Python-based Atmospheric Transport Model for Spruce Budworm (SBW–pyATM) v1
Technical Document, last updated 24 January 2022

Matthew Garcia, Ph.D.
matt.e.garcia@gmail.com
Postdoctoral Research Associate
University of Wisconsin–Madison

in cooperation with
Jacques Régnière, Canadian Forest Service, Laurentian Forestry Centre
Rémi Saint-Amant, Canadian Forest Service, Laurentian Forestry Centre
Brian Sturtevant, USDA Forest Service, Northern Research Station
and numerous others

NASA Application Readiness Level: 2
Maximum for stand-alone model: 3

1. Principal Sources (see References for full citations)

Greenbank et al. (1980, *Mem. Ent. Soc. Can.*)
Sturtevant et al. (2013, *Ag. For. Meteor.*)
Boulanger et al. (2017, *Ag. For. Meteor.*)
Régnière et al. (2019a, *Forests*)
Régnière et al. (2019b, *Forests*)
Garcia et al. (2022, *Ag. For. Meteor.*)

2. Overview

Sturtevant et al. (2013) introduced an individual-based model of spruce budworm (SBW) moth migration that was driven by WRF-based meteorological fields. That model examined the likelihood that an area of known SBW outbreak populations in northern Minnesota could have produced the moths that were caught in pheromone traps along the northwestern shoreline of Lake Superior, where it was known that the native SBW population was not yet ready for flight at the time of the simulated dispersal events. In this sense, the model examined "ill-timed flights" where immigrant and native SBW populations were phenologically distinct.

The Sturtevant et al. (2013) flight model was based on the movement of individual moths based on early work in aerobiology by Scott and Achtemeier (1987) and later modeling by Isard et al. (2005), who considered the insect dispersal/migration flight in three phases: liftoff, cruising, and landing. Figure 1 of that paper shows this phased process:

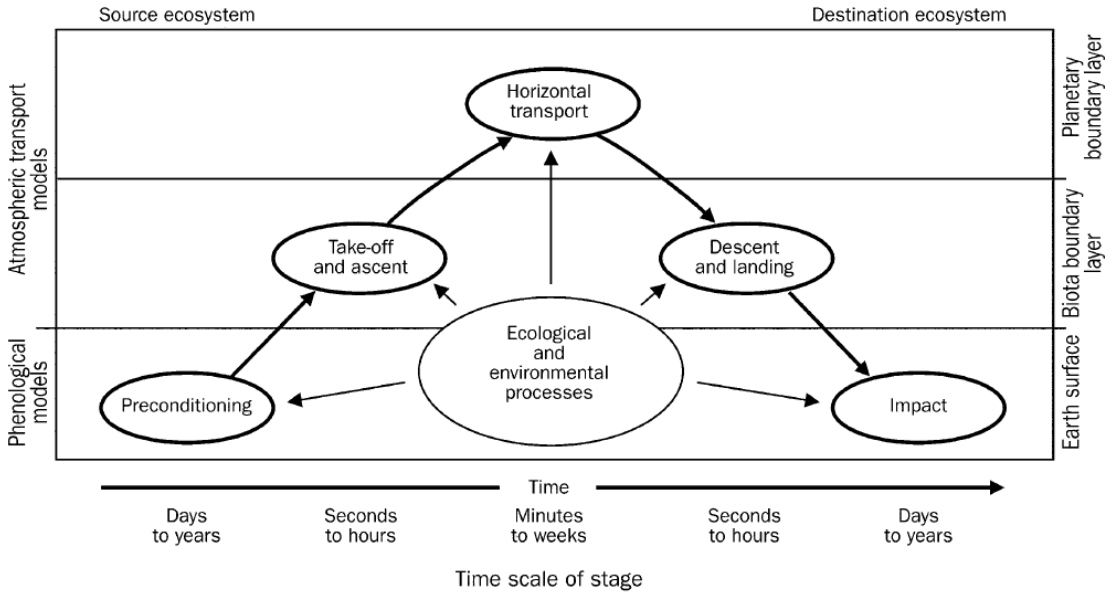

Figure 1. Aerobiology process model. Developed during the 1970s, this model provides a unifying approach to synthesize knowledge of processes associated with the aerial movement of biota. It was intended to focus research on ecological and environmental factors that affect aerobiota and to facilitate the study of organisms that use the atmospheric pathway as an integral part of dynamic ecological systems (Edmonds 1979). The time scales associated with these processes typically range from days to years for preconditioning and impact, minutes to weeks for horizontal transport, and seconds to hours for takeoff and ascent as well as descent and landing. Preconditioning and impact stages occur in ecosystems to which the aerobiota are functionally adapted. Takeoff and ascent as well as descent and landing occur within the biota boundary layer, which varies in depth with organism, weather conditions, time, and space. Winds encountered during the horizontal transport stage govern aerial pathways and consequently geographic destinations of aerobiota. Quantification of the atmospheric pathway requires the integration of phenological models that characterize aerobiota and their hosts in source and destination ecosystems with multiscale atmospheric transport models.

Figure 1 from Isard et al. (2005, *BioScience*).

This three-phase model of insect flight was elaborated for the simulation of many moths over a large geographical area. The object-oriented model structure developed for the 2013 paper treated each moth as an individual in an “agent-based” simulation environment, with individuals having their own identity and morphology (body size, flight speed, etc.) and all individuals subject to a common set of flight rules in the shared environment. The model structure in Sturtevant et al. (2013) was illustrated in Figure 1 of that paper:

