')
OPEN (62,FILE='inf_data.dat',STATUS='replace',ACTION='write')
OPEN (44,FILE='spread_data.dat',STATUS='replace',ACTION='write')

END SUBROUTINE init

!==============================================
A.2.2 redistribute (2.2.2)

SUBROUTINE redistribute !--- ready to restart with same host distribution
IMPLICIT NONE

INTEGER :: i,j,cows,k
REAL(mk) :: x

hosts(:,:,2)=0
IF (host_types==1) hosts(:,:,3)=0
IF (host_types==2) hosts(:,:,4)=0

hsick(1)=0
csick_loc =0

msick=0
msick_loc=0

hsick_com(1)=0
csick_loc_com=0

clin_sign =0
clinph=0

IF (host_types>1) THEN
hsick(2)=0
hsick_com(2)=0
ssick_loc =0
ssick_loc_com=0
ENDIF

REWIND (21)

a_fly =2000
a_msick=10000

IF (allocated(fly)) DEALLOCATE(fly)
DEALLOCATE(msick_loc)

ALLOCATE(fly(a_fly,3))
ALLOCATE(msick_loc(a_msick,m_var+mstages+2))

fly =0
msick_loc=0

END SUBROUTINE
A.2.3  seeed (2.2.3)

SUBROUTINE seeed !—— seed the random number generator
IMPLICIT NONE

   INTEGER :: isize,idate(8)
   INTEGER,ALLOCATABLE :: iseed(:)

   CALL DATE_AND_TIME(VALUES=idate)
   CALL RANDOM_SEED(SIZE=isize)
   ALLOCATE( iseed(isize) )
   CALL RANDOM_SEED(GET=iseed)
   iseed = iseed + 1
   iseed = iseed * (idate(8)−500) ! idate(8) contains milisecond
   ! iseed = 100 !— non-random run
   CALL RANDOM_SEED(PUT=iseed)

!PRINT *,iseed

END SUBROUTINE

A.2.4  alloc_cphid (2.2.4)

SUBROUTINE alloc_cphid
IMPLICIT NONE

   INTEGER :: n

   ALLOCATE(cphid(ysize,xsize))

   SELECT CASE (area_sel)

   CASE (4,5,8)
      OPEN (42,FILE=''/home/kagr/Desktop/Data/uk/fields/cphraster.dat',&
            STATUS='old',ACTION='read')
   CASE (9)
      OPEN (42,FILE=''/home/kagr/Desktop/Data/uk/fields/cphraster2.dat',&
            STATUS='old',ACTION='read')
   END SELECT
CASE (10)
    OPEN (42, FILE='/home/kagr/Desktop/Data/uk/fields/cphraster5.dat', &
    & STATUS='old', ACTION='read')
END SELECT

DO n = 1, 6
    READ (42, *)
ENDDO

DO n = 1, ysize
    READ (42, *) cphid(n, :)
ENDDO

END SUBROUTINE alloc_cphid

END MODULE
A.3 initialize.f90 (2.3)

MODULE initialize ! the BT epidemic in different ways
CONTAINS

A.3.1 init_inf (2.3.1)

SUBROUTINE init_inf

USE gdata
USE functions
IMPLICIT NONE

INTEGER :: i,j,k,cows
REAL(mk) :: x

IF (.FALSE.) THEN
!==============================================
!−−−−−−initiate start with cattle
!==============================================

ht=1
xstart=xsize/2
ystart=ysize/2
i=ystart
j=xstart

CALL seeed

DO WHILE (hosts(i,j,1)==0)
    CALL random_number(x)
    k=FLOOR(x*xsize*ysize)+1
    i=k/xsize+1
    j=MOD(k,xsize)+1
    ystart=i
    xstart=j
ENDDO

cows=hosts(ystart,xstart,1)

hsick(ht)=1
csick_loc =0
csick_loc (1,1:2)=(/ystart, xstart/)
csick_loc (1, h_var+1)=MAX(cows−5,1)
csick_loc (1, h_var+2)=5

msick=0
msick_loc=0

hsick_com(ht)=1
csick_loc_com=0
csick_loc_com (1,1:2)=(/ystart, xstart/)  
csick_loc_com (1,3)=0
ENDIF

!=============================================  
!--- initiate start with midges  
!=============================================  

xstart=NINT(50./sizefac)  
ystart=NINT(50./sizefac)  
i=ystart  
j=xstart

DO WHILE (hosts(i,j,1)==0)  
   i=i+1  
   ystart=i
ENDDO

hsick=0  
csick_loc =0  
ssick_loc =0

msick=1
msick_loc=0

hosts(i, j,1+2*host_types)=msick  
msick_loc(msick,1:2)=(/i, j/)  
msick_loc(msick,3:4)=(/ hosts(i, j ,2), hosts(i, j ,1)/)  
msick_loc(msick,5:)=0  
msick_loc(msick,3+2*host_types)=hosts(i,j,1)
IF (host_types>1) THEN  
   msick_loc(msick,3+2*host_types)=msick_loc(msick,3+2*host_types)+hosts(i,j,3)  
   msick_loc(msick,5:6)=(/ hosts(i, j ,4), hosts(i, j ,3)/) 
ENDIF
msick_loc(msick,m_var+mstages+2)=10
hsick_com=0
csick_loc_com=0
ssick_loc_com=0

END SUBROUTINE

END MODULE
MODULE functions ! contains functions used repeatedly
! OPEN–statements should be in 60’ties

CONTAINS

A.4.1 BINV (2.4.1)

!=== BINV simulates a draw from binomial distribution
!=== by V. Kachitvichyanukul and B.W. Schmeiser

FUNCTION binv(pp,n)
IMPLICIT NONE

INTEGER :: binv
INTEGER, INTENT(IN) :: n
DOUBLE PRECISION, INTENT(IN) :: pp

DOUBLE PRECISION :: p,q,s,a,r,u

p=pp

IF (pp>=1.) THEN
    binv=n
    'print *,’Input error in functions:binv’
    RETURN
ENDIF

IF (pp>0.5) p=(1.–pp)

IF (p*n > 500.) THEN
    binv=NINT(p*n)
    IF (pp>0.5) binv=n–binv
    RETURN
ENDIF

IF (n==0) THEN
    binv=0
    RETURN
ENDIF

q=1.–p
s = p/q

a = REAL(n+1.KIND(1.0D0)) * s
r = q**REAL(n,KIND(1.0D0))

CALL random_number(u)
binv = 0

DO WHILE (u >= r .and. binv < n)
    u = u - r
    binv = binv + 1
    r = (a/binv - s) * r
ENDDO

IF (pp > 0.5) binv = n - binv

END FUNCTION

!====================================

A.4.2 viraspread (2.4.2)

FUNCTION viraspread()

USE gdata

IMPLICIT NONE

REAL(mk) :: viraspread
INTEGER :: i, j, h

viraspread = 0._mk

DO ht = 1, host_types
    DO h = 1, hsick(ht)
        IF (ht == 1) THEN
            i = csick_loc(h, 1)
            j = csick_loc(h, 2)
        ENDIF
        IF (ht == 2) THEN
            i = ssick_loc(h, 1)
        ENDIF
    END DO
END DO
A.4.3 herd_inf (2.4.3)

SUBROUTINE herd_inf

USE gdata

IMPLICIT NONE

INTEGER :: k, h, i, j, cmi
REAL :: cpi

infherds=0
cmi=0
cpi=0.

DO h=1,hsick(ht)
  IF (hsick(ht)>0) THEN
    DO h=1,hsick(ht)
      IF (ht==1) THEN
        IF (csick_loc(h,1)>0) THEN
          infherds=infherds+1
          cmi=cmi+SUM(csick_loc(h,h_var+2:(h_var+hstages(ht)+2)))
          IF (SUM(csick_loc(h,h_var+1:(h_var+hstages(ht)+2))>0) THEN
            cpi=cpi+SUM(csick_loc(h,h_var+2:(h_var+hstages(ht)+2)))&
            & /REAL(SUM(csick_loc(h,h_var+1:(h_var+hstages(ht)+2))))
          ELSE
            cpi=cpi+0.
          ENDIF
        ENDIF
      ELSE
        ENDIF
    END DO
 ENDIF
ENDIF

IF (inf_date==170) WRITE (44,*) csick_loc(h,2),csick_loc(h,1),repeat
ENDIF

IF (ht==2) THEN
IF (ssick_loc(h,1)>0) THEN
infherds=infherds+1

cmi=cmi+SUM(ssick_loc(h,h_var+2:(h_var+hstages(ht)+2)))

IF (SUM(ssick_loc(h,h_var+1:(h_var+hstages(ht)+2)))>0) THEN
  cpi=cpi+SUM(ssick_loc(h,h_var+2:(h_var+hstages(ht)+2)))
  & /REAL(SUM(ssick_loc(h,h_var+1:(h_var+hstages(ht)+2))))
ELSE
  cpi=cpi+0.
ENDIF
ENDIF

ENDIF

IF (inf_date==170) WRITE (44,*) ssick_loc(h,2),ssick_loc(h,1),repeat
ENDIF

ENDDO
ENDIF
ENDDO

meaninf=0.
meanpinf=0.
IF (infherds>0) THEN
  meaninf=REAL(cmi)/REAL(infherds)
  meanpinf=cpi/REAL(infherds)
ENDIF

END SUBROUTINE

!====================================================================

A.4.4 flysort/flysortm (2.4.4)

SUBROUTINE flysort

USE gdata

IMPLICIT NONE

INTEGER :: h,i,j
ht=host_types

DO h=1,fly_c
i=fly(h,1)
j=fly(h,2)

IF (hosts(i,j,1+2*ht)==0) THEN
  msick=msick+1
  IF (msick>=a_msick) CALL re_al_msick
  hosts(i,j,1+2*ht)=msick
  IF (hosts(i,j,2)/=0) csick_loc(hosts(i,j,2),3)=msick
  IF (ht>1 .and. hosts(i,j,4)/=0) ssick_loc(hosts(i,j,4),3)=msick
  msick_loc(msick,1:2)=(/i,j/)
  msick_loc(msick,3:4)=(/ hosts(i,j,2), hosts(i,j,1)/)
  msick_loc(msick,3+2*ht)=hosts(i,j,1)
ENDIF
msick_loc(msick,m_var+1:)=0
ENDIF
msick_loc(hosts(i,j,1+2*ht),fly(h,3)) = &
& msick_loc(hosts(i,j,1+2*ht),fly(h,3)) + 1
ENDDO
END SUBROUTINE

==================================
SUBROUTINE flysortm
USE gdata
IMPLICIT NONE

INTEGER :: h,i,j
ht=host_types

DO h=1,41
  DO j=1,xsize
    DO i=1,ysize
      IF (flym(i,j,h)==0) CYCLE

      IF (hosts(i,j,1+2*ht)==0) THEN
        msick=msick+1
        IF (msick>=a_msick) CALL re_al_msick
        hosts(i,j,1+2*ht)=msick
        IF (hosts(i,j,2)/=0) csick_loc(hosts(i,j,2),3)=msick
      ENDIF
    ENDDO
  ENDDO
ENDDO
IF (ht>1 .and. hosts(i, j,4)/=0) ssick_loc (hosts(i, j,4),3)=msick
msick_loc(msick,1:2)=(/i, j/)
msick_loc(msick,3:4)=(/ hosts(i, j,2), hosts(i, j,1)/)
msick_loc(msick,3+2*ht)=hosts(i,j,1)
IF (ht>1) THEN
  msick_loc(msick,5:6)=(/ hosts(i, j,4), hosts(i, j,3)/)
  msick_loc(msick,3+2*ht)=msick_loc(msick,3+2*ht)+hosts(i,j,3)
ENDIF
msick_loc(msick,m_var+1:)=0
ENDIF
& msick_loc(hosts(i, j,1+2*ht),m_var+1+h) = &
& msick_loc(hosts(i, j,1+2*ht),m_var+1+h) + flym(i,j,h)
ENDDO
ENDDO
ENDDO

