Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Hive Simulation

Stefan Gugler & Elias Huwyler & Fabian Tschopp

Zurich
December 2013

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Stefan Gugler   Elias Huwyler   Fabian Tschopp

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

This sheet must be signed and enclosed with every piece of written work submitted at ETH.

I hereby declare that the written work I have submitted entitled

Hive Simulation

is original work which I alone have authored and which is written in my own words.*

Author(s)

Last name	First name
Gugler, Huwyler, Tschopp	Stefan, Elias, Fabian

Supervising lecturer

Last name	First name
Kuhn, Woolley	Tobias, Olivia

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

ETH Zurich, 12.12.2013
Place and date

Signature [illegible] [illegible] [illegible]

*Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Print form

Abstract

In this article, we present a model that simulates the behaviour of a single hive, interacting with its environment. Three different kinds of bees were implemented: The forager, the scout bee and the hive bee. The hive is a quantitative model mostly based on the works of *David S. Khoury et al.* [4] using a range of differential equations to represent hive growth, social inhibition, food availability and bee ageing. The focus of this model lies, in contrast to *Khoury et al.*, in the hive's interaction with the environment and it's dependency on the seasonal changes (no food income in winter, different flowers in each month). To simulate seasons, a fine-grained and easily extendible environment simulation is used, in which the bees navigate and collect food. For the development of the environment simulation, empirical observations by *T.D. Seeley* [6] were used.

Contents

1 Individual contributions

We discussed and extended the model together on a weekly basis, collecting and mixing individual ideas. The report was written in a cooperative manner. Every team member wrote the sections according to their *individual tasks* and then corrected and extended the sections of the other two members. The individual tasks have been assigned by taking into account the programming skills, personal interests and available time.

Individual tasks:

- *Stefan Gugler:* Interpretation and application of empirical data in the basic and advanced model. Searching references and data in the literature.
- *Elias Huwyler:* MATLAB scripts for map generation, map loading and data plotting. Interpretation and analysis of the simulation results.
- *Fabian Tschopp:* Implementation of the basic and advanced model in MATLAB (object oriented structure, code hierarchy, actual implementation).

2 Introduction and Motivations

The global decrease of honey bee populations (*Apis mellifera*) is not only conspicuous, but also alarming. Our dependency on pollination in almost all forms of agricultural cultivation (such as fruit and vegetables, spices, nuts etc.) urges us to find out more about the bee's behaviour and their mortality. In recent years, many studies [5] [7] have been carried out to find out the reasons for the mass decline of pollinators: diseases, destruction of the environment, pesticides/fungicides (imidacloprid, deltamethrin [**decourtye04**], thiamethoxam, clothianidin, fipronil, chlorpyriphos, cypermethrin and deltamethrin) and maybe even climate change. The sum of all these factors culminates in the mass death of bees and can lead to a serious economic crash as well as to a impairment of agricultural resources.
This motivated us to develop a simulation to find corner situations in which the hive's health becomes unstable and does not survive. We started with an existing simulation by *Khoury et al.* [4] and extended it with knowledge and empirical data from *T.D. Seeley* [6].

3 Description of the Model

3.1 Basic model: Hive simulation

Our model is widely based on the studies and equations of *Khoury et al.* [4]. We used his quantitative models (equation 1 to 6) as described bellow and added new parameters to find out more about the bee's behaviour.
Food, seasons, brood, foragers and hive bees are the cornerstone of our model. The dynamics of the hive is based on the bee's behaviour and their interaction with the environment as well as natural influences, i.e. seasons (which are separately added to *Khoury's* model). Food here means nectar and pollen, which doesn't need to be further distinguished after according to *Khoury* for simplicity reasons. After the queen laid an egg, a larvae develops inside a honeycomb cell. The equations show the proportionality of the brood to the food income and the relations between the bee number. Neglecting the complex process of reaching adulthood, we assume, according to *Khoury* that larvae become adult hive bees 12 days after pupation. The mortality rate of hive bees or capped brood is negligible if there are not specific diseases implemented. The agents are only female bees, since they are responsible for maintenance and sustainability of the hive and foraging process. The males are ordinary hive bees. With all these information we can set up the equation for the brood number change:

$$\frac{dB}{dt} = LS(H,f) - \varphi B \tag{1}$$

It associates different factors to the brood number. L is rate of the bee queen and $S(H,f)$ is a function of the survival rate dependent on the amount of food and the number of hive bees. φ is the adult bee emerging factor and B represents the uncapped brood. The equation gives us the survival rate of the brood B. $S(H,f)$ with H representing the number of hive bees and f the amount of food, is modelled as following:

$$S(H,f) = \frac{f^2}{f^2+b^2}\frac{H}{H+v} \tag{2}$$

The variable v indicates the effect of the hive bees on the brood, whereas b shows the food effect on brood survival. It decreases as food increases. As f and H become very large, $S(H,f)$ becomes constant. The first factor is a sigmoid function and shows the correlation of the food available and the capped brood. A decrease in brood can occur because of lack of food (they cannot be fed) and / or because adult bees consume the larvae to keep the resources in the hive and recycle the proteins (cannibalism). The second factor models the interdependency of the hive

bee numbers on the survival of the brood. If there are large stores of food but no hive bees that can provide the food to the larvae, the brood survival also declines.
The second differential equation describes the rate at which the number of hive bees changes.