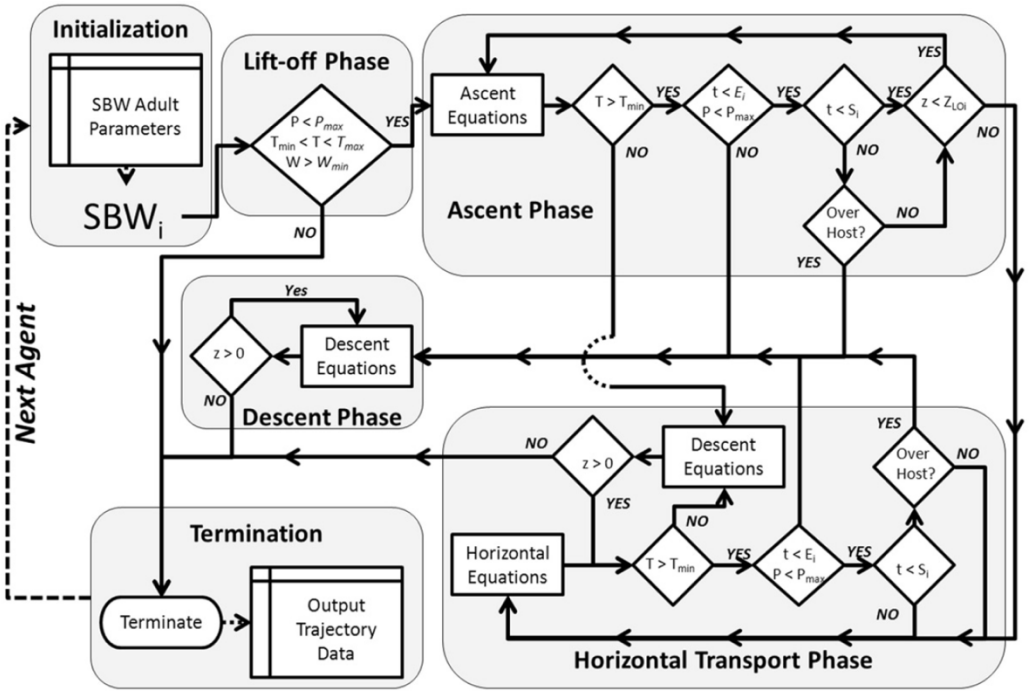

Fig. 1. Model flow diagram illustrating the logic of the spruce budworm flight behavior model. See Table 1 for parameter descriptions and values. Ascent, horizontal, and descent equations are delineated in Eqs. (1–3).

Figure 1 from Sturtevant et al. (2013, *Agricultural and Forest Meteorology*).

In addition to ill-timed flight events, the individual-based model of moth flight could be applied to any migration flight event to judge the likely landing locations of SBW moths from a known outbreak area, closing a major methodological gap in understanding the spatiotemporal dynamics of SBW outbreaks. Models of the SBW life cycle were well-established. However, the behavior of the adult SBW as a moth, subject to wind and weather and known to travel in large numbers, in coherent directions, and over long distances in a single night had been observed by several methods but had not been modeled. Prior models generally considered local dispersal, but none had been able to develop a way to anticipate and track long-distance migration as observed in SBW (and several other species).

The three-phase flight model developed by Sturtevant et al. (2013) was rewritten by Régnière and Saint-Amant as part of the Canadian Forest Service BioSIM modeling framework in C++ and known as the Atmospheric Transport Model (BioSIM–ATM). In that setting, the growth phases of the SBW larva as determined by weather conditions (esp. temperature) are estimated stochastically for an area of known outbreak right up to the adult phase, at which point each individual in the BioSIM model is allowed to fly according to the rules of the ATM and subject to the weather conditions applied to the flight portion of the model. That selection of weather conditions has varied according to the interests of the developers but is generally a medium-resolution numerical model analysis, such as the North American Model (NAM) or similar products, covering the area of interest in eastern Canada.

This Python-based Atmospheric Transport Model for Spruce Budworm (SBW–pyATM) is a counterpart to the original flight model (Sturtevant et al., 2013) and the BioSIM–ATM modeling frameworks. The most important parts related to SBW moth flight behavior, primarily the empirical descriptions of moth physiology and circadian rhythm, have been collected in a limited number of the components described below. Physiology and flight calculations for SBW are updated to the statistical-empirical formulations developed by Jacques Régnière of the Canadian Forest Service (Régnière et al., 2019a, b). Moth migration is determined in large part by atmospheric conditions, for which SBW–pyATM takes in WRF model output grids (based primarily on NARR large-scale input datasets) at a user-defined interval for the area and time of investigation. These are not raw WRF output files, but specific variables that have been preprocessed to a specific geographic configuration for pyATM use. This preprocessing also reduces the volume of data required for testing on an individual computer or transferred through a computing cluster when distributed simulations are performed. WRF preprocessing scripts are also available as part of this software package.

Version 1 of SBW–pyATM is oriented on single-night simulations of SBW flight activity, as documented by Garcia et al. (2022). The following pages contain component listings, descriptions, and flowcharts that outline the three major phases of a simulation: startup, temporal loop procedures, and rule-based decisions to determine individual moth flight behavior. The behavioral ruleset is broken into parts for ease of illustration. Additional flowcharts representing pre- and post-processing steps, parameter calibration (including comparison with radar), and multi-night simulations will be added upon completion.

3. Design

The overall approach to moth phenology and migration in BioSIM–ATM is an individual- or agent-based process. Accordingly, I developed SBW–pyATM in an object-oriented manner. Each moth is represented as a single instance (object) carrying its own morphological and environmental information but subject to a common set of flight rules (behaviors) that apply to every moth in the simulation. In this formulation, the agents are non-interactive (agents are influenced only by their environment) and are not intelligent; that is, the moths do not “learn” and adjust their behavioral rules over the course of the simulation. The user can specify any number of moths in a simulation but will likely notice a decline in model performance at large numbers because of the need to interpolate meteorological variables to each moth’s individual location at every time step. Given the nature of the simulation with no interactions between the agents, simulations are linearly separable: the results of several smaller simulations over the same domain and times can be combined to give the effect of a large simulation, but with those results obtained with far less computing time. I typically run simulations with 1000 moths for speed and ease of post-processing and have run as many as 2500 such simulations simultaneously on UW–CHTC and OSG distributed computing resources, bringing their results together afterward with post-processing routines to gather flight statistics and maps of outcomes.

A high-level schematic of the SBW–pyATM model infrastructure and data processing stream, including necessary inputs, was given by Garcia et al. (2022, Figure 2):