END SUBROUTINE flysortm

END MODULE
A.5 iohosts.f90 (2.5)

MODULE iohosts ! Distributes cattle from raster files

CONTAINS

A.5.1 read_mark_blocks (2.5.1)

SUBROUTINE read_mark_blocks

USE gdata
USE functions
IMPLICIT NONE

INTEGER :: ndata, ndata2
INTEGER :: n, i, hist (0:432)
INTEGER :: idump, j, indhi, indlow, field
INTEGER :: cpumapp, oldchr, count
INTEGER :: frommapnr, tomapnr, mapnr, nc, ns
INTEGER :: ncols, nrows, nodat, val, k, unikbloknr
INTEGER :: trow, brow, lcol, rcol, location, tkpa, tmkpa

INTEGER, ALLOCATABLE :: map(:, :), cpumap(:, :], cpu(:, :], chr2(:))
INTEGER, ALLOCATABLE :: chr(:, :), cpun(:, :), upc(:, :), chk(:, :)
INTEGER, ALLOCATABLE :: pos2(:, :), chk2(:, :)

REAL :: xllc, yllc, cellsize, rdump, dist, distc, ccpu, fwp
REAL :: totalarea, cattdens, x, cutoff, x

REAL, ALLOCATABLE :: area(:, :), graes(:, :), kvaeg(:, :), kvaeg2(:, :)
REAL, ALLOCATABLE :: kpa(:, :), totarea(:, :)

CHARACTER :: dump*14, temp*8, cdump*99, prename*54
CHARACTER, ALLOCATABLE :: unikblok(:, :)

CALL seeed

SELECT CASE (area_sel)

CASE (1, 11:13) ! Denmark without bornholm read from file --------
! maps 001–004 are with 100% cattle outside
! maps 101–104 are with 50% cattle outside
! maps 201–204 are with 0% cattle outside
frommapnr=101
tomapnr=104

CALL random_number(x)
mapnr=NINT((tomapnr-frommapnr+1)*x-0.5)+frommapnr
mapnr=MIN(mapnr,tomapnr); mapnr=MAX(mapnr,frommapnr)

IF (area_sel==1) WRITE (prename,'(a41,i3.3,a9)') &
& '/home/kagr/Desktop/Data/fields/cattle_dk/',mapnr,'dkmap.dat'

IF (area_sel==11) WRITE (prename,'(a41,i3.3,a10)') &
& '/home/kagr/Desktop/Data/fields/cattle_dk/',mapnr,'dkmap2.dat'

IF (area_sel==12) WRITE (prename,'(a41,i3.3,a10)') &
& '/home/kagr/Desktop/Data/fields/cattle_dk/',mapnr,'dkmap3.dat'

IF (area_sel==13) WRITE (prename,'(a41,i3.3,a10)') &
& '/home/kagr/Desktop/Data/fields/cattle_dk/',mapnr,'dkmap5.dat'

OPEN (1,FILE=prename,STATUS='old',ACTION='read')

READ (1,*) dump,ncols
READ (1,*) dump,nrows
READ (1,*) dump,xllc
READ (1,*) dump,yllc
READ (1,*) dump,cellsize
READ (1,*) dump,nodat_val

ysize=nrows
xsize=ncols
sizefac = cellsize /1000.

IF (allocated(hosts) .eqv. .false .) THEN
ALLOCATE(hosts(ysize,xsize,1+2*host_types))
ENDIF
hosts=0

DO n=1,nrows

READ (1,*) hosts(n,:,1)
ENDDO
CLOSE (1)
CASE (2) !---lolland---

OPEN (1, FILE = '/home/kagr/Desktop/Data/fields/ubr_lol.dat', &
     STATUS = 'old', ACTION = 'read')
OPEN (11, FILE = '/home/kagr/Desktop/Data/fields/check_lol.dat', &
     STATUS = 'old', ACTION = 'read')
OPEN (2, FILE = '/home/kagr/Desktop/Data/fields/kvaegdata_lol.dat', &
     STATUS = 'old', ACTION = 'read')
!OPEN (3, file = 'cattle_lol.dat', status = 'replace', action = 'write')

ndata = 452; ndata2 = 182
fwf = 1./(92./182.) !-- fWF = 1/farms with fields

CASE (3) !---jutland

OPEN (1, FILE = '/home/kagr/Desktop/Data/fields/ubr_jut.dat', &
     STATUS = 'old', ACTION = 'read')
OPEN (11, FILE = '/home/kagr/Desktop/Data/fields/check_jut.dat', &
     STATUS = 'old', ACTION = 'read')
OPEN (2, FILE = '/home/kagr/Desktop/Data/fields/kvaegdata_jut.dat', &
     STATUS = 'old', ACTION = 'read')
!OPEN (3, file = 'cattle_jut.dat', status = 'replace', action = 'write')

ndata = 5224; ndata2 = 1128
fwf = 1./(863./1128.) !-- fWF = 1/farms with fields

CASE (5) !--- England farms

OPEN (1, FILE = '/home/kagr/Desktop/Data/uk/EnglandFarms.txt', &
     STATUS = 'old', ACTION = 'read')

ysize = 4880
xsize = 5182

IF (allocated(hosts) .eqv. .false.) THEN
ALLOCATE(hosts(ysize,xsize,1+2*host_types))
hosts=0
ENDIF

DO n = 1, 195977

READ (1, '(2(i6,x),2(i5,x))') j, i, nc, ns
hosts(NINT(i/100.),NINT(j/100.),1)=nc
IF (host_types==2) hosts(NINT(i/100.),NINT(j/100.),3)=ns

ENDDO

CASE (6) !---- full Denmark ---------------

OPEN (1,FILE='/home/kagr/Desktop/Data/fields/maps/unikblokraster.txt',&
&               STATUS='old',ACTION='read')
OPEN (11,FILE='/home/kagr/Desktop/Data/fields/check.dat',&
&               STATUS='old',ACTION='read')
OPEN (2,FILE='/home/kagr/Desktop/Data/fields/kvaegdata.dat',&
&               STATUS='old',ACTION='read')

ndata=107118; ndata2=22092
fwf=1./(16641./22092.) !-- fwf = 1/farms with fields

CASE (7) !---- Denmark without bornholm

OPEN (1,FILE='/home/kagr/Desktop/Data/fields/ubr.dk-b.dat',&
&               STATUS='old',ACTION='read')
OPEN (11,FILE='/home/kagr/Desktop/Data/fields/check.dk-b.dat',&
&               STATUS='old',ACTION='read')
OPEN (2,FILE='/home/kagr/Desktop/Data/fields/kvaegdata.dk-b.dat',&
&               STATUS='old',ACTION='read')

ndata=81836; ndata2=21877
fwf=1./(15442./21877.) !-- fwf = 1/farms with fields

CASE (8:10) !---- ukpasture ---------------

! maps 201–204 are with 50% hosts outside 30% with resc 5
! maps 101–104 are with 25% hosts outside
! maps 001–004 are with 0% hosts outside

frommapnr=201
tomapnr=201

!---- read in cattle

CALL random_number(x)
mapnr=NINT((tomapnr–frommapnr+1)*x–0.5)+frommapnr
mapnr=MIN(mapnr,tomapnr); mapnr=MAX(mapnr,frommapnr)

IF (area_sel==8) &
& WRITE (prename,'(a34,i3.3,a11)') &
& '/home/kagr/Desktop/Data/uk/fields/',mapnr,'ukmap_c.dat'

IF (area_sel==9) &
& WRITE (prename,'(a34,i3.3,a12)') &
& '/home/kagr/Desktop/Data/uk/fields/',mapnr,'ukmap_c2.dat'

IF (area_sel==10) &
& WRITE (prename,'(a34,i3.3,a12)') &
& '/home/kagr/Desktop/Data/uk/fields/',mapnr,'ukmap_c5.dat'

OPEN (1,FILE=prename,STATUS='old',ACTION='read')

READ (1,*) dump,ncols
READ (1,*) dump,nrows
READ (1,*) dump,xllc
READ (1,*) dump,yltc
READ (1,*) dump,cellsize
READ (1,*) dump,nodat_val

ysize=nrows
xsize=ncols
sizefac = cellsize/1000.

IF (allocated(hosts) .eqv. .false .) THEN
ALLOCATE(hosts(ysize,xsize,1+2*host_types))
hosts=0
ENDIF

DO n=1,nrows
    READ (1,*) hosts(n,:,1)
ENDDO

CLOSE (1)
!--- read in sheep

CALL random_number(x)
mapnr=NINT((tomapnr-frommapnr+1)*x-0.5)+frommapnr
mapnr=MIN(mapnr,tomapnr); mapnr=MAX(mapnr,frommapnr)

IF (area_sel==8) &
& WRITE (prename,'(a34,i3.3,a11)') &
& '/home/kagr/Desktop/Data/uk/fields/',mapnr,'ukmap_s.dat'

IF (area_sel==9) &
& WRITE (prename,'(a34,i3.3,a12)') &
& '/home/kagr/Desktop/Data/uk/fields/',mapnr,'ukmap_s2.dat'

IF (area_sel==10) &
& WRITE (prename,'(a34,i3.3,a12)') &
& '/home/kagr/Desktop/Data/uk/fields/',mapnr,'ukmap_s5.dat'

OPEN (1,FILE=prename,STATUS='old',ACTION='read')

READ (1,*) dump,ncols
READ (1,*) dump,nrows
READ (1,*) dump,xllc
READ (1,*) dump,yllc
READ (1,*) dump,cellsize
READ (1,*) dump,nodat_val

ysize=nrows
xsize=ncols

DO n=1,nrows

   READ (1,*) hosts(n,:,3)

ENDDO

END SELECT

!OPEN (17,file='chist3.dat',status='replace ', action='write ')
!OPEN (22,file='test.dat', status='replace ', action='write ')

IF (allocated(cpu) .eqv. .false. .and. area_sel /=1) THEN
ALLOCATE(cpu(ndata,3),chr2(ndata2))
ALLOCATE(chr(ndata),cpun(ndata),upc(ndata),chk(ndata))
ALLOCATE(pos2(ndata2,2),chk2(ndata2))
ALLOCATE(area(ndata),graes(ndata),kvaeg(ndata),kvaeg2(ndata2))
ALLOCATE(kpa(ndata),totarea(ndata))
ALLOCATE(unikblok(ndata))
ENDIF

IF (allocated( csick_loc ) .eqv. .false .) THEN
   ALLOCATE( csick_loc(a_hsick(1),h_var+hstages(1)+2))
ALLOCATE(msick_loc(a_msick,m_var+mstages+2))
ALLOCATE(csick_loc_com(a_hsick_c(1),4))
ENDIF

csick_loc = 0
msick_loc = 0
csick_loc_com = 0

IF (allocated(ssick_loc) .eqv. .false. .and. host_types > 1) THEN
ALLOCATE(ssick_loc(a_hsick(2),2+h_var+hstages(2)))
ALLOCATE(ssick_loc_com(a_hsick_c(2),4))
ENDIF

ssick_loc = 0
ssick_loc_com = 0

IF (allocated(fly) .eqv. .false.) THEN
ALLOCATE(fly(a_fly,3))
ENDIF

SELECT CASE (area_sel)
  CASE (1,5,8:13)
    RETURN
END SELECT

!--- READ FILES -------------------

READ (11,*) cdump

DO n=1,ndata
  READ (11,'(i6,x,a9,x,f12.3,x,f8.3,x,i4,x,i4)') &
  & chr(n),unikblok(n),area(n),graes(n),upc(n),cpun(n)
ENDDO

CLOSE (11)

!PRINT *,MAXVAL(cpun(:)),MAXLOC(cpun(:))

READ (2,*) cdump

DO n=1,ndata2
  READ (2,*) cdump,pos2(n,1),pos2(n,2),chr2(n),kvaeg2(n)
ENDDO

CLOSE (2)

!PRINT *,SUM(kvaeg2(:))

CALL seed

!------ Randomly assign which farms have cattle on grass

i=1
kvaeg=0.
chk2=0
oldchr=0

DO n=1,ndata

DO WHILE (chr(n)>=chr2(i))

IF (chr(n)==chr2(i)) THEN

IF (chr(n)!=oldchr) THEN

CALL random_number(x)

IF (x < fwf*hosts_outside(1)) THEN !- fwf = 1/farms with fields

kvaeg(n)=kvaeg2(i)
chk2(i)=1

ELSE

kvaeg(n)=0.