$$\frac{dH}{dt} = \varphi B(t-\tau) - HR(H,F,f) \tag{3}$$

τ is the ageing time and $\varphi B(t-\tau)$ is the rate at which adult bees develop from pupation to adult bees (emergence rate). H is again the hive bee number, the last term is the recruitment function and given as

$$R(H,F,f) = \alpha_{min} + \alpha_{max}(\frac{b^2}{b^2+f^2}) - \sigma(\frac{F}{F+H}) \tag{4}$$

This function models the change from a hive bee to a forager bee. α_{min} denotes the transition when there is enough food but not much foragers. Vice versa, α_{max} is used for less food. As before, b is the food effect on brood survival and f is the amount of food. σ is the strength of social inhibition and is dependent on the number of foragers and hive bees.
The third differential equation describes correspondingly the rate at which the number of foragers changes:

$$\frac{dF}{dt} = HR(H,F,f) - mF \tag{5}$$

The mortality rate mF indicated the rate at which the foragers F die. The recruitment function is the same as above.
The last equation outlines with the rate of food change in the hive's stores. Forager bees and hive bees are treated equally because their consumption is almost the same.

$$\frac{df}{dt} = cF - \gamma_A(F+H) - \gamma_B B \tag{6}$$

where c describes the average food a single forager collects per day.γ is the consumption of adult bees (A) and brood (B). In our extended model, we implemented an actual environment simulation so that the food income changes over time (see chapter 3.3). Equation 6 is then replaced by equation 8, which takes the daily food income instead of a fixed rate c per forager.

With those four differential equations, the change of brood (1), hive bee number (3), foragers (5) and change of food store (6) and the functions of the brood survival rate, correlating to the hive bee number and the food store (2) and a recruitment function (4), we can simulate a basic model of a hive with its interactions and changes. So far, it is identical with *Khoury's* model.

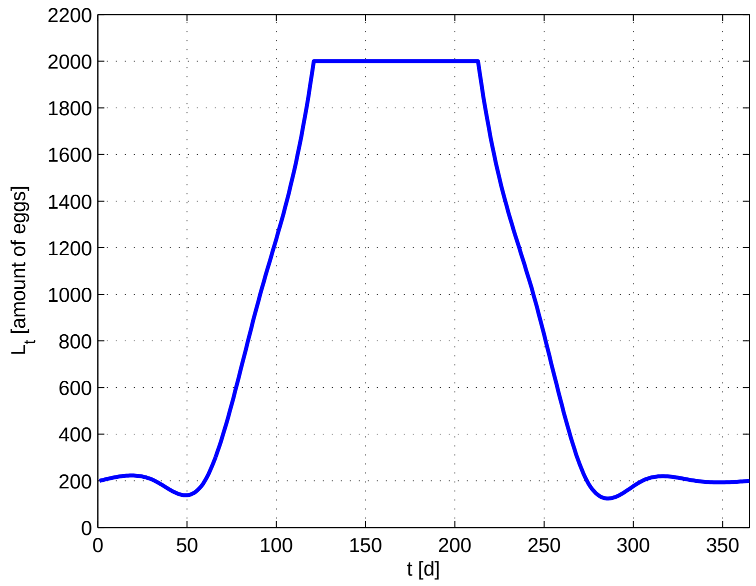


Figure 1: *The laying rate of the bee queen plotted over a year. The amount of eggs is capped after 2000 eggs.*

3.2 Changing laying rate over seasons

To make the model more realistic, equation 1 is replaced by equation 7. This incorporates that the bee queen does not have a constant laying rate. Instead, interpolated values based on *Wisdom of the Hive*, p. 34 [6] are used for the laying rate L_t (see fig. 1). *T.D. Seeley* denotes that the queen starts reproducing by early spring. We assumed that reproduction between April/May and October is realistic, as we obtained similar numbers of bees in summer (maxima) and winter (minima) to the empirical data of *T.D. Seeley* [6]. The minimal laying rate of 200 eggs per day is used to compensate mortality during winter. The peak of the laying rate (2000 eggs per day) is the same as used by *Khoury et al.* [4] and *Wright* [8]. The results are discussed in chapters 5.1 and 5.2 with the corresponding plots in appendix B.1.2 and B.1.3.

$$\frac{dB}{dt} = L_t S(H, f) - \varphi B \tag{7}$$

The introduction of dynamic laying rates led to problems with the mortality rates tested by *Khoury et al.* [4]. We discuss this in chapter 5.2.