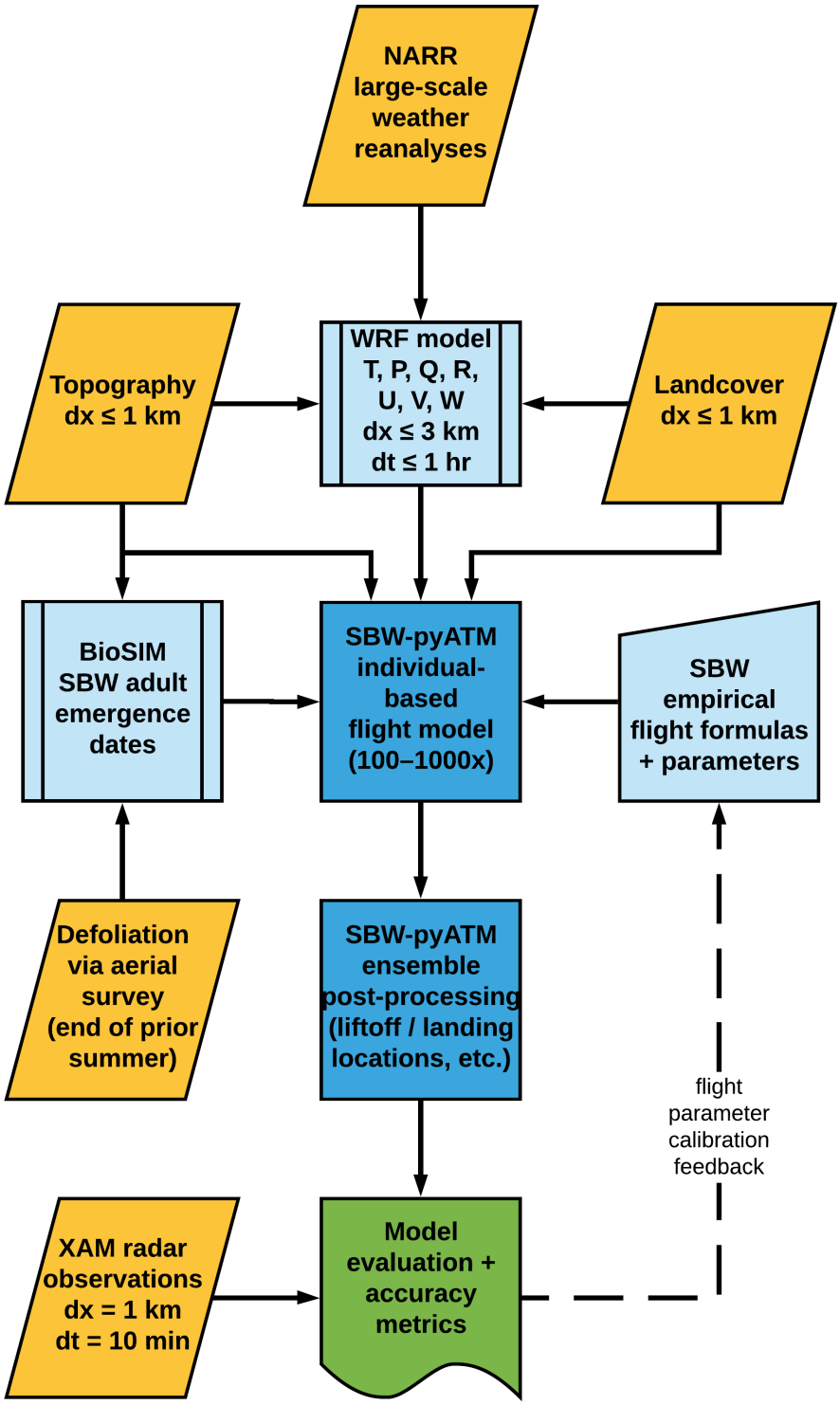

Figure 1: SBW–pyATM modeling procedure and data processing stream.

SBW–pyATM is a stand-alone model for use on any system that can support Python v3.8, such as Linux and MacOS (both tested) as well as Windows (though not yet specifically tested). In only the loosest sense is pyATM a "translation" (a.k.a. "port") of the original C++ components of BioSIM: I didn't use any pre-programmed translators or tools to convert C++ code to Python, and I didn't develop any wrappers to embed C++ code in a Python programming structure. Portions of the BioSIM–ATM modeling procedures, specifically the flight decision-making processes, are replicated in SBW–pyATM but the infrastructure and calculations necessary to drive the simulation were written entirely from scratch in Python. *Any errors in SBW–pyATM are my own.*

4. External requirements (each with their own dependencies)

copy, *datetime*, *warnings*, *os*, *sys* (Python standard libraries)
iso8601
matplotlib
mpl_toolkits.basemap (NOTE: to be replaced with *cartopy*)
netCDF4
numpy
GDAL (called via "*osgeo*") (NOTE: to be replaced with *geotiff*)
pandas
pyproj
scipy.interpolate
wrf-python (called as "*wrf*")

5. Components (by file name, in alphabetical order)

Notation
`Courier` = persistent data structure or variable name
`Courier_italics` = file name
`Courier_bold` = class (object) name
`Courier_bold_italics` = class method name
Times_New_Roman_italics = standard or external Python library
Times_New_Roman_bold = pyATM component as a Python module
Times_New_Roman_bold_italics = function name

`Clock.py`

Description:
Initialize and update the simulation time clock.
External module requirements:
datetime
Internal module dependencies:
(none)
Classes defined:
`Clock`
Class methods defined:

__init__
classes used: **Simulation**
advance_clock
Functions defined:
(none)

Circadian_calculations.py
Description:
Initialize an individual moth's circadian temporal function and pre-calculate necessary parameters using formulations in Régnière et al. (2019b).
External module requirements:
datetime, numpy
Internal module dependencies:
Interpolation, WRFgrids_class
Classes defined:
(none)
Class methods defined:
(none)
Functions defined:
assign_circadian
classes used: **Simulation, SBW, Flier**
calc_circadian_deltas
classes used: **SBW**
calc_circadian_from_WRF_T
classes used: **Simulation, SBW, Flier**
calc_circadian_p
classes used: **Clock, Flier**
get_flier_environments
classes used: **Simulation, WRFgrids**
initialize_circadian_attributes
classes used: **Simulation, SBW, Flier**

Flier_class.py
Description:
Each moth is initialized as an individual of the **Flier** class, with variable declarations for moth identification, status, location, morphology, flight characteristics, and environment. Contains methods for updating these variables on a time-step basis, for recording those values throughout the moth's activity and flight period, and for reporting that record as JSON output. Moth locations are updated in Cartesian (UTM) coordinates on a time-step basis and converted back to geographic coordinates for other uses, e.g., mapping, post-processing, etc.
External module requirements:
numpy, *pandas*
Internal module dependencies:
Flier_summary, Geography, Map_class, Plots_gen
Classes defined:

Flier

Class methods defined:

__init__

classes used: **Simulation, SBW**

calc_AM_ratio

flight_status_info

classes used: **Clock**

inside_grid

classes used: **Simulation**

landing_loc_info

liftoff_conditions

classes used: **Simulation, SBW**

liftoff_loc_info

classes used: **Clock**

report_status

classes used: **Simulation, Clock**

state_decisions

classes used: **Simulation, Clock, SBW, Map, Radar**

update_empirical_values

classes used: **Simulation, SBW**

update_environment

update_location

classes used: **Clock**

update_motion

classes used: **Simulation, SBW, Radar**

update_nu

classes used: **Simulation, SBW**

update_state

update_state_motion

classes used: **Simulation, SBW, Radar**

update_status

classes used: **Clock**

update_v_h_components

zero_motion

classes used: **Simulation**

Functions defined:

(none)

Flier_grids.py

Description:

Create and populate grids of flier/liftoff/landing/egg density during simulation. *(NOTE: this module may be deprecated as most of these fields are now mapped on an ensemble basis, not for individual simulations. In addition, post-processing of liftoff/landing/egg density now uses actual lat–lon locations for hexbin-based mapping. –MG, 12 Jul 2021)*

External module requirements:

numpy, *pandas*, *pyproj*

Internal module dependencies:
(none)
Classes defined:
(none)
Class methods defined:
(none)
Functions defined:
create_fill_grids
classes used: **`Simulation, Proj`** (via *pyproj*)
grid_flier_dvels
classes used: **`Simulation, Clock, Radar`**
grid_flier_locations
classes used: **`Simulation, Clock, Radar`**
grid_liftoff_locations
classes used: **`Simulation`**
grid_landing_locations
classes used: **`Simulation`**
grid_egg_deposition
classes used: **`Simulation, Proj`** (via *pyproj*)

`Flier_setup.py`

Description:
Moth location is initialized in geographic coordinates and converted immediately to UTM coordinates. Starting locations and morphological characteristics of the simulated moths can be provided in a BioSIM output file (CSV format) or determined by a random selection of locations on a map of defoliation severity (GeoTIFF format). If a BioSIM output file is provided, the moth morphology given in that file are used, with a random selection of the provided moths depending on the user's specification. If a defoliation map is provided, moth morphology for each random location is determined from the supplied empirical distributions.
External module requirements:
datetime, *numpy*, *pandas*
Internal module dependencies:
Interpolation, Geography, Map_class
Classes defined:
(none)
Class methods defined:
(none)
Functions defined:
generate_flier_attributes
classes used: **`SBW`**
generate_flier_locations
classes used: **`Map, WRFgrids`**
read_flier_locations_attributes
classes used: **`Simulation, Clock`**

Flier_summary.py

Description:

Summarize various statistics and histograms of moth flight characteristics, including full-flight trajectories.