ENDIF

ELSE

kvaeg(n)=kvaeg2(i)

ENDIF

ENDIF

i=i+1

ENDDO

i=i-1

oldchr=chr(n)
!--- Distribute cattle onto grass fields, prioritize fields with !--- highest grass percentage, distribute with max 50 cows/ha.

cutoff=90.0000 !--- less than cutoff grasses => no cows on this field
totarea=0.
kpa=0.
i=0
chk=0

DO n=1,ndata

   IF (i==0) THEN
      i=upc(n)
j=upc(n)
indlow=n-(upc(n)-i)
indhi=n+i-1
cutoff=90.0000
totarea=0.1

   DO WHILE (kvaeg(n)/totarea > 50./10000. .and. j>0)

      CALL random_number(x)

      field =NINT(x*(indhi-indlow-1))+indlow

      IF (graes(field) > cutoff .and. chk(field)==0) THEN
         totalarea=totalarea+area(field)/REAL(cpun(field))
         chk(field)=1
      ENDIF

      j=j-1

      IF (j==0) THEN
         cutoff=cutoff-10.
      IF (cutoff > 1.) j=upc(n)
   ENDIF

ENDDO
ENDIF

IF (chk(n)==1) THEN

totarea(n)=totalarea
kpa(n)=kvaeg(n)/totalarea*10000.

ENDIF

i=i−1

! WRITE (3, '(i6,x,a9,x,f12.3,x,f8.3,x,i4,x,i4,x,f8.3,x,f15.3,x,f9.4)') &
!& chr(n),unikblok(n),area(n),graes(n),upc(n),cpun(n),kvaeg(n),totarea(n),&
!& kpa(n)

ENDDO

!—— data extraction
IF (2<1) THEN

hist=0

DO n=1,ndata

  IF (kpa(n)>0.001) hist(NINT(kpa(n)))=hist(NINT(kpa(n)))+1

ENDDO

DO n=0,432

  WRITE (17, ∗) n,hist(n)

ENDDO

PRINT *,maxval(kpa(:)),minval(kpa(:)),minval(totarea(:)),minloc(totarea(:))
println *,sum(hist(:))

CLOSE (17)

ENDIF

!—— READ RASTER FILE

READ (1,∗) dump,ncols
READ (1,∗) dump,nrows
READ (1,∗) dump,xllc
READ (1,*) dump,yllc  
READ (1,*) dump,cellsizesize  
READ (1,*) dump,nodat_val

!PRINT *,ncols,nrows,nodat_val,cellsizesize

ALLOCATE(map(ncols,nrows))  
ALLOCATE(cpumap(ncols,nrows))  
cpumap=0

DO i=1,nrows
  READ (1,*) map(:,i)  
  !PRINT *,i
ENDDO

CLOSE (1)

DO n=1,ndata
  temp=unikblok(n)(1:6)//unikblok(n)(8:9)  
  READ (temp,'(i8)') unikbloknr

  tmkpa=MAX(NINT(2.∗kpa(n)),1)  
  tkpa=binv(REAL((kpa(n)/REAL(tmkpa)),KIND(1.0D0)),tmkpa)

  cpu(n,:)=(/ chr(n) , unikbloknr , tkpa /) !==== NB ====  
ENDDO

!--- Assign Cattle data ---------------------

lcol =1; rcol=ncols  
brow=1; trow=nrows

ysize=(rcol−lcol+1)  
xsize=(trow−brow+1)

sizefac = cellsize /1000.

IF (allocated(cattle) .eqv. .false .) THEN  
ALLOCATE(cattle(ysize,xsize,3))  
ENDIF
IF (allocated(fly) .eqv. .false.) THEN
ALLOCATE(fly(a_fly,3))
ENDIF

IF (allocated(hosts) .eqv. .false.) THEN
ALLOCATE(hosts(ysize,xsize,1+2*host_types))
hosts=0
ENDIF

cattle=0
!midge=0

cpumap=0
cpumapp=0
!k=20
k=0
count=ndata2−SUM(chk2(:))

DO n=1,ndata2

IF (chk2(n)==0) THEN
  IF (k>0 .and. count>1) THEN
    cpumapp=cpumapp+kvaeg2(n)
    k=k−1
    count=count−1
    CYCLE
  ENDIF
  cpumapp=cpumapp+kvaeg2(n)
  j=FLOOR(REAL(pos2(n,1)−xllc)/cellsize)+1
  i=FLOOR(REAL(pos2(n,2)−yllc)/cellsize)+1
  cpumap(j,i)=cpumapp
  !k=binv(REAL(0.5,KIND(1.0D0)),40)
  k=0
  count=count−1
  cpumapp=0
ENDIF

ENDDO
DO i=brow,trow
  DO j=lcol,rcol

    cpumapp=0

    IF (map(j,i) /= nodat_val) THEN
      DO n=1,ndata
        IF (map(j,i) == cpu(n,2)) THEN
          cpumap(j,i) = cpumap(j,i) + cpu(n,3)
        ENDIF
      ENDDO
    ENDIF

    cattle(j,i,1) = cpumap(j,i)
    ! midges(j,i,1) = mtc * cpumap(j,i)
    ! WRITE (22,'(i5,x)',advance='no') cpumap(j,i)

  ENDDO

ENDIF

ENDDO

! WRITE (22,'(a)')

ENDDO

! close(22)

! print *, sum(cpumap(:,:,))
hosts=0
hosts(:,:,1:2) = cattle(:,:,1:2)
! hosts(:,:,3:4) = cattle(:,:,1:2)/2
DEALLOCATE(cattle)

END SUBROUTINE read_mark_blocks

!====================================

A.5.2 find_herds/find_farms (2.5.2)
SUBROUTINE find_herds
USE gdata
IMPLICIT NONE

INTEGER :: n,cph(125118)

IF (allocated(cphid) .eqv. .false .) CALL alloc_cphid

DO ht=1,host_types
   DO n=1,hsick(ht)
      IF (ht==1) THEN
         cph(cphid(csick_loc(n,1), csick_loc(n,2)))=1
         IF (day==365) WRITE (44,**) csick_loc(n,2),csick_loc(n,1),repeat
      ENDIF
      IF (ht==2) THEN
         cph(cphid(ssick_loc (n,1), ssick_loc (n,2)))=1
         IF (day==365) WRITE (44,**) ssick_loc(n,2),ssick_loc(n,1),repeat
      ENDIF
   ENDDO
ENDDO

infcph=SUM(cph(:))

END SUBROUTINE find_herds

!=================================================================================================

SUBROUTINE find_farms
USE gdata
IMPLICIT NONE

INTEGER :: n,cph(125118),field

IF (allocated(cphid) .eqv. .false .) CALL alloc_cphid

DO ht=1,host_types
   DO n=1,hsick(ht)
      IF (ht==1) THEN
         cph(cphid(csick_loc(n,1), csick_loc(n,2)))=1
         IF (day==365) WRITE (44,**) csick_loc(n,2),csick_loc(n,1),repeat
      ENDIF
      IF (ht==2) THEN
         cph(cphid(ssick_loc (n,1), ssick_loc (n,2)))=1
         IF (day==365) WRITE (44,**) ssick_loc(n,2),ssick_loc(n,1),repeat
      ENDIF
   ENDDO
ENDDO

END SUBROUTINE find_farms
IF (ht==1) THEN  
! if (SUM(csick_loc(n,h_var+2:h_var+hstages(1)+1))>0) then  
  field =cphid(csick_loc(n,1), csick_loc(n,2))  
cph(field)=1  
! if (day==endday) write (44,* ) csick_loc(n,2), csick_loc(n,1), repeat  
IF (day==endday) WRITE (44,*) field,repeat  
! endif  
ENDIF

IF (ht==2) THEN  
! if (SUM(ssick_loc(n,h_var+2:h_var+hstages(2)+1))>0) then  
  field =cphid(ssick_loc(n,1), ssick_loc(n,2))  
cph(field)=1  
! if (day==endday) write (44,* ) ssick_loc(n,2), ssick_loc(n,1), repeat  
IF (day==endday) WRITE (44,*) field,repeat  
! endif  
ENDIF

ENDDO  
ENDDO  

infcph=SUM(cph(:))  

END SUBROUTINE find_farms  

END MODULE
A.6 iomodule.f90 (2.6)

MODULE iomodule

CONTAINS

A.6.1 read_inputfile (2.6.1)

SUBROUTINE read_inputfile

USE gdata
USE iohosts
USE functions
IMPLICIT NONE

INTEGER :: n,check
CHARACTER :: dump*10,vdump*19

OPEN (41,FILE='input.txt',STATUS='old',ACTION='read')
check=1
!--- read header
DO n=1,22
   READ (41,* ) dump
ENDDO

!--- read section A
READ (41,* ,IOSTAT=ios,ERR=666) vdump,input_type
check=2
DO n=1,8
   READ (41,* ) dump
ENDDO

!--- read section B
READ (41,* ,IOSTAT=ios,ERR=666) vdump,area_sel
check=3
DO n=1,5
   READ (41,* ) dump
ENDDO

IF (input_type==1 .and. area_sel==4) THEN
   READ (41,* ,IOSTAT=ios,ERR=666) vdump,sizefac
   READ (41,* ) dump
READ (41,*) dump
check=4
READ (41,*,IOMOD=io,ERR=666) vdump,xsize
   xsize=NINT(real(xsize)/sizefac)
READ (41,*,IOMOD=io,ERR=666) vdump,ysize
   ysize=NINT(real(ysize)/sizefac)
check=5
READ (41,*) dump
READ (41,*) dump
READ (41,*,IOMOD=io,ERR=666) vdump,n_farms(1)
READ (41,*,IOMOD=io,ERR=666) vdump,n_hosts(1)
READ (41,*) dump
READ (41,*) dump
READ (41,*,IOMOD=io,ERR=666) vdump,n_farms(2)
READ (41,*,IOMOD=io,ERR=666) vdump,n_hosts(2)
check=6
READ (41,*) dump
READ (41,*) dump
READ (41,*,IOMOD=io,ERR=666) vdump,nmean
   nmean=MAX(nmean−1,0)
READ (41,*,IOMOD=io,ERR=666) vdump,nmax
   IF (nmax<nmean) GOTO 666
check=7
ELSE
   DO n=1,17
      READ (41,*) dump
   ENDDO
ENDIF
DO n=1,7
   READ (41,*) dump
ENDDO

! --- read section C
READ (41,*,IOMOD=io,ERR=666) vdump,host_types
   host_types=MIN(host_types,2)
   m_var=3+2*host_types
READ (41,*) dump
READ (41,*) dump

READ (41,*,IOSTAT=ios,ERR=666) vdump,hosts_outside(1)
   hosts_outside=hosts_outside(1)/100.
READ (41,*) dump
READ (41,*) dump
check=8
READ (41,*,IOSTAT=ios,ERR=666) vdump,mstages
READ (41,*,IOSTAT=ios,ERR=666) vdump,hstages(1)
READ (41,*,IOSTAT=ios,ERR=666) vdump,hstages(2)
READ (41,*) dump
READ (41,*) dump
check=9
READ (41,*,IOSTAT=ios,ERR=666) vdump,hrec(1)
   hrec(1)=1./hrec(1)
READ (41,*,IOSTAT=ios,ERR=666) vdump,hrec(2)
   hrec(2)=1./hrec(2)
READ (41,*) dump
READ (41,*) dump
check=10
READ (41,*,IOSTAT=ios,ERR=666) vdump,wile
READ (41,*) dump
READ (41,*) dump
check=11
READ (41,*,IOSTAT=ios,ERR=666) vdump,probtrans_mh(1)
   probtrans_mh=probtrans_mh(1)
READ (41,*) dump
READ (41,*) dump
check=12
READ (41,*,IOSTAT=ios,ERR=666) vdump,probtrans_hm(1)
   probtrans_hm=probtrans_hm(1)
READ (41,*) dump
READ (41,*) dump
check=13
READ (41,*,IOSTAT=ios,ERR=666) vdump,flytime
READ (41,*) dump
READ (41,*) dump
check=14
READ (41,*,IOSTAT=ios,ERR=666) vdump,midgestay
READ (41,*) dump
READ (41,*) dump
check=15
A.6 iomodule.f90 (2.6)