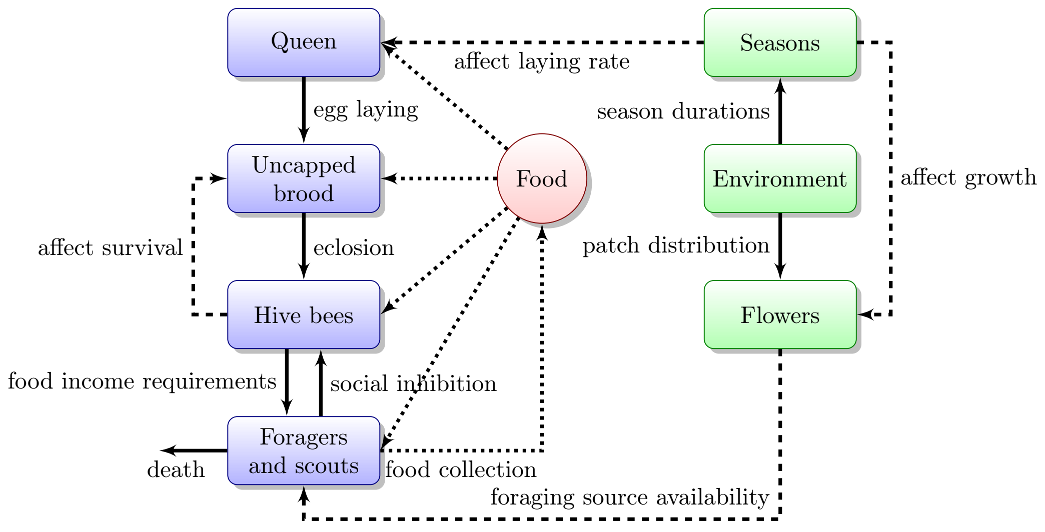


Figure 2: *Advanced honey bee social dynamics covered by this model.*

3.3 Advanced model: Environment simulation

As we were interested in observing the social dynamics under different food availability conditions, in contrast to the model of *Khoury et al.* [4], we had to implement a simulation for the environment. This advanced model introduces an environment with seasons which affect the queen's laying rate (see equation 7), the distribution and quality of flower patches and seasonal flower growth. The forager bees are then simulated on the environment, searching and collecting food according to the rules in chapter 3.3.1. An overview of the complete model is given in Figure 2.

3.3.1 Bees' working states

The environment simulation only simulates forager and scout bees. Hive bees are not considered as agents. For the foragers and scouts, we developed a set of simple rules which try to be as close as possible to the foraging process described by *T.D. Seeley* [6]. All essential rules are represented in Figure 3.

At the beginning of every day ($t_s = 0$), all bees are unemployed. There are three possibilities for every unemployed bee which are considered in every simulation step:

- If the amount of active bees according to the current time of the day is not reached, the bee can get a job assigned. Otherwise the bee stays unemployed.
- If the maxima of scouts is not reached (6% of the bees' foragers [6], p. 86),