External module requirements:

datetime, *iso8601*, *numpy*, *pandas*

Internal module dependencies:

Flier_grids, Plots_gen

Classes defined:

(none)

Class methods defined:

(none)

Functions defined:

get_alt_histogram
get_dist_histogram
get_horizontal_speed_histogram
get_liftoff_speed_histogram
get_speed_stats_histogram
get_time_histogram
get_time_mean_stdv
get_time_stats_histogram
flight_status_columns
plot_trajectories
classes used: **`Simulation, WRFgrids`**
report_flier_locations
classes used: **`Simulation, Clock, Radar`**
report_flier_statistics
report_liftoff_stats
report_mean
report_mean_stdv
report_mean_stdv_pooled
report_stats
report_time_stats
summarize_activity
classes used: **`Clock, Flier`**
summarize_locations
classes used: **`Clock, Flier`**
summarize_motion
classes used: **`Flier`**

Geography.py

Description:

Functions for geography-based calculations and conversions.

External module requirements:

numpy, pyproj

Internal module dependencies:
(none)
Classes defined:
(none)
Class methods defined:
(none)
Functions defined:
calc_GpH
get_utm_zone
inside_grid
classes used: **`Simulation`**
inside_init_box
classes used: **`Simulation`**
lat_lon_to_utm
classes used: **`Proj`** (via *pyproj*)
utm_to_lat_lon
classes used: **`Proj`** (via *pyproj*)

`Interpolation.py`
Description:
Functions for interpolation of WRF environment information in space and time.
External module requirements:
numpy, *scipy.interpolate*
Internal module dependencies:
(none)
Classes defined:
(none)
Class methods defined:
(none)
Functions defined:
get_interp_vals_2D
get_nearest_columns
get_nearest_vals_2D
get_vals_1D
interpolate_time

`Map_class.py`
Description:
Methods for reading GeoTIFF files of topography, landcover, and defoliation intensity. Each GeoTIFF is initialized as an object of the **`Map`** class with a method for querying map values by location during the simulation. Maps of topography and landcover are cropped to the specified simulation domain. The landcover map query function uses IGBP/MODIS landcover class values, translated to the forest/non-forest categories needed for moth status determination: water or non-host forest landing, host forest landing and thus egg-laying event, etc. For topography and/or landcover the user may also use the fields provided in

preprocessed WRF output files. The provided defoliation map is not required to cover the entire simulation domain.

External module requirements:

numpy, *gdal* (via "*osgeo*"), *sys*

Internal module dependencies:

(none)

Classes defined:

`Map`

Class methods defined:

`__init__`

classes used: **`Simulation`**

`check_map_boundaries`

classes used: **`Simulation`**

`get_value`

`get_values` `(static method)`

`subset_grid`

Functions defined:

lc_categories

lc_category

setup_defoliation_map

classes used: **`Simulation`**

setup_lc_map

classes used: **`Simulation`**

setup_topo_map

classes used: **`Simulation`**

`Model_control.py`

Description:

Central initialization and loop-based execution of the simulation. Initialization includes invoking the **`Simulation`**, **`Clock`**, and **`SBW`** classes, loading the topography and landcover **`Map`** objects, loading the defoliation **`Map`** object (if provided) and generating **`Flier`** locations and attributes, assigning the collection of **`Flier`** objects, loading the initial **`WRFgrid`** objects, and determination of the initial **`Flier`** environments. The temporal (outer) loop then proceeds from the designated start to end times, at the designated time step, with directives for updating **`WRFgrid`** objects at their own designated times. Inner loops cover the collection of moth **`Flier`** objects with steps for updating individual moth location, environment, decision-making (described in greater detail below), and resulting motion. For efficiency in the required interpolation routines, the environments of all moths are updated together. Also included are end-of-simulation directives to write any remaining active **`Flier`** records and egg deposition mapping information.

External module requirements:

sys

Internal module dependencies:

Clock, Flier_summary, Model_initialization, Model_wrapup, Oviposition_calculations, SBW_empirical, Simulation_specifications, Temporal_operations

Classes defined:

(none)

Class methods defined:

(none)

Functions defined:

ATM_main

classes used: **`Simulation, Clock, SBW, WRFgrids, Radar`**

`Model_initialization.py`

Description:

Functions for initialization of simulation classes and persistent data structures. Includes processing of command-line arguments to change some model parameters.

External module requirements:

numpy

Internal module dependencies:

Circadian_calculations, Flier_class, Flier_setup, Flier_summary, Map_class, Radar_class, Solar_calculations, WRFgrids_class

Classes defined:

(none)

Class methods defined:

(none)

Functions defined:

command_line_args

classes used: **`Simulation`**

load_initial_WRF_grids

classes used: **`Simulation, Clock, WRFgrids`**

setup_fliers

classes used: **`Simulation, Clock, SBW, WRFgrids`**

setup_maps

classes used: **`Simulation, Map`**

setup_radar

classes used: **`Simulation, Radar`**

`Model_wrapup.py`

Description:

Delegates generation of summary reports and plots at end of simulation.

External module requirements:

(none)

Internal module dependencies:

Flier_grids, Flier_summary

Classes defined:

(none)

Class methods defined:
(none)
Functions defined:
report_remaining_fliers
classes used: **`Simulation, Clock, Flier`**
report_statistics
classes used: **`Simulation, Clock`**
report_summary_grids
classes used: **`Simulation`**
report_trajectories
classes used: **`Simulation`**

`Oviposition_calculations.py`
Description:
Simple oviposition function, in the absence of BioSIM-type daily calculations.
External module requirements:
(none)
Internal module dependencies:
(none)
Classes defined:
(none)
Class methods defined:
(none)
Functions defined:
oviposition
classes used: **`SBW, Flier`**
update_flier_morphology
classes used: **`Simulation, SBW, Flier`**

`Plots_gen.py`
Description:
Mapping moth trajectories at end of flight or simulation. *(NOTE: this module may be relegated to post-simulation processing to reduce python time and memory overhead incurred in matplotlib and basemap usage. –MG, 12 Jul 2021)*
External module requirements:
iso8601, *matplotlib*, *mpl_toolkits.basemap*, *numpy*, *warnings*
Internal module dependencies:
(none)
Classes defined:
(none)
Class methods defined:
(none)
Functions defined:
plot_all_flights
classes used: **`Simulation, WRFgrids`**
plot_single_flight

classes used: **Simulation**

setup_basemap

classes used: **Simulation, Basemap** (via *mpl_toolkits*)

Radar_class.py

Description:

Initialize a radar location and define its coverage area for mapping of moth flight locations and Doppler velocity.