READ (41,*) dump
READ (41,*) dump
check=16
READ (41,*,IOSTAT=ios,ERR=666) vdump,loclen
READ (41,*) dump
READ (41,*,IOSTAT=ios,ERR=666) vdump,mth
READ (41,*) dump
READ (41,*,IOSTAT=ios,ERR=666) vdump,vac_cov

!--- error message
666 IF (ios/=0) THEN
   WRITE (*,*) 'Read from Input.txt failed!',check
   STOP
ENDIF

!================================ Assignments ===================
loclen=loclen/sizefac/0.675
m_var=3+2*host_types

!================================ Assign values =========
SELECT CASE (input_type)
  CASE (1) ! Random distribution
    SELECT CASE (area_sel)
      CASE (1) ! Denmark
        sizefac =0.1
        xsize=4000
        ysize=3000
        n_farms(1)=17000
        n_farms(2)=0
        n_hosts(1)=1700000
        n_hosts(2)=0
        nmean=20
        nmax=48
        host_types=1
CASE (2) ! Lolland
  sizefac =0.1
  xsize =400
  ysize =400
  n_farms(1)=182
  n_farms(2)=0
  n_hosts(1)=8500
  n_hosts(2)=0
  nmean=15
  nmax=48
  host_types=1

CASE (3) ! Jutland
  sizefac =0.1
  xsize =400
  ysize =400
  n_farms(1)=1128
  n_farms(2)=0
  n_hosts(1)=117000
  n_hosts(2)=0
  nmean=25
  nmax=48
  host_types=1

CASE (4) ! Own def
  sizefac =sizefac

CASE (5) ! UK farms
  sizefac =0.1
  xsize =5182
  ysize =4880
  n_farms(1)=67020
  n_farms(2)=58337
  n_hosts(1)=6247964
  n_hosts(2)=21929519
  nmean=0
  nmax=48
  host_types=2

CASE DEFAULT
  WRITE (*,*) 'Selected input type for B1 (when A1=1) must be 1,2,3,4 or 5'
  STOP

END SELECT
ht=1; CALL distribute
IF (host_types==2) THEN
   ht=2; CALL distribute
ENDIF

CASE (2) ! CHR–Data

SELECT CASE (area_sel)
   CASE (1:3,5,8:13)
      CALL read_mark_blocks
      loclen=loclen/sizefac/0.675
   CASE DEFAULT
      WRITE (*,*) 'Selected input type for B1 (when A1=2) must be 1:3,5,8:13'
      STOP
   END SELECT
   CASE DEFAULT
      WRITE (*,*) 'Selected input type in A1 must be 1 or 2'
      STOP
   END SELECT
END SELECT

END SUBROUTINE

END MODULE
### INPUT FILE for BTV-SPREAD program ###

Author: Kaare Græsbøll (Graesboell) contact: kagr@imm.dtu.dk


Instructions:

---

A. SECTION:

A1: Choose Random distribution (1) or CHR-Data (2):

Value.A1...........: 2

---

B. SECTION

B1: Choose region: Denmark (1), Lolland (2), Jutland (3), own definition (4), UK farms (5), UK pasture sc 1,2,5 (8,9,10), DK-b sc 2,3,5 (11,12,13)

Value.B1...........: 12

---

If A1=1 and B1=4 Define:

- side length of each grid cell (km): REAL
  sizefac...........: 0.1

- Size of area (grids): INTEGER
  xsize............: 5182
...ysize..........: 4880

   number of cattle: INTEGER !uk 67020
...farms+fields...: 67020
...cattle.......: 6247964

   number of sheep: INTEGER !uk 58337, 19040196
...farms+fields...: 58337
...sheep.........: 21929519

.  Field distribution defined by binomial distribution: INTEGER
...mean.size.(np): 1
...max.size.(n): 48

---

C: SECTION

.  host types: (1) cattle (2) cattle + sheep: INTEGER (must be 2!)
.hosts_types.....: 2

.  percentage of hosts outdoors: INTEGER
.hosts_out.......: 0

.  stages of infection to mimic Gamma-dist for: INTEGER
.midges.........: 40
.cattle........: 5
.sheep..........: 7 !--- different because of dt

.  Recovery time for host (days): REAL
.cattle........: 20.6
.sheep.........: 16.4

.  width to length scale in wind spread cone: REAL
.width.to.length: 0.5

.  probability of transmission midge -> host: REAL
.probtrans_mh....: 0.9

.  probability of transmission host -> midge: REAL
.probtrans_hm....: 0.1

.  length of midge flight time (hours): REAL
flytime............: 1.5
.
  fraction of midges that fly with the wind: REAL
midgefly..........: 0.05
.
  fraction of midges that stay where no hosts: REAL
midgestay........: 0.05
.
  local length scale for M random walk (km 50% of midges reach): REAL
loclen............: 0.5
.
  number of midges per host (vectors per host): INTEGER
mth..............: 5000
.
  vaccination cover (fraction): REAL
vac_cov..........: 0.
MODULE host ! moves hosts

CONTAINS

A.7.1 host_move (2.7.1)

SUBROUTINE host_move !-- sub to move hosts

USE gdata
USE functions
IMPLICIT NONE

INTEGER :: nmove,h,tomove,htomove,m,l,i,j,k,tot_host,hid
REAL :: x(3), dist ,angle,y

DO ht=1,host_types

   nmove=binv(REAL(hmove(ht),KIND(1.0D0)),NINT(1.*hsick(ht)/hmovef(ht)))

   DO h=1,nmove

      !-- which host to move -- only sick gets to be moved
      CALL random_number(x(:))
      tomove=ceiling(x(1)*hsick(ht))

      IF (ht==1 .and. SUM(csick_loc(tomove,(h_var+2):(h_var+hstages(ht)+1)))==0) &
         & CYCLE
      IF (ht==2 .and. SUM(ssick_loc(tomove,(h_var+2):(h_var+hstages(ht)+1)))==0) &
         & CYCLE

      IF (ht==1) THEN
         i=csick_loc(tomove,1)
         j=csick_loc(tomove,2)
      ENDIF

      IF (ht==2) THEN
         i=ssick_loc(tomove,1)
         j=ssick_loc(tomove,2)
      ENDIF

   !-- deleted part from v964 check host move dist
! where to move based on 'decription of inputs dk FMD'
dist=(12.*(-log((1.000001-x(2))/1.2))**2.)/sizefac
angle=2.*pi*x(3)

IF (movingrestrictions) dist=MIN(dist,50./sizefac)

m=i+NINT(dist*sin(angle))
l=j+NINT(dist*cos(angle))

m=MAX(m,1);m=MIN(m,ysize)
l=MAX(l,1);l=MIN(l,xsize)
!write (27,* ) dist ,angle,m,i,l,j

k=1
i=m; j=l

kloop: DO WHILE (hosts(m,l,(2*ht-1))==0)

! if needed introduce randomness by modulo statement

m=MIN(i+k,ysize)
DO l=MAX(j-k,1),MIN(j+k,xsize)
   IF (hosts(m,l,(2*ht-1))/=0) EXIT kloop
ENDDO

m=MAX(i-k,1)
DO l=MAX(j-k,1),MIN(j+k,xsize)
   IF (hosts(m,l,(2*ht-1))/=0) EXIT kloop
ENDDO

l=MIN(j+k,xsize)
DO m=MAX(i-k,1),MIN(i+k,ysize)
   IF (hosts(m,l,(2*ht-1))/=0) EXIT kloop
ENDDO

l=MAX(j-k,1)
DO m=MAX(i-k,1),MIN(i+k,ysize)
   IF (hosts(m,l,(2*ht-1))/=0) EXIT kloop
ENDDO

k=k+1
ENDDO kloop

!− register the move
i=m; j=1
tot_host=hosts(i,j,1)
IF (host_types>1) tot_host=tot_host+hosts(i,j,3)

IF (hosts(i,j,2*ht)==0) THEN
  hsick(ht)=hsick(ht)+1
  IF (hsick(ht)>a_hsic(h)) CALL re_al_hsic

IF (ht==1) THEN
  csick_loc(hsick(ht),:,:) = 0
  csick_loc(hsick(ht),1:(h_var+1)) = &
  (/i,j,hosts(i,j,1+2*host_types),tot_host,&
  hosts(i,j,ht) /)
  IF (hosts(i,j,1+2*host_types)/=0) &
  & msick_loc(hosts(i,j,1+2*host_types),3)=hsick(ht)
ENDIF

IF (ht==2) THEN
  ssick_loc(hsick(ht),:,:) = 0
  ssick_loc(hsick(ht),1:(h_var+1)) = &
  (/i,j,hosts(i,j,1+2*host_types),tot_host,&
  hosts(i,j,ht+1) /)
  IF (hosts(i,j,1+2*host_types)/=0) &
  & msick_loc(hosts(i,j,1+2*host_types),5)=hsick(ht)
ENDIF

hid=hsick(ht)

hosts(i,j,2*ht)=hid
ELSE
  hid = hosts(i,j,2*ht)
ENDIF

IF (ht==1) THEN
  DO k=(h_var+2),(h_var+hstages(ht)+1)
    htomove=binv(REAL(hmovef(ht),KIND(1.0D0)),csick_loc(tomove,k))
    csick_loc (tomove,k)=csick_loc(tomove,k)−htomove
    csick_loc (tomove,h_var+1)=csick_loc(tomove,h_var+1)+htomove
csick_loc (hid,k)=csick_loc(hid,k)+htomove
csick_loc (hid,h_var+1)=csick_loc(hid,h_var+1)−htomove
ENDDO
!write (28,*) csick_loc (hsick(ht,:),’...’),hosts(i,j,:)
ENDIF

IF (ht==2) THEN
  DO k=(h_var+2),(h_var+hstages(ht)+1)
    htomove=binv(REAL(hmovef(ht),KIND(1.0D0)),ssick_loc(tomove,k))
    ssick_loc (tomove,k)=ssick_loc(tomove,k)−htomove
    ssick_loc (tomove,h_var+1)=ssick_loc(tomove,h_var+1)+htomove
    ssick_loc (hid,k)=ssick_loc(hid,k)+htomove
    ssick_loc (hid,h_var+1)=ssick_loc(hid,h_var+1)−htomove
  ENDDO
!write (28,*) ssick_loc (hsick(ht,:),’...’),hosts(i,j,:)
ENDIF

ENDDO
ENDDO

END SUBROUTINE host_move

END MODULE
A.8 temperatures.f90 (2.8)

MODULE temperatures !contains code related to temperature

CONTAINS

A.8.1 temp_var_h (2.8.1)

SUBROUTINE temp_var_h !--- determine parameters based on hourly temperature
                      !--- watch it if the mean temperatures goes above 27C

USE gdata
IMPLICIT NONE
REAL(mk) :: T,m_clim,T_clim,ntmin(max_id_temp),ntmax(max_id_temp)
REAL(mk) :: dt,ndt,th(24),heip(24),hbite(24),toff
INTEGER :: mit,dump,n
toff=2.