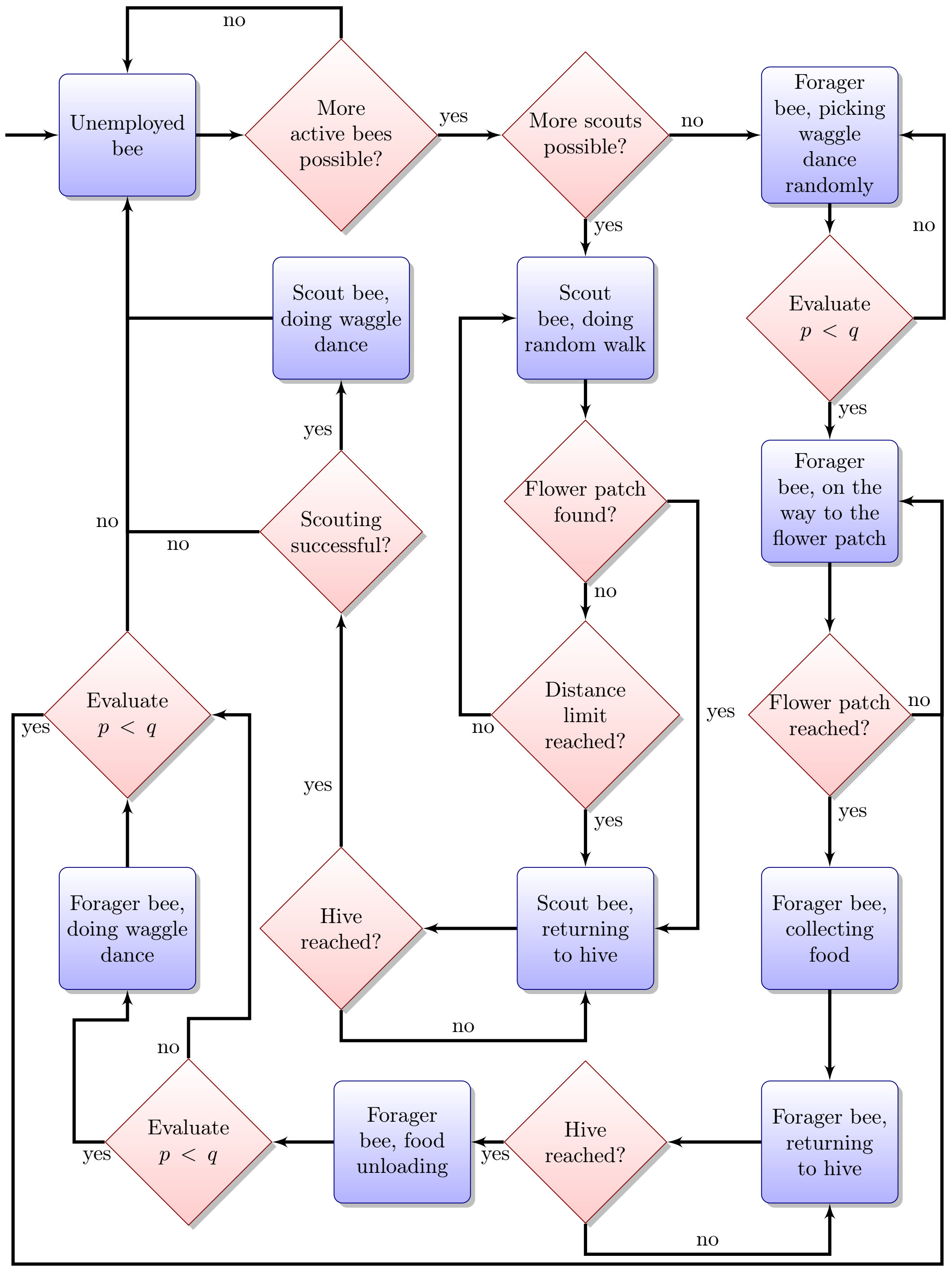

no
Unemployed bee
More active bees possible?
yes
More scouts possible?
no
Forager bee, picking waggle dance randomly
yes
no
Scout bee, doing waggle dance
Scout bee, doing random walk
Evaluate $p < q$
yes
Forager bee, on the way to the flower patch
Flower patch found?
no
Scouting successful?
no
no
yes
Distance limit reached?
no
Evaluate $p < q$
yes
yes
Flower patch reached?
no
yes
yes
yes
Forager bee, doing waggle dance
Hive reached?
Scout bee, returning to hive
Forager bee, collecting food
no
no
Evaluate $p < q$
yes
Forager bee, food unloading
yes
Hive reached?
Forager bee, returning to hive
no

Figure 3: *State transitions of the bees in the environmental simulation.*

then the bee becomes a scout.

- Otherwise, the bee becomes a forager.

When a bee agent gets a scouting job assigned, the agent continues the following way:

- The scout performs a random walk (see chapter 4.4.1) and checks with every iteration step, if any flower patch has been passed (see chapter 4.4.2).
- If a flower patch is found or the flight distance limit is reached, the bee returns to the hive. Path optimization (see chapter 4.5.1) may occur.
- Back in the hive, the bee performs a waggle dance if scouting has been successful and goes back to the unemployed state.

When a bee agent gets a foraging job assigned, the agent continues the following way:

- Among the available waggle dances, the forager chooses randomly and evaluates if the food source is good enough ($p < q$) to visit or not (see chapter 3.3.2). This is repeated in every iteration step until the forager decides to visit the flower patch represented by the chosen waggle dance.
- The forager flies to the patch and, once reached, collects as much food as the flowers in the patch allows (see chapter 4.2).
- After foraging, the bee returns to the hive. Path optimization may occur again.
- Back in the hive, the forager unloads the collected food (increasing $f_{d,t}$, equation 8).
- The bee now has to evaluate the food source several times (($p < q$), once per iteration step) and decide if it is worth reporting the patch (executing a waggle dance) or not.
- A final evaluation ($p < q$) has to be done to decide if the forager will abandon it's own food source and become unemployed or fly to the same patch again.

3.3.2 Foragers' distribution across flower patches

It is clear from chapter 3.3.1 that after some time, the bees will focus on newly reported and then on the most profitable food sources because of relative quality evaluation. $p \in [0,1]$ denotes an equally distributed random value. q denotes the relative quality of a given flower patch compared to all patches currently available as a waggle dance. In detail, the evaluation $(p < q)$ works as follows:

Depending on the random variable p and the relative patch quality q represented

$$\frac{df}{dt} = f_{d,t} - \gamma_A(F + H) - \gamma_B B \tag{8}$$

Where $f_{d,t}$ denotes the amount of food the foragers collected on day t.

3.4 Empirical data bases

3.4.1 Area around the hive

By analysing the different waggle dances of the forager bees, *T.D. Seeley* states in *Wisdom of the Hive* pp. 37ff that 95% of the foraging process occurs in a radius of 6.0 km (maximum distance at 10.9 km, mean distance at 2.2 km and the modal distance at 1.6 km [6]. The approximation of our modelled square $(6\text{ km} \cdot 2)^2 \approx 100\text{ km}^2$ should be sufficient for the simulation and comprehend all important data. In our model, we split the area up into a grid of 10 m $\cdot$ 10 m tiles of flower patches.

3.4.2 Flower patches and food

The flower patches themselves are divided into their different blossom: spring, summer and autumn. The empirical data was taken from *Wisdom of the Hive* p. 44f (year 1982). We merged the raw data and received a new graph for the whole year (see fig. 4). With equation 9, we can find out the weight of food a bee carries per flight.

$$w\,[\frac{g}{flight\,and\,bee}] = \frac{M\,[\frac{kg}{day}]}{C\,[bees] \cdot l\,[\frac{flights}{day\,and\,bee}]} \cdot 1000\,[\frac{g}{kg}] \tag{9}$$

The result w is an approximation in gram per bee and flight. M denotes the change of the mass of the hive per day in kilograms (see fig. 4) and C represents the forager bee count approximated for a specific day of *T.D. Seeley's* empirical data [6]. l is the flight count a bee can do in average per day. Thus, we can take hive weight change per day as an indicator for the flower quality and flower type present at a given day of the year. This is the best approximation we tried, we did not find