External module requirements:

numpy, *pyproj*, *warnings*

Internal module dependencies:

Geography

Classes defined:

Radar

Class methods defined:

__init__

classes used: **Proj** (via *pyproj*)

accumulate_grid

average_grid

count_grid

doppler_vel

classes used: **Proj** (via *pyproj*)

Functions defined:

(none)

SBW_empirical.py

Description:

Variable declarations and methods for eastern spruce budworm (*Choristoneura fumiferana*) adult moth morphology and flight parameterization, based primarily on Greenbank et al. (1980) and Régnière et al. (2019a, b).

External module requirements:

numpy

Internal module dependencies:

(none)

Classes defined:

SBW

Class methods defined:

__init__

calc_fecundity

calc_fecundity_0

calc_mass_from_gravidity

calc_mass_from_wing_area

calc_nu_L

calc_nu_L_Ts

calc_nu_T

calc_TL

calc_Ts

Functions defined:
(none)

Simulation_specifications.py

Description:
Variable declarations for simulation: geographic domain, start/end times, time step, external data files (defoliation maps, WRF grids, etc.), flier (agent) numbers, and any parameterized variables that are subject to calibration. This file is the main interface for the user to change model simulation parameters.
External module requirements:
(none)
Internal module dependencies:
(none)
Classes defined:
Simulation
Class methods defined:
__init__
Functions defined:
(none)

Solar_calculations.py

Description:
Calculates sunrise and sunset times for a provided location and date.
External module requirements:
datetime, *numpy*
Internal module dependencies:
(none)
Classes defined:
Sun
Class methods defined:
__init__
__calc
__preptime
__timefromdecimalday (static method)
solarnoon
sunrise
sunset
Functions defined:
update_suntimes
classes used: **Clock, Flier**

Temporal_operations.py

Description:
Functions used in the main temporal loop of the model.
External module requirements:
copy, *datetime*
Internal module dependencies:

Circadian_calculations, Interpolation, Solar_calculations, WRFgrids_class

Classes defined:

(none)

Class methods defined:

(none)

Functions defined:

count_active_fliers

classes used: **`Simulation, Clock, Flier`**

end_sim_no_fliers (poss. deprecated)

classes used: **`Clock, Flier`**

end_sim_no_flights

classes used: **`Simulation, Clock, Flier`**

interpolate_flier_environments

classes used: **`Clock, Flier`**

load_next_WRF_grids

classes used: **`Simulation, Clock`**

query_flier_environments

classes used: **`Simulation, Clock, WRFgrids`**

remove_fliers

classes used: **`Simulation, Clock, Flier`**

shuffle_WRF_grids

classes used: **`Simulation, Clock, WRFgrids`**

update_flier_environments

classes used: **`Clock, Flier`**

update_flier_locations

classes used: **`Clock, Flier`**

update_flier_states

classes used: **`Simulation, Clock, SBW, WRFgrids, Flier`**

update_flier_status

classes used: **`Clock, Flier`**

`WRFgrids_class.py`

Description:

Methods for reading pre-processed WRF output files on a user-defined interval, storing variables for use on a time-step basis, interpolating 3D WRF variable cubes from sigma coordinates to vertical coordinates based on geopotential height, and 2D interpolation to a moth location to obtain surface environmental variables (using 2D WRF surface variable grids) or a 1D column of values that vary with geopotential height (using 3D WRF upper-air variable cubes), from which variable values at the moth's calculated geopotential height are then interpolated. The 2D interpolation uses a nearest neighbor selection method and 1D vertical columns use a linear interpolation method to obtain the **`Flier`** environment. A single preprocessed WRF output file at one time-point is initialized as a **`WRFgrid`** object containing all available surface variable grids and upper air variable cubes for that time, and the use of these grids and

interpolation methods is determined by the moth altitude (above ground level, AGL) at each time step in the simulation.

External module requirements:

netCDF4, *numpy*, *os*, *sys*, *wrf-python* (called as "*wrf*")

Internal module dependencies:

Interpolation

Classes defined:

`WRFgrids`

Class methods defined:

`__init__`

classes used: **`Dataset`** (via *netCDF4*)

`get_col_3D`

classes used: **`Simulation`**

`get_flier_environments`

classes used: **`Simulation, Map`**

`get_interp_columns`

`get_nearest_locs`

`get_vals_2D`

classes used: **`Simulation`**

`get_WRF_grid_by_name`

`interpolate_space`

classes used: **`Simulation`**

Functions defined:

check_for_WRF_file

6. Usage

First, edit the file *`Simulation_specifications.py`* for your desired simulation domain and time period, simulation time step, WRF file path and interval, number of fliers (max 100K), BioSIM file path, landcover and topography GeoTIFF file path(s), experimental variable values, etc. Then, for a simulation using all default values, at the command prompt ("`$`") type

```
$ python Model_control.py 0
```

For an experiment in which the sensitivity of the model to different parameter values is used, type something like

```
$ python Model_control.py 1 min_windspeed 1.0
```

The first integer value after the program name is a simple numerical label, as for a sensitivity experiment, and has no effect on the model performance. If a flight parameter name and value are also given, the given value overrides the default value for that parameter in the simulation.