IF (day==1) THEN
  DO n=1,max_id_temp
    READ (21,'(i4,x,i5,2(x,f5.1),2(x,f5.2), x,f7.2) ') &
    & dump,dump,tmin(n),tmax(n)
  ENDDO
  tmin(:)=tmin(:)+toff
  tmax(:)=tmax(:)+toff
ENDIF

IF (day==365) REWIND (21)

DO n=1,max_id_temp

  READ (21,'(i4,x,i5,2(x,f5.1),2(x,f5.2), x,f7.2) ') dump,dump,ntmin(n),ntmax(n)
  ntmin(n)=ntmin(n)+toff
  ntmax(n)=ntmax(n)+toff
  dt = tmax(n)−tmin(n)
  ndt = tmax(n)−ntmin(n)
  T=(tmax(n)+tmin(n))/2.
  th(:)= (/ tmin(n) , 0.017*dt+tmin(n) , 0.067*dt+tmin(n) ,0.146*dt+tmin(n) , &
    & 0.250*dt+tmin(n) , 0.371*dt+tmin(n) , 0.500*dt+tmin(n) , &
    & 0.629*dt+tmin(n) , 0.750*dt+tmin(n) , 0.854*dt+tmin(n) , &
    & 1.000*dt+tmin(n) )
ENDDO

& 0.933*dt+tmin(n) , 0.983*dt+tmin(n) , tmax(n) , &
& 0.983*ndt+ntmin(n) , 0.933*ndt+ntmin(n) , 0.854*ndt+ntmin(n) , &
& 0.750*ndt+ntmin(n) , 0.629*ndt+ntmin(n) , 0.500*ndt+ntmin(n) , &
& 0.371*ndt+ntmin(n) , 0.250*ndt+ntmin(n) , 0.146*ndt+ntmin(n) , &
& 0.067*ndt+ntmin(n) , 0.017*ndt+ntmin(n) /)

!heip(:) = 0.0000125*th(:)*(th(:)−10.4) !-- old EIP
heip(:) = 0.018*(th(:)−13.4)/24.

!mrec(n) = MAX( SUM(heip, mask = th .gt. 10.4) , 0.01 ) !-- old EIP
mrec(n) = MAX( SUM(heip, mask = th .gt. 13.4) , 0.01 )

mmort(n) = MAX(1.−exp(−0.009_mk*exp(0.16_mk*T)) , 0.1)

hbite(:)= (0.0002*th(:)∗(th(:)−3.7)*41.9−th(:)**(0.37))/24.
biterate (n) = SUM(hbite, mask = th .gt. 3.7)

mboxes(n) = MAX( NINT(0.5/mrec(n)) , 3 )
mboxes(n) = MIN( NINT(0.5/mrec(n)) , 11 )
mbjr(n) = (1.*mstages)/(1.*mboxes(n))
mbji(n) = NINT(mbjr(n))

IF (.FALSE.) THEN !-- add microclimate temperatures to eip

T_clim=T+2.5
m_clim = MAX(0.01,0.0003_mk*T_clim∗(T_clim−10.4_mk))
IF (T_clim<0.) m_clim = 0.01

mboxes(n) = MAX( NINT(1./m_clim)−1 , 1 )
mbjr(n) = (1.*mstages)/(1.*mboxes(n))
mbji(n) = NINT(mbjr(n))

ENDIF

ENDDO

temperature=T
tmin=ntmin
tmax=ntmax

!--- Seasonal midges ---

IF (day>150 .and. day<310) THEN

mth=NINT(abs(sin(pi*(day−150.)/42.)*2950.*sin(pi*(day−150.)/160.)))+50
ELSE
  mth=0
ENDIF
! mth=700

END SUBROUTINE temp_var_h

!=================================

A.8.2 read_temp_id (2.8.2)

SUBROUTINE read_temp_id
USE gdata
IMPLICIT NONE

INTEGER :: icols,irows,n,mit

SELECT CASE (area_sel)

  CASE (1)
OPEN (57,FILE='/home/kagr/Desktop/Data/temperature/temp_indexr.dat', &
     & STATUS='old',ACTION='read')
max_id_temp=70

  CASE (4,5,8)
OPEN (57,FILE='/home/kagr/Desktop/Data/uk/temp_indexruk.dat', &
     & STATUS='old',ACTION='read')
max_id_temp=19

  CASE (9)
OPEN (57,FILE='/home/kagr/Desktop/Data/uk/temp_indexruk2.dat', &
     & STATUS='old',ACTION='read')
max_id_temp=19

  CASE (10)
OPEN (57,FILE='/home/kagr/Desktop/Data/uk/temp_indexruk5.dat', &
     & STATUS='old',ACTION='read')
max_id_temp=19

  CASE (11)
OPEN (57,FILE='/home/kagr/Desktop/Data/temperature/temp_indexr2.dat', &
     & STATUS='old',ACTION='read')
max_id_temp=70
CASE (12)
max_id_temp=70

CASE (13)
OPEN (57, FILE=’/home/kagr/Desktop/Data/temperature/temp_indexr5.dat’, & & STATUS=’old’, ACTION=’read’) 
max_id_temp=70

CASE DEFAULT 
PRINT *, ’No temperature index map available for the selected area’
STOP

END SELECT

icols=xsize
irows=ysize

ALLOCATE(id_temp(irows,icols))

DO n=1,icols
    READ (57,*) id_temp(:,n)
ENDDO

mit=max_id_temp

CLOSE (57)

ALLOCATE(mrec(mit), mmort(mit), biterate(mit), mboxes(mit), mbjr(mit), mbji(mit))
ALLOCATE(tmin(mit), tmax(mit))

END SUBROUTINE read_temp_id

END MODULE
A.9 makemap.f90 (2.9)

MODULE makemap ! Distributes cattle from raster files

CONTAINS

A.9.1 make_mark_blocks (2.9.1)

SUBROUTINE make_mark_blocks
USE gdata
USE functions
IMPLICIT NONE

INTEGER :: ndata,ndata2
INTEGER :: n,i
INTEGER :: idump,j,indhi,indlow, field
INTEGER :: cpumapp,oldchr, count,m
INTEGER :: ncols,nrows,nodat, val,k,unikbloknr
INTEGER :: trow,brow,lcol , rcol , location ,tkpa,tmkpa

INTEGER, ALLOCATABLE :: map(:,:),cpumap(:,:),cpu(:,:),chr2(:)
INTEGER, ALLOCATABLE :: chr(:,),cpu(:,),upc(:,),chk(:,),blokno(:,)
INTEGER, ALLOCATABLE :: pos2(:,),chk2(:,),blokid(:,),cblockid(:)

REAL :: xllc , yllc , cellsize ,rdump,dist,distc,ccpu,fwf
REAL :: totalarea ,cattdens,x,cutoff

REAL, ALLOCATABLE :: area(:,),graes(:,),kvaeg(:,),kvaeg2(:,)
REAL, ALLOCATABLE :: kpa(:,),totarea(:)

CHARACTER :: dump*14,temp*8,cdump*99
CHARACTER, ALLOCATABLE :: unikblok(:,)*9

SELECT CASE (area_sel)

!----  Denmark without bornholm

!----  shared chr permitted!

CASE (1)
   OPEN (1,FILE=’/home/kagr/Desktop/Data/fields/ubr_dk−b1.dat’,&
      STATUS=’old’,ACTION=’read’)

OPEN (11, FILE='/home/kagr/Desktop/Data/fields/check_dk−b1.dat',&
 & STATUS='old', ACTION='read')
OPEN (2, FILE='/home/kagr/Desktop/Data/fields/kvaegdata_dk−b1.dat',&
 & STATUS='old', ACTION='read')

ndata=100631; ndata2=21877 ! rescale=1
fwf=1./(16472./21877.) !– f wf resc 1

CASE (11)
OPEN (1, FILE='/home/kagr/Desktop/Data/fields/ubr_dk−b2.dat',&
 & STATUS='old', ACTION='read')
OPEN (11, FILE='/home/kagr/Desktop/Data/fields/check_dk−b2.dat',&
 & STATUS='old', ACTION='read')
OPEN (2, FILE='/home/kagr/Desktop/Data/fields/kvaegdata_dk−b2.dat',&
 & STATUS='old', ACTION='read')

ndata=89756; ndata2=21877 ! rescale=2
fwf=1./(16364./21877.) !– f wf resc 2

CASE (12)
OPEN (1, FILE='/home/kagr/Desktop/Data/fields/ubr_dk−b3.dat',&
 & STATUS='old', ACTION='read')
OPEN (11, FILE='/home/kagr/Desktop/Data/fields/check_dk−b3.dat',&
 & STATUS='old', ACTION='read')
OPEN (2, FILE='/home/kagr/Desktop/Data/fields/kvaegdata_dk−b3.dat',&
 & STATUS='old', ACTION='read')

ndata=81492; ndata2=21877 ! rescale=3
fwf=1./(16281./21877.) !– f wf resc 3

CASE (13)
OPEN (1, FILE='/home/kagr/Desktop/Data/fields/ubr_dk−b5t.dat',&
 & STATUS='old', ACTION='read')
OPEN (11, FILE='/home/kagr/Desktop/Data/fields/check_dk−b5t.dat',&
 & STATUS='old', ACTION='read')
OPEN (2, FILE='/home/kagr/Desktop/Data/fields/kvaegdata_dk−b5t.dat',&
 & STATUS='old', ACTION='read')

ndata=64444; ndata2=21877 ! rescale=5
ndata=48043; ndata2=21877 ! rescale=5t
fwf=1./(11780./21877.) !– f wf resc 5
fwf=1./(14790./21877.) !-- fwf resc 5t

CASE DEFAULT
    PRINT *, 'ERROR IN CHOSING AREA_SEL!!!'
    STOP
END SELECT

IF (allocated(cpu) .eqv. .false.) THEN
    ALLOCATE(cpu(ndata,3),chr2(ndata2))
    ALLOCATE(chr(ndata),cpun(ndata),upc(ndata),chk(ndata))
    ALLOCATE(pos2(ndata2,2),chk2(ndata2),blokno(ndata))
    ALLOCATE(area(ndata),graes(ndata),kvaeg(ndata),kvaeg2(ndata2))
    ALLOCATE(kpa(ndata),totarea(ndata),blokid(93577,13),cblokid(93577))
    ALLOCATE(unikblok(ndata))
ENDIF

IF (allocated(csick) .eqv. .false.) THEN
    ALLOCATE(csick(a_hsick(1),h_var+hstages(1)+2))
    ALLOCATE(msick(a_msick,m_var+mstages+2))
    ALLOCATE(csick_com(a_hsick_c(1),4))
ENDIF
    csick=0
    msick=0
    csick_com=0

IF (allocated(ssick) .eqv. .false. .and. host_types>1) THEN
    ALLOCATE(ssick(a_hsick(2),2+h_var+hstages(2)))
    ALLOCATE(ssick_com(a_hsick_c(2),4))
ENDIF
    ssick=0
    ssick_com=0

IF (allocated(fly) .eqv. .false.) THEN
    ALLOCATE(fly(a_fly,3))
ENDIF

!-- READ FILES ---------------------

READ (11,*) cdump

blokid=0; cblokid=0
count=0
DO n=1,ndata
  !READ (11,'(i6,x,i5,x,f12.1,x,f8.3,x,i4,x,i4,') ) &
  READ (11,'(i6,x,i5,x,f12.3,x,f8.3,x,i4,x,i4)') &
    & chr(n),blokno(n),area(n),graes(n),upc(n),cpun(n)
  IF (blokno(n)==0) CYCLE
  cblokid(blokno(n))=cblokid(blokno(n))+1
  blokid(blokno(n),cblokid(blokno(n)))=n
ENDDO

CLOSE (11)

!PRINT *,MAXVAL(blokno(:)),MAXVAL(cpun(:)),count
!stop

READ (2,* ) cdump

DO n=1,ndata2
  READ (2,* ) cdump,pos2(n,1),pos2(n,2),chr2(n),kvaeg2(n)
ENDDO

CLOSE (2)

!PRINT *,SUM(kvaeg2(:))

!--- READ RASTER FILE -----------------------

READ (1,* ) dump,ncols
READ (1,* ) dump,nrows
READ (1,* ) dump,xllc
READ (1,* ) dump,yllc
READ (1,* ) dump,cellsize
READ (1,* ) dump,nodat_val