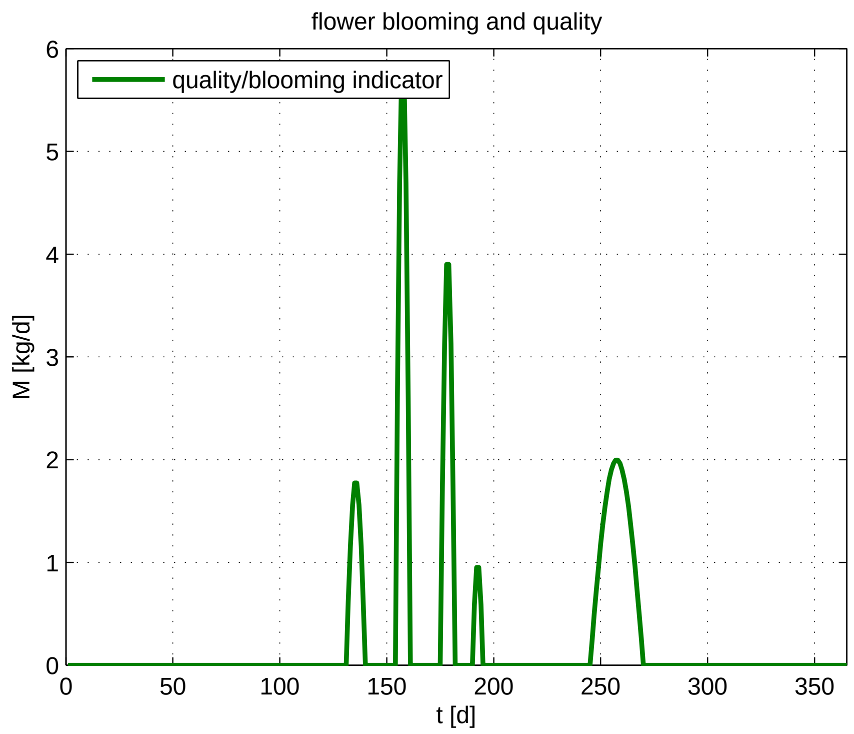


Figure 4: *This figure shows the change of the mass of the hive for any day in the year. It correlates with the quality of a flower patch and the type of flower, respectively.*

actual data of blooming and foraging profit as it is very difficult to measure this in nature.

4 Implementation

4.1 General implementation

4.2 Map and flower patch quality

For the environment, two maps are used. One map indicates which flower type grows at a given integer location

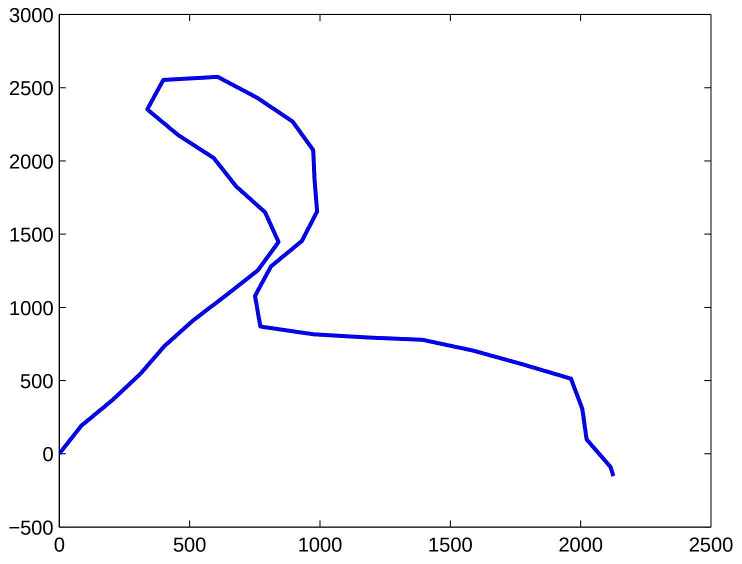


Figure 5: *Example of a random walk executed by a scout bee.*

4.3 Hives

4.4 Scouts

4.4.1 Random walk

4.4.2 Bresenham line algorithm

As the scout bees pass a distance of $v \cdot \Delta t_s$, respectively $\sqrt{(v_x \cdot \Delta t_s)^2 + (v_y \cdot \Delta t_s)^2}$ for a velocity v and time step Δt_s in every iteration step, multiple field are being passed at once. In order to check all fields between the last and current iteration step for flower patches, the Bresenham line algorithm is used. With this method, all integer coordinates between two given points on the map are reported back. Afterwards, every obtained point can be checked against the current quality- and type-map of the environment simulation. For this simple algorithm, a pre-existing implementation is used [2], appendix A.12. Figure 6 is an example on a 15x15 map with coordinates $(x = 4, y = 3)$ to $(x = 12, y = 14)$ and reported points (black boxes).