7. Model Internal Operations

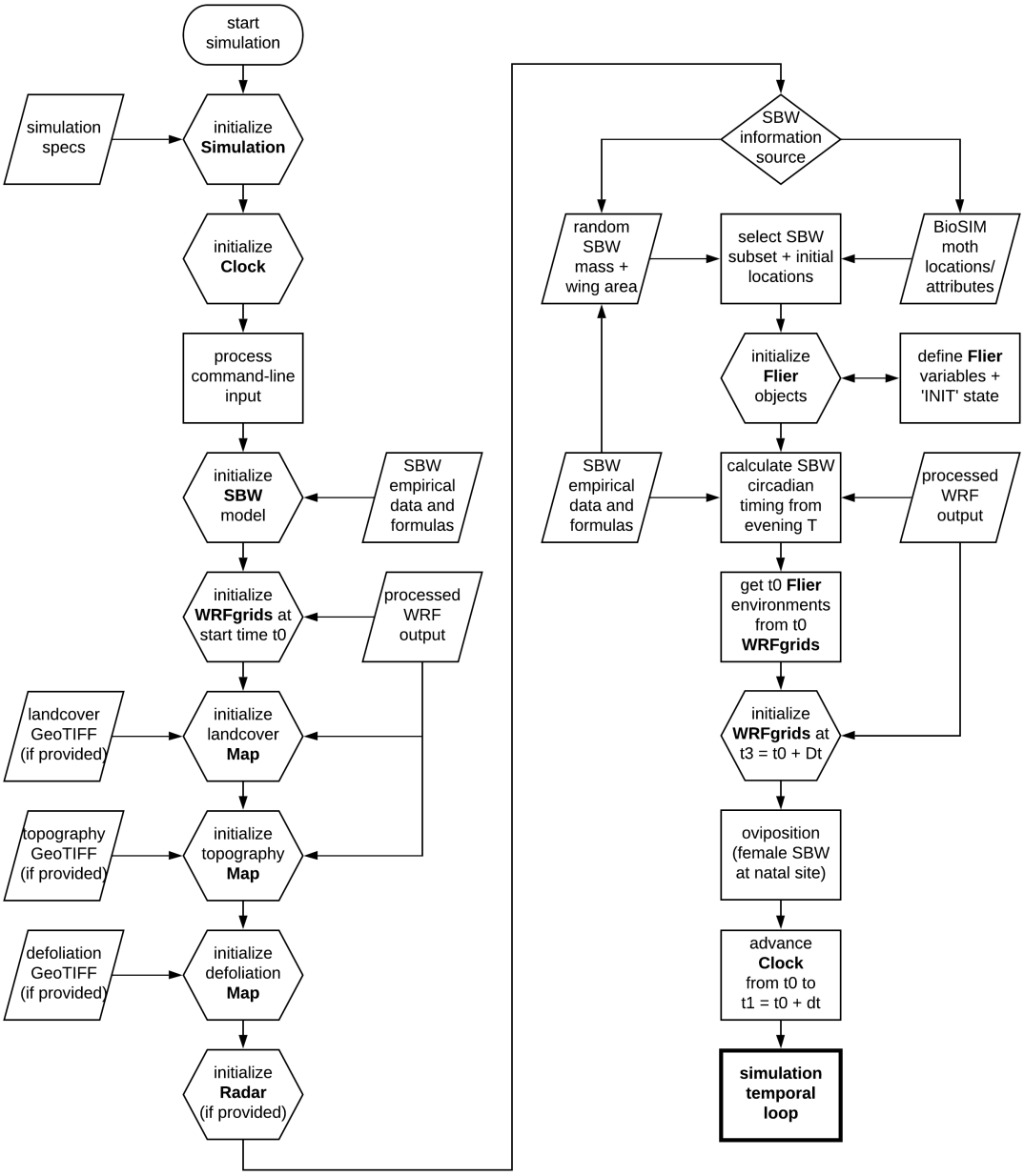

Figure 2: SBW–pyATM v1 simulation startup procedure. Note: Dt = WRF input interval, dt = model time step.

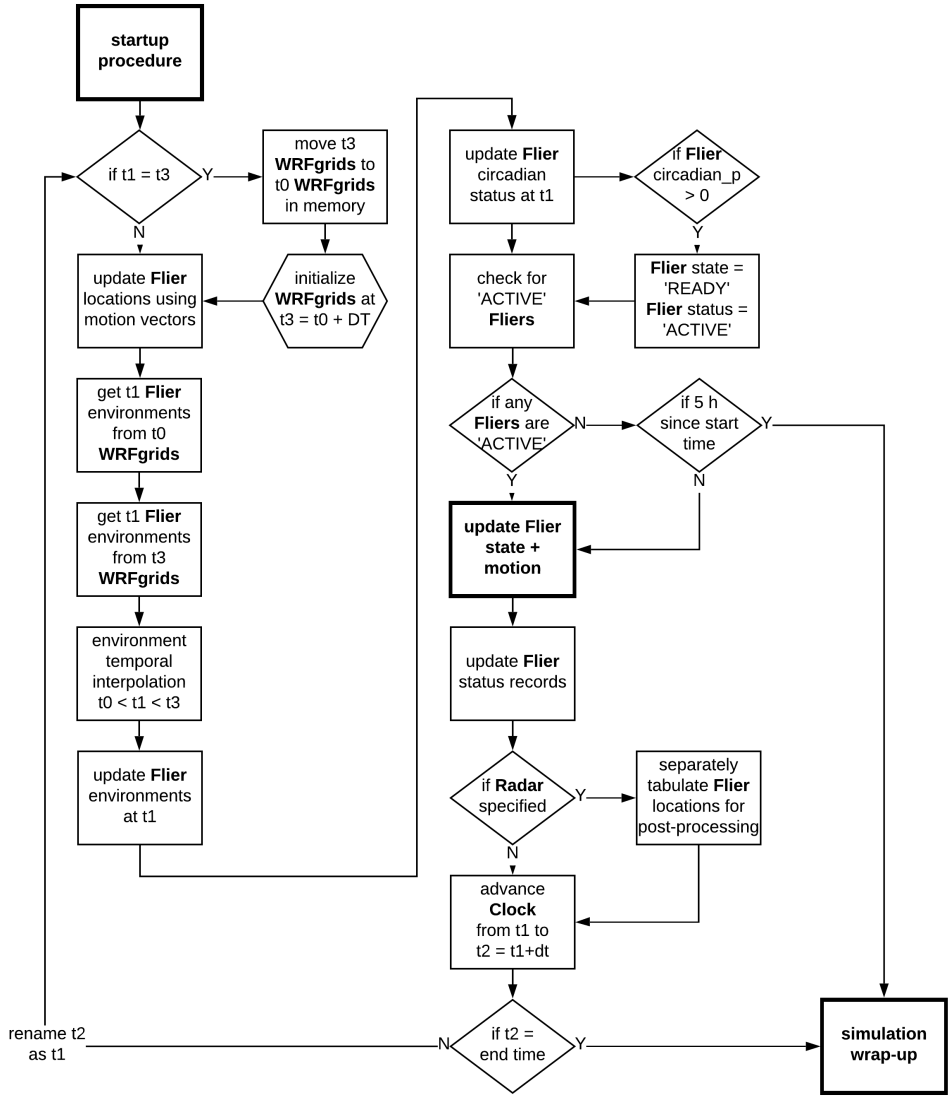

Figure 3: SBW–pyATM v1 simulation temporal loop procedure. Note: Dt = WRF input interval, dt = model time step.