!PRINT *,ncols,nrows,nodat_val,cellsize

ALLOCATE(map(ncols,nrows))
ALLOCATE(cpumap(ncols,nrows))
cpumap=0
IF (allocated(hosts) .eqv. . false .) THEN
  ALLOCATE(hosts(ysize,xsize,1+2*host_types))
ENDIF
hosts=0

DO i=1,nrows
  READ (1,*) map(:,i)
  !PRINT *,i
ENDDO

CLOSE (1)

DO n=1,ndata
  unikbloknr=blokno(n)
  
tmkpa=MAX(NINT(2.*kpa(n)),1)
tkpa=binv(REAL((kpa(n)/REAL(tmkpa)),KIND(1.0D0)),tmkpa)
  cpu(n,:)=(/ chr(n) , unikbloknr , tkpa /) !==== NB ====
ENDDO

!---- READ some from RASTER FILE ------------------------
READ (1,*) dump,ncols
READ (1,*) dump,nrows
READ (1,*) dump,xllc
READ (1,*) dump,yllc
READ (1,*) dump,cellsize
READ (1,*) dump,nodat_val

REWIND (1)

!-----------------------------
CALL seeed

!---- Randomly assign which farms have cattle on grass
  i=1
kvaeg=0.
chk2=0
oldchr=0

DO n=1ndata
  DO WHILE (chr(n)>=chr2(i))
    IF (chr(n)==chr2(i)) THEN
      IF (chr(n)/=oldchr) THEN
        CALL random_number(x)
        IF (x < fwf*hosts_outside(1)) THEN ! fwf = 1/farms with fields
          kvaeg(n)=kvaeg2(i)
          chk2(i)=1
        ELSE
          kvaeg(n)=0.
        ENDIF
      ELSE
        kvaeg(n)=kvaeg2(i)
      ENDIF
    ENDIF
    i=i+1
  ENDDO
  i=i−1
  oldchr=chr(n)
ENDDO

!print *,sum(chk2(:))
!stop

!---- Distribute cattle onto grass fields, prioritize fields with
!---- highest grass percentage, distribute with max 50 cows/ha.
cutoff=90.0000 !--- less than cutoff graes => no cows on this field
totarea=0.
kpa=0.
i=0
chk=0

DO n=1,ndata

IF (i==0) THEN
  i=upc(n)
  j=upc(n)
  indlow=n-(upc(n)-i)
  indhi=n+i-1
  cutoff=90.0000
  totalarea=0.1
  !do while (kvaeg(n)/totalarea > 50./10000. .and. j>0)
  DO WHILE (kvaeg(n)/totalarea > 50./(cellsize**2) .and. j>0)

    CALL random_number(x)

    field =NINT(x*(indhi-indlow))+indlow

    IF (graes(field) > cutoff .and. chk(field)==0) THEN
      totalarea=totalarea+area(field)/REAL(cpun(field))
      chk(field)=1
      ENDF

  j=j-1

  IF (j==0) THEN
    cutoff=cutoff-10.
    IF (cutoff > 1.) j=upc(n)
    ENDF

ENDDO

ENDIF

IF (chk(n)==1) THEN

  totarea(n)=totalarea
  kpa(n)=kvaeg(n)/totalarea*cellsize**2.

ENDIF

i=i-1
! WRITE (3,'(i6,x,a9,x,f12.3,x,f8.3,x,i4,x,i4,x,f8.3,x,f15.3,x,f9.4)') chr(n),unikblok(n),area(n),graes(n),upc(n),cpun(n),kvaeg(n),totarea(n),
! & kpa(n)
ENDDO
!stop

--- Assign Cattle data ---------------

lcol=1; rcol=ncols
brow=1; trow=nrows

ysize=(rcol-lcol+1)
xsize=(trow-brow+1)

sizefac = cellsize /1000.

cpumap=0
cpumapp=0
!k=20
k=0
count=ndata2-SUM(chk2(:))

DO n=1,ndata2
  IF (chk2(n)==0) THEN
    IF (k>0 .and. count>1) THEN
      cpumapp=cpumapp+kvaeg2(n)
k=k-1
      count=count-1
    CYCLE
    ENDIF
  cpumapp=cpumapp+kvaeg2(n)

  j=FLOOR(REAL(pos2(n,1)-xllc)/cellsize)+1
  i=FLOOR(REAL(pos2(n,2)-yllc)/cellsize)+1
  cpumap(j,i)=cpumapp
  !k=binv(REAL(0.5,KIND(1.0D0)),40)
k=0
  count=count-1
cpumapp=0

ENDIF

ENDDO

DO i=brow,trow
  DO j=lcol,rcol

    cpumapp=0

    IF (map(j,i) /= nodat_val) THEN

      n=cblokid(map(j,i))

      IF (n /= 0) THEN
        DO m=1,n
          field = cblokid(map(j,i), m)
          cpumap(j,i) = cpumap(j,i) + cpu(field,3)/REAL(cpun(field))
        ENDDO
      ENDIF

    ENDIF

  ! cattle (j , i,1)=cpumap(j,i)
  !WRITE (22,'(i5,x)',advance='no') cpumap(j,i)

  ENDDO

  !WRITE (22,'(a)')
  ENDDO

  hosts(:,:,1)= cpumap(:,:,)

END SUBROUTINE

END MODULE makemap
A.10 viraemia.f90 (2.10)

MODULE viraemia
!--- This module handles how cattle and midges move through infection stages.
CONTAINS

A.10.1 host_viraemia (2.10.1)

SUBROUTINE host_viraemia

USE gdata
USE functions

IMPLICIT NONE
INTEGER :: i,j,k,h,dc

ht=1
DO h=1,hsick(ht)
    IF (SUM(csick_loc(h,h_var+2:h_var+hstages(ht)+1))==0) CYCLE
    DO k=h_var+hstages(ht)+1,h_var+2,-1
        IF (csick_loc(h,k)==0) CYCLE
        dc=binv(REAL((hrec(ht)*hstages(ht)),KIND(1.0D0)),csick_loc(h,k))
        csick_loc (h,k)=csick_loc(h,k)−dc
        csick_loc (h,k+1)=csick_loc(h,k+1)+dc
    ENDDO
    !write (28,*) day,h,SUM(csick_loc(h,5:10)), csick_loc (h,1:2) ! testing.dat
ENDDO

IF (host_types>1) THEN
    ht=2
    DO h=1,hsick(ht)
        IF (SUM(ssick_loc(h,h_var+2:h_var+hstages(ht)+1))==0) CYCLE
    ENDDO
DO  k=h.var+hstages(ht)+1,h.var+2,−1

    IF ( ssick_loc (h,k)==0) CYCLE
    dc=binv(REAL((hrec(ht)*hstages(ht)),KIND(1.0D0)),ssick_loc(h,k))
    ssick_loc (h,k)=ssick_loc (h,k)−dc
    ssick_loc (h,k+1)=ssick_loc(h,k+1)+dc

ENDDO

ENDDO
ENDIF

END SUBROUTINE host_viraemia

!=================================

A.10.2 midge_viraemia (2.10.2)

SUBROUTINE midge_viraemia

USE gdata
USE functions

IMPLICIT NONE
INTEGER :: i,j,k,h,ddm,jump,jt,T_id,cand,smbji,smboxes,T_ido,m
REAL :: x,jc,y(2),smbjr,smrec,smmort

T_ido=0

DO h=1,msick

    IF (SUM(msick_loc(h,m_var+2:))==0) THEN
        hosts(msick_loc(h,1),msick_loc(h,2),1+2*host_types)=0
        IF (msick_loc(h,3)/=0) csick_loc(msick_loc(h,3),3)=0
        IF (host_types>1 .and. msick_loc(h,5)/=0) ssick_loc(msick_loc(h,5),3)=0
        msick_loc(h,:)=0
        CYCLE
    ENDIF

END
T_id=id_temp(msick_loc(h,1),msick_loc(h,2)) !--maybe faster if T_id in msick

IF (T_id/=T_ido) THEN
  smbjr=mbjr(T_id)
  smbji=mbji(T_id)
  smrec=mrec(T_id)
  smmort=mmort(T_id)
  smboxes=mboxes(T_id)
  T_ido=T_id
ENDIF

IF (msick_loc(h,mstages+m_var+2)>0) THEN
  ddm=binv(REAL(smmort,KIND(1.0D0)),msick_loc(h,mstages+m_var+2))
  msick_loc(h,mstages+m_var+2)=msick_loc(h,mstages+m_var+2)−ddm
  !msick_loc(h,m_var+1)=msick_loc(h,m_var+1)+ddm
  !--maintains constant amount of midges
ENDIF

jc=0.
CALL random_number(x)
IF (x<abs(smbjr−1.*smbji)) jc=1.
jc=sign(jc,smbjr−1.*smbji)
jump=smbji+jc

DO k = (mstages+m_var+1) , (m_var+2) ,−1
  cand=msick_loc(h,k)
  SELECT CASE (cand)
    CASE (0)
      CYCLE
    CASE (1:5)
      dm=0; ddm=0
      DO m=1,cand
        CALL random_number(y)
        IF (y(1)<smmort) THEN
          ddm=ddm+1
        ELSE IF (y(2)<smrec*smboxes) THEN
          dm=dm+1
        ENDIF
ENDDO

dm=MIN(dm,cand−ddm)

CASE DEFAULT

ddm=binv(REAL(smmort,KIND(1.0D0)),cand)
dm=binv(REAL(smrec*smboxes,KIND(1.0D0)),cand−ddm)

END SELECT

jt=min(k+jump,m_var+mstages+2)

msick_loc(h,k)=cand−dm−ddm
msick_loc(h,jt)=msick_loc(h,jt)+dm
!msick_loc(h,m_var+1)=msick_loc(h,m_var+1)+ddm
!−maintains constant amount of midges

ENDDO

IF (SUM(msick_loc(h,m_var+2:))==0) THEN
hosts(msick_loc(h,1),msick_loc(h,2),1+2*host_types)=0
IF (msick_loc(h,3)==0) csick_loc(msick_loc(h,3),3)=0
IF (host_types>1 .and. msick_loc(h,5)/=0) ssick_loc(msick_loc(h,5),3)=0
msick_loc(h,:)=0
ENDIF

ENDDO

END SUBROUTINE

END MODULE
A.11 windy.f90 (2.11)

MODULE windy

CONTAINS

A.11.1 wind_data (2.11.1)

SUBROUTINE wind_data
USE gdata
IMPLICIT NONE
REAL(mk) :: x,y

CALL random_number(x);CALL random_number(y)
!x=0.5_mk;y=0.5_mk

wind_a = 2._mk*pi*x
!wind_a = 2.*pi*x + 0.5*sin(2.*pi*x + 3.*pi/4.) − 0.3536 ! wind uk
!wind_a = 2.*pi*x + 0.75*sin(2.*pi*x + 3.*pi/4.) − 0.53
wind_s = 5._mk*y

END SUBROUTINE

END MODULE
A.12 midge.f90 (2.12)

MODULE midge
!
! Notice: to save calc.time healthy midges do not fly!
!
CONTAINS

A.12.1 midge_local (2.12.1)

SUBROUTINE midge_local
USE gdata
USE functions
IMPLICIT NONE
INTEGER :: i,k,j,l,m,h,BM
CALL midgesflying,hh,midgesfly
REAL(mk) :: x(2),y,theta, Rnorm,length,local_midgefly

! fly = 0
BM_call=0
fly_c =0
ht=host_types

DO h=1,msick
  IF (SUM(msick_loc(h,m_var+2:)) == 0) THEN
    !msick_loc(h,(m_var+1):)=0
    CYCLE
  ENDIF
  i=msick_loc(h,1)
  j=msick_loc(h,2)
  IF (msick_loc(h,3+2*host_types) == 0) THEN
    local_midgefly=MAX(1._mk−midgestay,0._mk)
  ELSE
    local_midgefly=midgefly
  ENDIF
  DO k=m_var+2,m_var+mstages+2
    IF (msick_loc(h,k)==0) CYCLE
    midgesflying=binv(REAL(local_midgefly,KIND(1.0D0)),msick_loc(h,k))
    midgesfly=0
DO hh=1,midgesflying