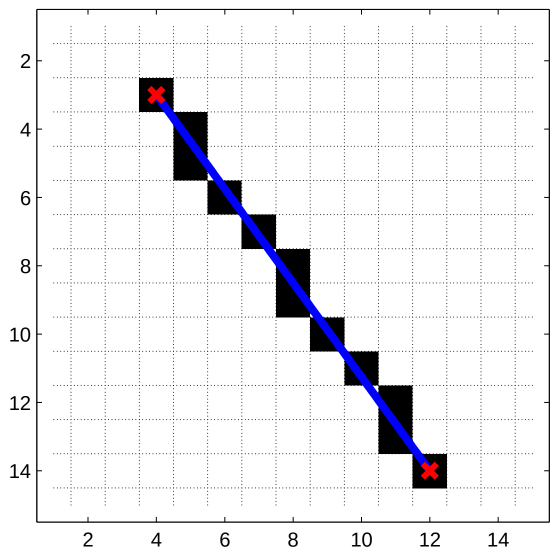


Figure 6: *Example of Bresenham's line algorithm with map and path segment.*

4.5 Foragers

4.5.1 Path optimization

We assume bees can optimize the original path they obtain from a waggle dance, as the bees are able to orientate themselves in the environment with sun positioning and so on indicates that the path reported by a scout bee can be very random and not efficient (not the most direct possible path)...

5 Simulation Results and Discussion

5.1 Constant food and egg laying

The first iterative simulation run is based only on the equations of *David S. Khoury et al.*[4]. As expected the hive stabilizes at an equilibrium point while the stored food increases infinite. (See graph B.1.2)

5.2 Constant food, dynamic egg laying

The second iterative simulation run was to see if the new laying function (Figure 1) was working as intended. As a result the bee population does not stabilize at one

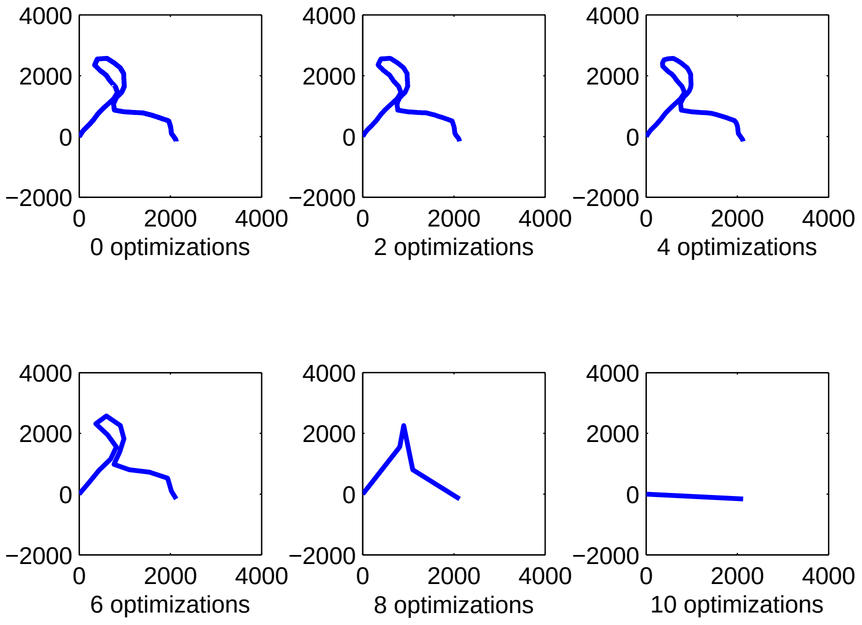


Figure 7: *Example of path optimization used to short cut the path to flower patches.*

point, it now describes a more natural periodic form with a population maximum around August and a minimum around February.

The colony we simulated did not survive with any daily mortality rates higher than 0.1. There are not enough bees emerging from eggs in order to compensate the mortality rate. However, we found that the new mortality rates assumed in our simulation (0.075 daily mortality) are still realistic. *Henry et al.* [1] describe mortality rates between 0.102 and 0.316 in empirical testing, which is the range in which *Khoury et al.* [4] simulated. *R. Dukas* [3] observed that mortality of foragers is in this range mainly because of predation. As we apply the mortality to all days, even in times and seasons without foraging (all bees stay in the hive), our simulated mortality rates are naturally lower.

The stored food still increases infinite but now the correlation between the number of forager bees and the daily increase in stored food is very to see (linear correlation coefficient = 0.9775).

5.3 Environmental model

5.4 Empirical data based run

In this run we wanted to see if our model can produce the same results as *T.D. Seeley's* experiments from *Wisdom of the hive* pp. 44 fig. 2.14. Our date correlates quite well with the empirical data before swarming occurs, year 1982 compared to graph B.2.2. As swarming was never a part of our model were satisfied with the accuracy of our model. Compared to our earlier simulations, the food collection rate is now dominated by the availability of flowers. The forager count is still a important factor as later simulations will show.

5.4.1 Typical simulated day

5.4.2 Missing flower seasons

5.4.3 Variations on autumn flowers

6 Summary and Outlook

7 References

[1] Henry et al. "A Common Pesticide Decreases Foraging Success and Survival in Honey bees". In: *Sciencemag* (2012).