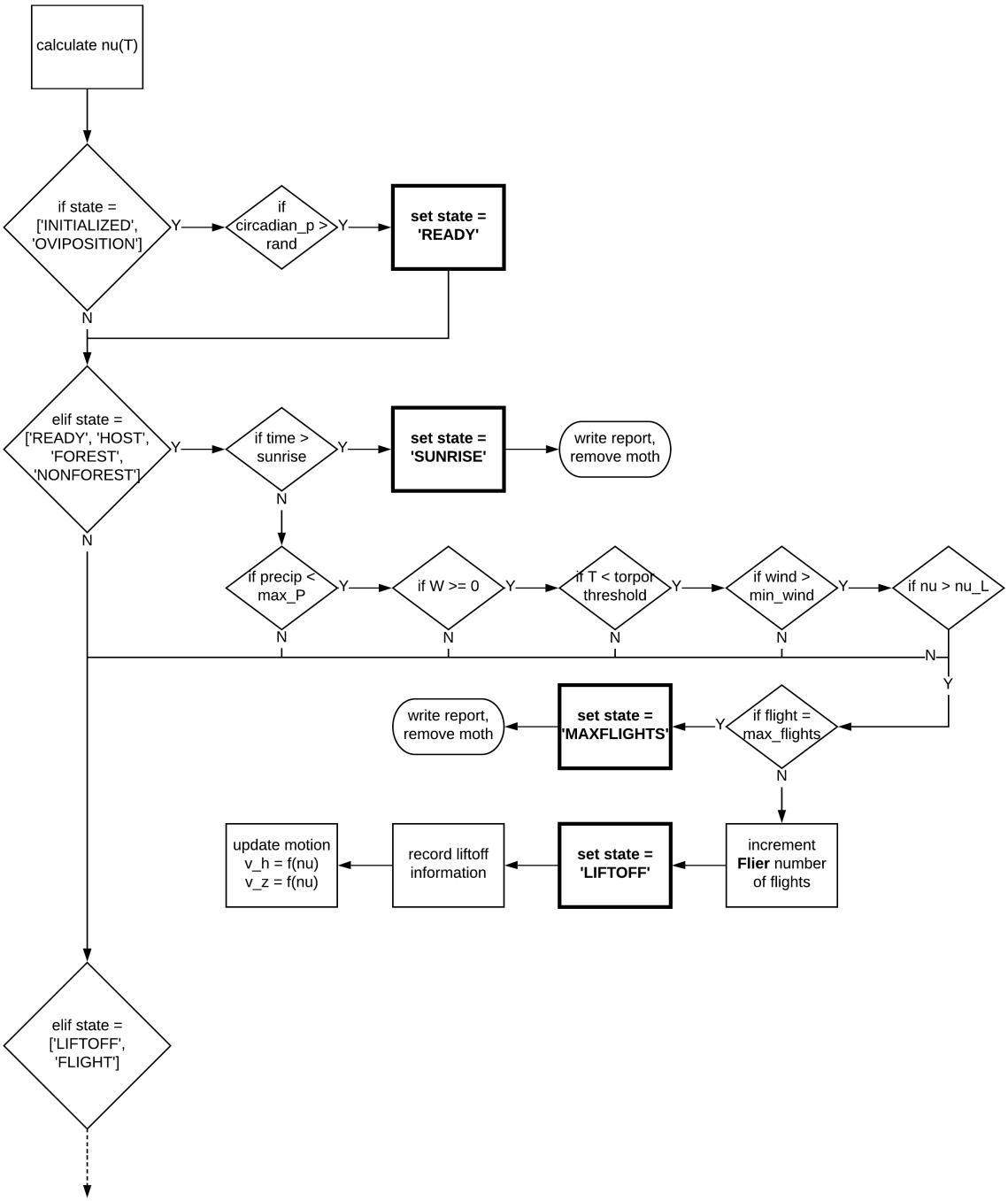

Figure 4: SBW–pyATM v1 rule-based moth behavior, part 1.

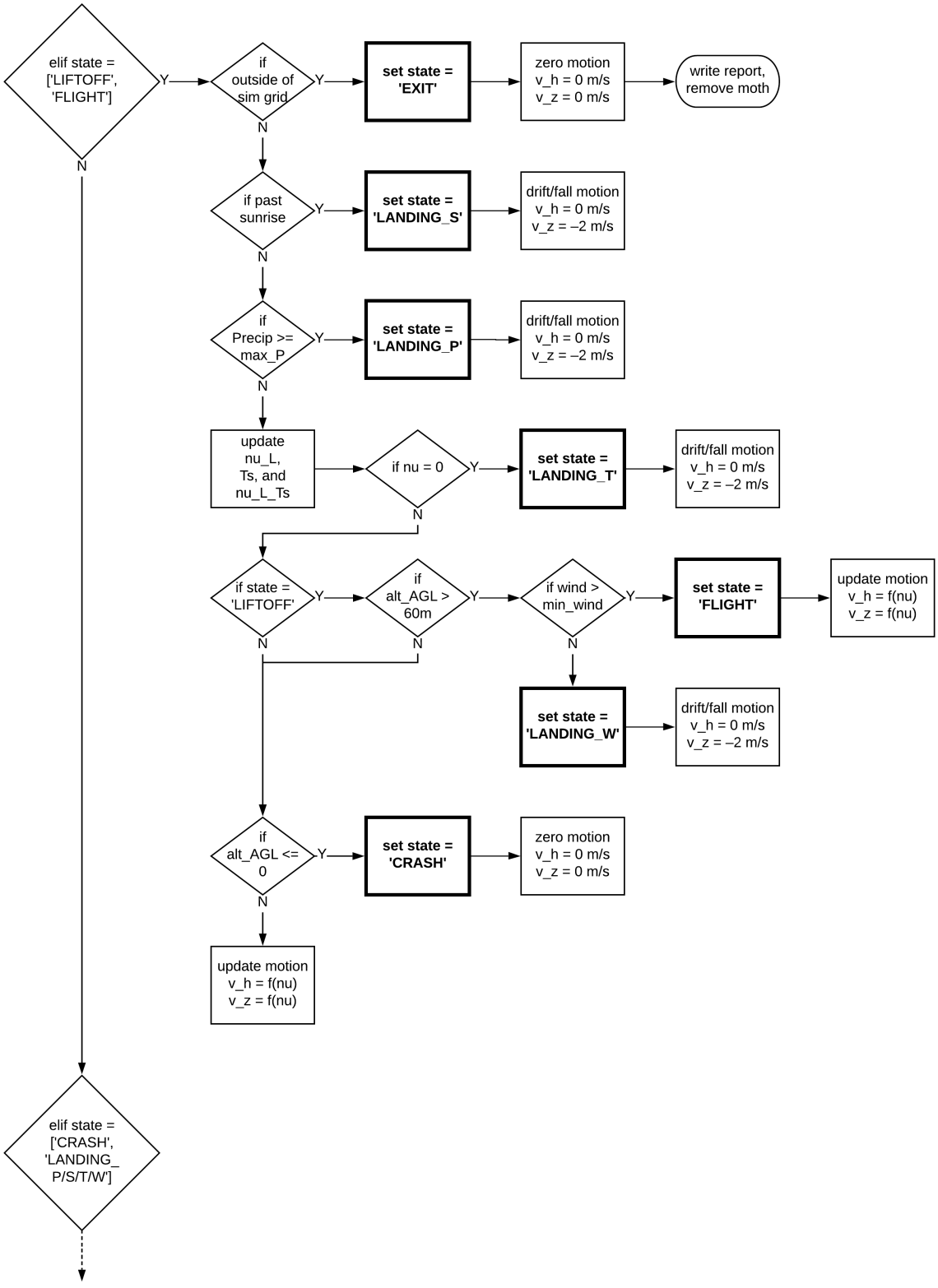

Figure 5: SBW–pyATM v1 rule-based moth behavior, part 2.