!−find fly coordinates in cylindrical coord. u. BOX−MULLER

IF (BM_call==0) THEN
CALL RANDOM_NUMBER(x)
theta = 2.∗pi ∗ x(1)
Rnorm = sqrt(−2.∗log(x(2)+10.∗(−16.)))

length = ABS( Rnorm∗sin(theta) ) ∗ loclen
BM_call=1
ELSE
length = ABS( Rnorm∗cos(theta) ) ∗ loclen
BM_call=0
ENDIF

CALL RANDOM_NUMBER(y)
y = 2.∗pi ∗ y

m=NINT( length∗ sin( y ) ) + i
l=NINT( length∗ cos( y ) ) + j

!−−− midges only fly to the edge of the simulation box
!m=MIN(m,ysize); m=MAX(1,m)
!l=MIN(l,xsize); l=MAX(1,l)

IF (m/=i .or. l/=j) THEN
IF (m>1 .and. m<ysize .and. l>1 .and. l<xsize) THEN
IF (day<flyminit) THEN
fly_c=fly_c+1
IF (fly_c>=a_fly) CALL re_al_fly
fly ( fly_c :)=(/ m,l,k /)
ELSE
flym(m,l,k−(m_var+1))=flym(m,l,k−(m_var+1))+1
ENDIF
ENDIF
ENDIF
midgesfly=midgesfly+1 !−−− don’t fly
ENDIF

!midgesfly=midgesfly+1 !−−− disappear over the edge
ENDIF

ENDDO
msick_loc(h,k)=msick_loc(h,k)−midgesfly

ENDDO

! if ((SUM(midges(i,j,2:))−SUM(fly_local(i,j,:)))==0) write (29,*) &
! Σ SUM(midges(i,j,2:mstages+2)),SUM(fly_local(i,j,:)),h,msick

END SUBROUTINE midge_local

!=================================

A.12.2 midge_wind (2.12.2)

SUBROUTINE midge_wind
USE gdata
USE functions
USE windy
IMPLICIT NONE
INTEGER :: i,k,j,l,m,h,midgesflying,hh,count,midgesfly
REAL (mk) :: x(2),angle,length,lesc,theta,Rnorm,wisc,windcut,wind_midgefly,y

fly =0
fly_c =0

! 0.90 midges is carried within 45 deg.
wisc=(pi/4_mk) / 1.645_mk

! if 0.90 midges fly within 1 square then cycle
windcut=(sizefac*1.645)/(flytime*3.6_mk)

CALL wind_data !--- All msick_loc should be updated in here.

DO h=1,msick

IF (SUM(msick_loc(h,(m_var+2):))==0) THEN
  CYCLE
ENDIF

IF (wind_s<windcut) CYCLE

! 0.90 midges is carried within lesc
lesc=flytime*3.6_mk*wind_s/sizefac/1.645_mk
\begin{verbatim}

i=msick_loc(h,1)
j=msick_loc(h,2)

IF (msick_loc(h,3+2*host_types) == 0) THEN
   wind_midgefly=MAX(1._mk−midgestay,0._mk)
ELSE
   wind_midgefly=midgefly/5.
ENDIF

DO k= (m_var+2) , mstages+m_var+2
   IF (msick_loc(h,k)==0) CYCLE
   midgesflying=binv(REAL(wind_midgefly,KIND(1.0D0)),msick_loc(h,k))
   midgesfly=0
END IF

DO hh=1,midgesflying
   !−find fly coordinates in cylindrical coord. BOX−MULLER

   CALL RANDOM_NUMBER(x)
   theta = 2._mk * pi * x(1)
   Rnorm = sqrt(−2._mk * log(x(2)+10.**(-16.)))
   !log−term might cause Arithmetic error

   angle = Rnorm*sin(theta) * wisc + wind_a
   length = ABS( Rnorm*cos(theta) ) * lesc

   m=NINT( length* sin( angle ) ) + i
   l=NINT( length* cos( angle ) ) + j

   !−−−−−−−−−−−−−−−−− midges only fly to the edge of the simulation box
   !m=MIN(m,ysize); m=MAX(1,m)
   !l=MIN(l,xsize); l=MAX(1,l)

   IF (m/=i .or. l/=j) THEN
   IF (m>1 .and. m<=ysize .and. l>1 .and. l<=xsize) THEN

   IF (day<flyminit) THEN
      fly_c=fly_c+1
      IF (fly_c>=a_fly) CALL re_al_fly
      fly ( fly_c,;)=(/ m,l,k /)
   ELSE
      flym(m,l,k−(m_var+1))=flym(m,l,k−(m_var+1))+1

\end{verbatim}
ENDIF
midgesfly=midgesfly+1 ! or don’t fly
ENDIF
!midgesfly=midgesfly+1 ! disappear over the edge
ENDIF

ENDDO

msick_loc(h,k)=msick_loc(h,k)−midgesfly

ENDDO

! if ((SUM(midges(i,j,2:))−SUM(fly_wind(i,j,:)))==0) write (29,*) &
! & SUM(midges(i,j,2:)),SUM(fly_wind(i,j,:))

ENDDO

END SUBROUTINE midge_wind

END MODULE
Appendix A

A.13  bite.f90 (2.13)

MODULE bite

CONTAINS

A.13.1  inf_hosts (2.13.1)

SUBROUTINE inf_hosts

USE gdata
USE functions

IMPLICIT NONE
INTEGER :: i,j,k,hid,h,susc,tot_host,T_id,vac_d
REAL(mk) :: vac

DO h=1,msick

  IF (msick_loc(h,mstages+2+m_var)==0) CYCLE

  i=msick_loc(h,1)
  j=msick_loc(h,2)

  DO ht=1,host_types

    IF (msick_loc(h,2+2*ht)==0) CYCLE

    IF (msick_loc(h,1+2*ht)>0) THEN
      IF (ht==1) susc=csick_loc(msick_loc(h,3),h_var+1)
      IF (ht==2) susc=ssick_loc(msick_loc(h,5),h_var+1)
    ELSE
      susc=msick_loc(h,2+2*ht)
    ENDIF

    tot_host=msick_loc(h,3+2*host_types)
    T_id=id_temp(i,j)
    !T_id=1

    ! if (vac_cov>0.1) then
    ! vac_d= vac_id(cphid(i,j))
    ! vac=(1.−real(day−vac_d)/tfp(ht)*efficacy)
    ! if (day>=(vac_d+tfp(ht))) vac=(1.−efficacy)
! if (day<vac_d) vac=1.
! endif
vac=1.-vac_cov*efficacy

k=binv(REAL(biterate(T_id)*probtrans_mh(ht)*vac* & & msick_loc(h,mstages+2+m_var)/(1.*tot_host),KIND(1.0D0)),susc) IF (k>0) THEN IF (msick_loc(h,1+2*ht)==0) THEN hskick(ht)=hskick(ht)+1

IF (hskick(ht)>=a_hskick(ht)) CALL re_al_hskick

IF (ht==1) THEN
csick_loc (hskick(ht ),1:( h_var+1)) = & & (/i, j ,h,tot_host ,hosts(i , j ,ht) /)
ENDIF

IF (ht==2) THEN
ssick_loc (hskick(ht ),1:( h_var+1)) = & & (/i, j ,h,tot_host ,hosts(i , j ,ht+1) /)
ENDIF

hid=hskick(ht)

msick_loc(h,1+2*ht) = hid
hosts(i,j,2*ht)=hid
ELSE
hid = msick_loc(h,1+2*ht)
ENDIF

IF (ht==1) THEN
csick_loc (hid,h_var+1)=csick_loc(hid,h_var+1)−k
   csick_loc (hid,h_var+2)=csick_loc(hid,h_var+2)+k
ENDIF

IF (ht==2) THEN
ssick_loc (hid,h_var+1)=ssick_loc(hid,h_var+1)−k
   ssick_loc (hid,h_var+2)=ssick_loc(hid,h_var+2)+k
ENDIF

!write (27,* ) day,ht,k, tot_host ,msick_loc(h,mstages+2+m_var),i,j
ENDIF

ENDDO
ENDDO

END SUBROUTINE inf_hosts

!=====================================================================

A.13.2 inf_midges (2.13.2)

SUBROUTINE inf_midges

USE gdata
USE functions

IMPLICIT NONE
INTEGER :: i,j,k,h,tot,h,mid,T
REAL(mk) :: inf
h,vac,x

!== CATTLE ==
ht=1
DO h=1,hsick(ht)
i=csick_loc(h,1)
j=csick_loc(h,2)
tot_h=SUM(csick_loc(h,(h_var+1):))

IF (SUM(csick_loc(h,(h_var+2):h_var+hstages(ht)+1))=0) CYCLE

inf_h=REAL(SUM(csick_loc(h,(h_var+2):h_var+hstages(ht)+1)),mk)
inf_h=0.50*mk*csick_loc(h,h_var+2)+&
  2.00*mk*csick_loc(h,h_var+3)+&
  2.00*mk*csick_loc(h,h_var+4)+&
  0.50*mk*csick_loc(h,h_var+5)

T_id=id_temp(i,j)

! if (vac_cov>0.1) then
!vac_d=vac_id(cphid(i,j))
!vac=(1.-real(day-vac_d)/tfp(ht)*efficacy)
! if (day>=(vac_d+tfp(ht))) vac=(1.-efficacy)
! if (day<vac_d) vac=1.
!endif
vac=1.-vac_cov*efficacy

mid=csick_loc(h,3)

! --- assign vector to host ratio ---

IF (varmth) THEN
  IF (mid==0) THEN
    CALL random_number(x)
    lmth=NINT(x*mth)
  ELSE
    lmth=msick_loc(mid,m_var+1)
  ENDIF
ELSE
  lmth=mth
ENDIF

IF (varmthdk) THEN
  CALL random_number(x)
  lmth=NINT(x*mth)
ENDIF

! ---

k=binv(REAL(biterate(T_id)*probrans_hm(ht)*vac*inf_h,KIND(1.0D0)),lmth)

IF (k>0 .and. mid==0 ) THEN
  msick=msick+1
  IF (msick>=a_msick) CALL re_al_msick
  msick_loc(msick,:)=0
  msick_loc(msick,1:4)=(/i,j,h,tot_h/)
  msick_loc(msick,3+2*host_types)=hosts(i,j,1)
  IF (host_types>1) msick_loc(msick,3+2*host_types)=&
    & msick_loc(msick,3+2*host_types)+hosts(i,j,3)
  msick_loc(msick,m_var+1)=lmth
  mid=msick
  csick_loc (h,3)=mid !err count!
  hosts(i ,j,1+2*host_types)=mid
ENDIF

!write (28,*) ht,k, inf_h, biterate (T_id),temperature
msick_loc(mid,m-var+2)=msick_loc(mid,m-var+2)+k
msick_loc(mid,m-var+1)=msick_loc(mid,m-var+1)−k

ENDDO

IF (host_types>1) THEN
    !== SHEEP ==
    ht=2
    DO h=1,hsick(ht)
        i=ssick_loc(h,1)
        j=ssick_loc(h,2)
        tot_h=SUM(ssick_loc(h,(h-var+1):))
    IF (SUM(ssick_loc(h,(h-var+2):(h-var+hstages(ht)+1)))==0) CYCLE
    IF (oo_transkernel) THEN
        hsick_com(ht)=hsick_com(ht)+1
        IF (hsick_com(ht)>=a_hsick_c(ht)) CALL re_al_hsick_com
        ssick_loc_com(hsick_com(ht),:)=(/i,j,day,h/)
    ENDIF
    ! inf_h=REAL(SUM(ssick_loc(h,h-var+2:h-var+hstages(ht)+1)),mk)
    inf_h= 0.50*mk*SUM(ssick_loc(h,h-var+2 :h-var+3 ))+&
         & 2.00*mk*SUM(ssick_loc(h,h-var+4 :h-var+6))+&
         & 0.50*mk*SUM(ssick_loc(h,h-var+7:h-var+8))
    T_id=id_temp(i,j)

    ! if (vac_cov>0.1) then
    ! vac_d=vac_id(cphid(i,j))
    ! vac=(1.-real(day−vac_d)/tfp(ht)*efficacy)
    ! if (day>=(vac_d+tfp(ht))) vac=(1.-efficacy)
    ! if (day<vac_d) vac=1.
    ! endif
    vac=1.-vac_cov*efficacy

    mid=ssick_loc(h,3)

    ! ---- assign vector to host ratio ----
    IF (varmth) THEN
        IF (mid==0) THEN
CALL random_number(x)
    lmth=NINT(x*mth)
ELSE
    lmth=msick_loc(mid,m_var+1)
    IF (lmth==0) THEN
        CALL random_number(x)
        lmth=NINT(x*mth)
        msick_loc(mid,m_var+1)=lmth
    ENDIF
ENDIF
ELSE
    lmth=mth
ENDIF

IF (varmthdk) THEN
    CALL random_number(x)
    lmth=NINT(x*mth)
ENDIF
!