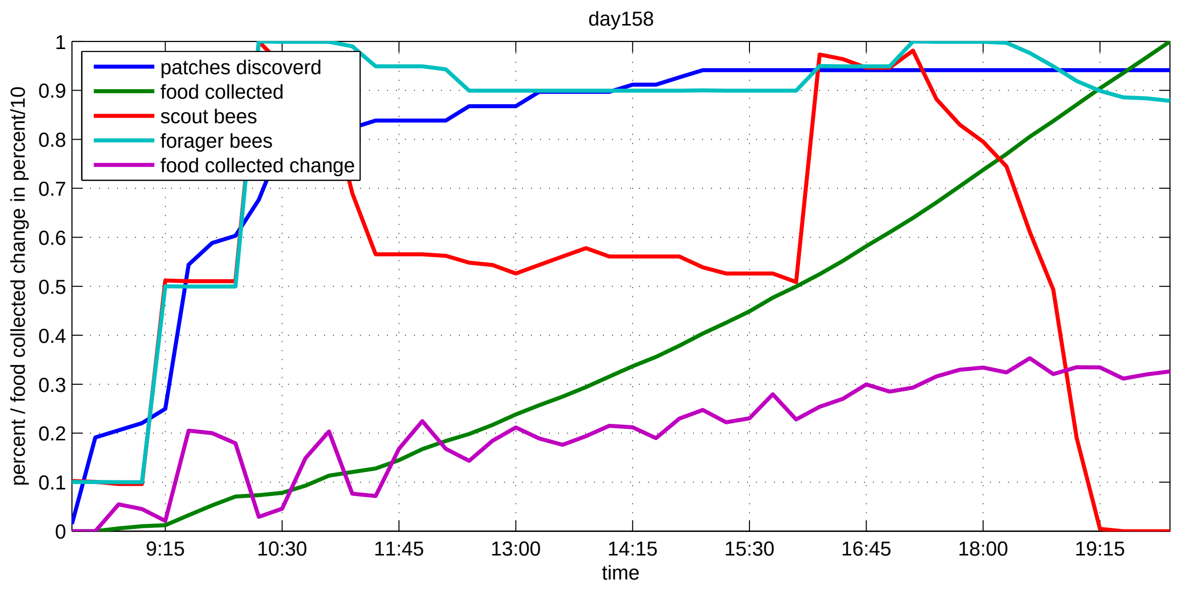


Figure 8: *A summer day from the standard run R1_1*

[2] Peter Corke. *Robotics and Machine Vision Toolboxes for MATLAB (MVTB)*. URL: `https://code.google.com/p/matlab-toolboxes-robotics-vision/`.

[3] R. Dukas. "Mortality rates of honey bees in the wild". In: *Insectes Sociaux* (2008).

[4] D. S. Khoury, B. A. Barron, and R. M. Myerscough. "Modelling Food and Population Dynamics in Honey Bee Colonies". In: *PLOS ONE* (2013).

[5] Simon G. Potts et al. "Global pollinator declines: trends, impacts and drivers". In: *Trends in Ecology Evolution* 25.6 (2010), pp. 345–353. ISSN: 0169-5347. DOI: `http://dx.doi.org/10.1016/j.tree.2010.01.007`.

[6] T. D. Seeley. *Thde Wisdom of the Hive. The Social Physiology of Honey Bee Colonies*. London: Harvard University Press, 1995.

[7] Michel Thomann et al. "Flowering plants under global pollinator decline". In: *Trends in Plant Science* 18.7 (2013), pp. 353–359. ISSN: 1360-1385. DOI: `http://dx.doi.org/10.1016/j.tplants.2013.04.002`.

[8] Walt Wright. *How Many Eggs CAN a Queen Lay?* 2008. URL: `http://www.beesource.com/point-of-view/walt-wright/how-many-eggs-can-a-queen-lay/`.