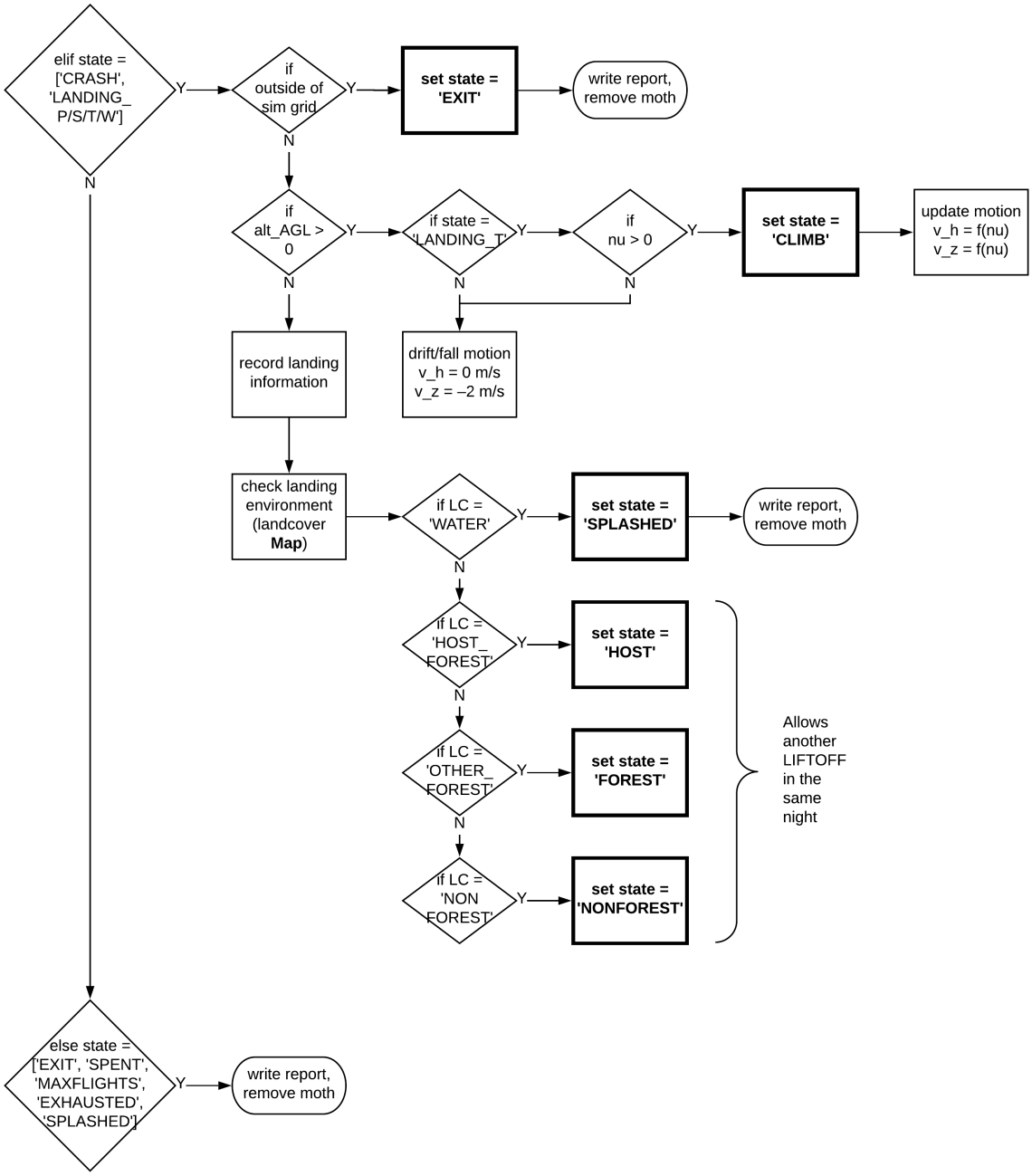

Figure 6: SBW–pyATM v1 rule-based moth behavior, part 3.

References

Boulanger, Y., F. Fabry, A. Kilambi, D.S. Pureswaran, B.R. Sturtevant, and R. Saint-Amant, 2017: The use of weather surveillance radar and high-resolution three dimensional weather data to monitor a spruce budworm mass exodus flight. *Agricultural and Forest Meteorology*, **234–235**, 127–135, http://doi.org/10.1016/j.agrformet.2016.12.018.

Garcia, M., B.R. Sturtevant, R. Saint-Amant, J.J. Charney, J. Delisle, Y. Boulanger, P.A. Townsend, and J. Régnière, 2022: Modeling weather-driven long-distance dispersal of spruce budworm moths (*Choristoneura fumiferana*). Part 1: Model description. *Agricultural and Forest Meteorology*, **315**, 108815, https://doi.org/10.1016/j.agrformet.2022.108815.

Greenbank, D.O., G.W. Schaefer, and R.C. Rainey, 1980: Spruce budworm (Lepidoptera: Tortricidae) moth flight and dispersal: New understanding from canopy observations, radar, and aircraft. *The Memoirs of the Entomological Society of Canada*, **112**, 1–49, http://doi.org/10.4039/entm112110fv.

Isard, S.A., S.H. Gage, P. Comtois, and J.M. Russo, 2005: Principles of the atmospheric pathway for invasive species applied to soybean rust. *BioScience*, **55**, 851–861, https://doi.org/10.1641/0006-3568(2005)055[0851:POTAPF]2.0.CO;2.

Régnière, J., J. Delisle, B.R. Sturtevant, M. Garcia, and R. Saint-Amant, 2019a: Modeling migratory flight in the spruce budworm: Temperature constraints. *Forests*, **10**, 802, http://doi.org/10.3390/f10090802.

Régnière, J., M. Garcia, and R. Saint-Amant, 2019b: Modeling migratory flight in the spruce budworm: Circadian rhythm. *Forests*, **10**, 877, http://doi.org/10.3390/f10100877.

Scott, R.W., and G.L. Achtemeier, 1987: Estimating pathways of migrating insects carried in atmospheric winds. *Environmental Entomology*, **16**, 1244–1254, https://doi.org/10.1093/ee/16.6.1244.

Sturtevant, B.R., G.L. Achtemeier, J.J. Charney, D.P. Anderson, B.J. Cooke, and P.A. Townsend, 2013: Long-distance dispersal of spruce budworm (*Choristoneura fumiferana* Clemens) in Minnesota (USA) and Ontario (Canada) via the atmospheric pathway. *Agricultural and Forest Meteorology*, **168**, 186–200, http://doi.org/10.1016/j.agrformet.2012.09.008.