! k = \text{binv}(REAL(biterate(T\_id)*protrans\_hm(ht)*vac\_inf\_h,KIND(1.0D0)),lmth)

IF (k>0 .and. mid==0) THEN
    msick=msick+1
    IF (msick>=a\_msick) CALL re\_al\_msick
    msick_loc(msick,:)=0
    msick_loc(msick,1:2)=(/i,j/)
    msick_loc(msick,5:6)=(/h,tot\_h/)
    msick_loc(msick,3+2*host\_types)=hosts(i,j,1)+hosts(i,j,3)
    msick_loc(mid,m_var+1)=lmth
    mid=msick
    ssick_loc(h,3)=mid !err count!
    hosts(i,j,1+2*host\_types)=mid
ENDIF

msick_loc(mid,m_var+2)=msick_loc(mid,m_var+2)+k
!msick\_loc(mid,m\_var+1)=msick\_loc(mid,m\_var+1)−k
ENDDO
ENDIF
END SUBROUTINE inf\_midges
END MODULE
A.14 errorhandl.f90 (2.14)

MODULE errorhandl

CONTAINS

A.14.1 findmidges (2.14.1)

SUBROUTINE findmidges ! test if midge ID is correctly handled in the hosts

USE gdata

IMPLICIT NONE

INTEGER :: i,j,k,mlist(msick)

IF (msick==0) RETURN

! if (2<1) then
mlist=0
DO i=1,ysize
  DO j=1,xsize
    IF (hosts(i,j,1+2*host_types)>0) THEN
      mlist(hosts(i,j,1+2*host_types))=mlist(hosts(i,j,1+2*host_types))+1
      IF (i/=msick_loc(hosts(i,j,1+2*host_types),1)) THEN
        WRITE (27,*) day,i,j,msick_loc(hosts(i,j,1+2*host_types),1:2),&
        & hosts(i,j,1+2*host_types),msick
        errorcounter=errorcounter+1
      ENDIF
    ENDIF
  ENDDO
ENDDO

! endif
! if (2<1) then
DO k=1,msick
  IF (k/=hosts(msick_loc(k,1),msick_loc(k,2),1+2*host_types)) THEN
    ! endif
  ENDIF
ENDDO
ENDDO
WRITE (28,*) day,k,hosts(msick_loc(k,1),msick_loc(k,2),1+2*host_types)
errorcounter=errorcounter+1
ENDIF

IF (mlist(k)>1) THEN
  WRITE (28,*) day,k,mplist(k)
  errorcounter=errorcounter+1
ENDIF

ENDDO

END SUBROUTINE

!-----------------------------------------------------------

A.14.2 findhosts (2.14.2)

SUBROUTINE findhosts !-- check if hosts have been correctly registered

USE gdata

IMPLICIT NONE

INTEGER :: n,i,j,k

DO n=1,msick
  i=msick_loc(n,1)
  j=msick_loc(n,2)
  IF (i==0 .or. j==0) CYCLE

  IF (msick_loc(n,m_var)/= (hosts(i,j,1)+hosts(i,j,3))) THEN
    WRITE (27,*) n,i,j,msick_loc(n,m_var),hosts(i,j,1),hosts(i,j,3)
    errorcounter=errorcounter+1
  ENDIF

  IF (msick_loc(n,3)>0) THEN
    IF (n/=csick_loc(msick_loc(n,3),3)) THEN
      WRITE (27,*) '1',n,csick_loc(msick_loc(n,3),3),&
& hosts(i, j,1+2*host_types),msick_loc(n,3), hsick(1),&
& i, j, csick_loc (msick_loc(n,3),1:2)
errorcounter=errorcounter+1
ENDIF
ENDIF

IF (host_types>1 .and. msick_loc(n,5)>0) THEN
  IF (n/=ssick_loc(msick_loc(n,5),3)) THEN
    WRITE (27,*) '2',n,ssick_loc(msick_loc(n,5),3),&
    & hosts(i, j,1+2*host_types),msick_loc(n,5), hsick(2),&
    & i, j, ssick_loc (msick_loc(n,5),1:2)
    errorcounter=errorcounter+1
  ENDIF
ENDIF
ENDIF

ENDDO

DO n=1,hsick(1)

  i=csick_loc(n,1)
  j=csick_loc(n,2)
  IF (i==0 .or. j==0) CYCLE

  IF ( csick_loc (n,4)/= (hosts(i, j,1)+hosts(i, j ,3))) THEN
    WRITE (27,*) n,i,j,csick_loc(n,4),hosts(i, j ,1), hosts(i, j ,3)
    errorcounter=errorcounter+1
  ENDIF

  IF (SUM(csick_loc(n,5:))/= hosts(i,j ,1)) THEN
    WRITE (27,*) n,i,j,csick_loc(n,5),hosts(i, j ,1)
    errorcounter=errorcounter+1
  ENDIF

  IF ( csick_loc (n,3)>0) THEN
    IF (i/=msick_loc(csick_loc(n,3),1)) THEN
      WRITE (27,*) &
      & '1',n,i,msick_loc( csick_loc (n,3),1)
      errorcounter=errorcounter+1
    ENDIF
  ENDIF

ENDDO

DO n=1,hsick(2)
i = ssick_loc(n,1)
j = ssick_loc(n,2)
IF (i == 0 .or. j == 0) CYCLE

IF (ssick_loc(n,4) /= (hosts(i,j,1) + hosts(i,j,3))) THEN
WRITE (28,*) n,i,j,ssick_loc(n,4),hosts(i,j,1), hosts(i,j,3)
errorcounter = errorcounter + 1
ENDIF

IF (SUM(ssick_loc(n,5:)) /= hosts(i,j,3)) THEN
WRITE (28,*) n,i,j,ssick_loc(n,5),hosts(i,j,3)
errorcounter = errorcounter + 1
ENDIF

ENDIF
ENDDO

IF (.true.) THEN

DO n = 1, hsick(1)
    i = csick_loc(n,1)
j = csick_loc(n,2)

    DO k = 1, hsick(2)
        IF (i == ssick_loc(k,1) .and. j == ssick_loc(k,2) .and. &
            csick_loc(n,3) /= ssick_loc(k,3)) THEN
            WRITE (28,*) day,n,k,csick_loc(n,1:3),ssick_loc(k,1:3)
            errorcounter = errorcounter + 1
        ENDIF
    ENDDO

ENDDO

ENDDO
ENDIF

flush (27); flush (28)

END SUBROUTINE

END MODULE
## A.15 Subroutine table

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>dk.f90 (2.1)</td>
<td>20</td>
</tr>
<tr>
<td>A.2</td>
<td>gdata.f90 (2.2)</td>
<td>24</td>
</tr>
<tr>
<td>A.2.1</td>
<td>init (2.2.1)</td>
<td>25</td>
</tr>
<tr>
<td>A.2.2</td>
<td>redistribute (2.2.2)</td>
<td>26</td>
</tr>
<tr>
<td>A.2.3</td>
<td>seeed (2.2.3)</td>
<td>28</td>
</tr>
<tr>
<td>A.2.4</td>
<td>alloc_cphid (2.2.4)</td>
<td>28</td>
</tr>
<tr>
<td>A.3</td>
<td>initialize.f90 (2.3)</td>
<td>30</td>
</tr>
<tr>
<td>A.3.1</td>
<td>init_inf (2.3.1)</td>
<td>30</td>
</tr>
<tr>
<td>A.4</td>
<td>functions.f90 (2.4)</td>
<td>33</td>
</tr>
<tr>
<td>A.4.1</td>
<td>BINV (2.4.1)</td>
<td>33</td>
</tr>
<tr>
<td>A.4.2</td>
<td>viraspread (2.4.2)</td>
<td>34</td>
</tr>
<tr>
<td>A.4.3</td>
<td>herd_inf (2.4.3)</td>
<td>35</td>
</tr>
<tr>
<td>A.4.4</td>
<td>flysort/flysortm (2.4.4)</td>
<td>36</td>
</tr>
<tr>
<td>A.5</td>
<td>iohosts.f90 (2.5)</td>
<td>39</td>
</tr>
<tr>
<td>A.5.1</td>
<td>read_mark_blocks (2.5.1)</td>
<td>39</td>
</tr>
<tr>
<td>A.5.2</td>
<td>find_herds/find_farms (2.5.2)</td>
<td>51</td>
</tr>
<tr>
<td>A.6</td>
<td>iomodule.f90 (2.6)</td>
<td>54</td>
</tr>
<tr>
<td>A.6.1</td>
<td>read_inputfile (2.6.1)</td>
<td>54</td>
</tr>
<tr>
<td>A.6.2</td>
<td>input.txt</td>
<td>60</td>
</tr>
<tr>
<td>A.7</td>
<td>host.f90 (2.7)</td>
<td>63</td>
</tr>
<tr>
<td>A.7.1</td>
<td>host_move (2.7.1)</td>
<td>63</td>
</tr>
<tr>
<td>A.8</td>
<td>temperatures.f90 (2.8)</td>
<td>67</td>
</tr>
<tr>
<td>A.8.1</td>
<td>temp_var_h (2.8.1)</td>
<td>67</td>
</tr>
<tr>
<td>A.8.2</td>
<td>read_temp_id (2.8.2)</td>
<td>69</td>
</tr>
<tr>
<td>A.9</td>
<td>makemap.f90 (2.9)</td>
<td>71</td>
</tr>
<tr>
<td>A.9.1</td>
<td>make_mark_blocks (2.9.1)</td>
<td>71</td>
</tr>
<tr>
<td>A.10</td>
<td>viraemia.f90 (2.10)</td>
<td>80</td>
</tr>
<tr>
<td>A.10.1</td>
<td>host_viraemia (2.10.1)</td>
<td>80</td>
</tr>
<tr>
<td>A.10.2</td>
<td>midge_viraemia (2.10.2)</td>
<td>81</td>
</tr>
<tr>
<td>A.11</td>
<td>windy.f90 (2.11)</td>
<td>84</td>
</tr>
<tr>
<td>A.11.1</td>
<td>wind_data (2.11.1)</td>
<td>84</td>
</tr>
<tr>
<td>A.12</td>
<td>midge.f90 (2.12)</td>
<td>85</td>
</tr>
<tr>
<td>A.12.1</td>
<td>midge_local (2.12.1)</td>
<td>85</td>
</tr>
<tr>
<td>A.12.2</td>
<td>midge_wind (2.12.2)</td>
<td>87</td>
</tr>
<tr>
<td>A.13</td>
<td>bite.f90 (2.13)</td>
<td>90</td>
</tr>
<tr>
<td>A.13.1</td>
<td>inf_hosts (2.13.1)</td>
<td>90</td>
</tr>
<tr>
<td>A.13.2</td>
<td>inf_midges (2.13.2)</td>
<td>92</td>
</tr>
<tr>
<td>A.14</td>
<td>errorhandl.f90 (2.14)</td>
<td>96</td>
</tr>
</tbody>
</table>
Appendix A

A.14.1 findmidge (2.14.1) .......................... 96
A.14.2 findhosts (2.14.2) .......................... 97
A.15 Subroutine table ............................... 101