A Matlab Code

A.1 Properties_Base.m

```
% This is a sample file as reference. For actual simulations, make a copy!

% SYSTEM
% Identifier for data record
Prop.Sys.identifier = 'Properties_Base';
% If parallelization is possible, max. processors that should be used
Prop.Sys.cores = 6;

% WORLD
% See "Wisdom of the Hive", P. 49 why this is a reasonable size for
% simulation, as most flower patches detected will be in this area and the
% 95th percentile is within 6km radius of the hive. This implies 100km^2 is
% sufficient
Prop.Sim.world_size = 10000;             % Meters
% 100m^2 raster size (10m * 10m) is a reasonable scaling for the map as
% flower patches, respective rastered areas can be assumed to be at least
% 100m^2, which was also the seeding size mentioned ("Wisdom of the Hive",
% P. 51)
Prop.Sim.world_size_10 = Prop.Sim.world_size/10;
Prop.Sim.world_file = 'data\equal_dist_normal.png'; % Image file the world is
    based on
Prop.Sim.eval_time_days = 470;           % Days of simulation per run
Prop.Sim.eval_time_seconds = 12*60*60;   % Seconds of simulation per day
Prop.Sim.eval_step_seconds = 60;         % Seconds between two evaluation
    points
Prop.Sim.record_step_seconds = 15*60;    % Daily simulation recording step size

% Flowers
% Describes how vital flowers are during a year
% From 1 to 365, arbitrary measure points + paddings
% Taken from P. 45, "Wisdom of the Hive"
% 2xN vectors, days vs. activity factor
% Peak values are in hive weight change in kg per day.
% Assuming bee count = 10'000, we get peak/10'000 = kg/bee/day
% which * 1000 gives g/bee/day, what we need. Therefore we assume this is
% the weight of pollen/nectar a bee can get per flight and flower.
% Therefore we can take hive weight change/day as indicator for
% flower/pollen quality at a given day in the year.

% Final equation: peak * daily quality * 1/100 = reward per flower visit in
% grams.


% Flowers (mainly active during spring):
```

```
Prop.Sim.Flower(1).peak = 1.8;
Prop.Sim.Flower(1).year_activity = [131:140;sin(linspace(0,pi,10))];
% Woods (mainly active during summer):
Prop.Sim.Flower(2).peak = 6;
Prop.Sim.Flower(2).year_activity = [154:161,175:182,190:195;sin(linspace(0,pi
    ,8)),sin(linspace(0,pi,8))/1.5,sin(linspace(0,pi,6))/6];
% Fruits (mainly active during autumn:
Prop.Sim.Flower(3).peak = 2;
Prop.Sim.Flower(3).year_activity = [245:270;sin(linspace(0,pi,26))];

% Left empty for future use:
Prop.Sim.Flower(4).peak = 0;
Prop.Sim.Flower(4).year_activity = [1:2;0,0];

% Maybe we'll implement smog later on, maybe not...
Prop.Sim.Smog.year_activity = [0,0;0,0];        % TODO

% HIVES
% Hive 1
% Most values from the reference paper
Prop.Sim.hive_count = 1;
Prop.Sim.Hive(1).x_pos = Prop.Sim.world_size_10/2;
Prop.Sim.Hive(1).y_pos = Prop.Sim.world_size_10/2;
Prop.Sim.Hive(1).uncapped_brood = 0;
Prop.Sim.Hive(1).hive_bees = 16000;
Prop.Sim.Hive(1).foragers = 8000;
Prop.Sim.Hive(1).food_brood_eff = 500;
Prop.Sim.Hive(1).hive_brood_eff = 5000;
Prop.Sim.Hive(1).aging_time = 12;
Prop.Sim.Hive(1).mortality = 0.075;
Prop.Sim.Hive(1).min_recruitment = 0.25;
Prop.Sim.Hive(1).max_recruitment = 0.25;
% Describes how the laying rate of the queen varies during the year
% From 1 to 365, arbitrary measure points + paddings
% Taken from P. 34ff, "Wisdom of the Hive"
% Approx. laying from April/May to October, in winter only to keep the hive
% alive
% 2xN vectors, days vs. laying factor
Prop.Sim.Hive(1).laying_function =
    [-30,1,32,60,91,121,153,182,213,244,274,305,335,366,395;0.1,0.1,0.1,0.1,0.5
    ,1,2,2,1,0.5,0.1,0.1,0.1,0.1,0.1];
Prop.Sim.Hive(1).laying_rate = 2000;
Prop.Sim.Hive(1).social_inhibition = 0.75;
Prop.Sim.Hive(1).adult_bee_emerging = 1/9;
Prop.Sim.Hive(1).food_per_forager = 0.1;
Prop.Sim.Hive(1).food = 50000;
% Food consumption adult bees, 0.007g/day
Prop.Sim.Hive(1).food_consumption_adult = 0.007;
% Food consumption brood/larvae, 0.018g/day
Prop.Sim.Hive(1).food_consumption_brood = 0.018;
```

```
% Describes how many bees fly out at a given daytime
% From 08:00 to 20:00, 12 hours, one measure point / hour + paddings
% Taken from P. 86 "Wisdom of the Hive"
% 2xN vector, hours vs. activity
Prop.Sim.Hive(1).daily_activity =  [-1,1,2,3,4,5,6,7,8,9,10,11,12,13;-1,0.1,0.5
    ,1,0.95,0.9,0.9,0.9,0.9,0.95,1,0.5,0.1,-1];
% Barrier for extinction. If total bee count is lower than this, a hive is
% considered extinct (no more computation, all values constant and/or zero)
Prop.Sim.Hive(1).extinct_barrier = 500;
% Max. percentage of foragers that are scouts
% Taken from P. 86 "Wisdom of the Hive", an average value
% 6% or 0.06 (normalized)
Prop.Sim.Hive(1).scout_count = 0.06;

% BEES
% Max. food carried by bee, average value, P. 42 "Wisdom of the Hive"
% 25mg or 0.025g
Prop.Sim.Hive(1).Bee.max_food = 0.025;
% Max. flight distance of a bee
% 8km or 8000m, use 2km padding to world borders in 10m resolution
Prop.Sim.Hive(1).Bee.max_dist = 8000/10;
% Probability to optimize a path during one way
% 50% chance. This is a freely chosen
% property for the random walk algorithm!
Prop.Sim.Hive(1).Bee.optimize_prob = 0.5;
% Factor to change movement angle, multiplied by randn();. This is a freely
% chosen property for the random walk algorithm!
Prop.Sim.Hive(1).Bee.rotate_scale = 0.5;
% Time after which to set a new waypoint (s). This is a freely chosen
% property for the random walk algorithm!
Prop.Sim.Hive(1).Bee.change_waypoint = 60;
% Flight speed
% Taken from P. 47 "Wisdom of the Hive"
% 25km/h or 7m/s, divided by ten as we work in 10m resolution
Prop.Sim.Hive(1).Bee.flight_speed = 7/10;

% An average of 5min spent to evaluate a flower patch
% Assumption made based on flight speed and patch size
Prop.Sim.Hive(1).scouting_eval_time = 3*60;

% An average of 4min spent to unload food and get a new patch assigned,
% respectively doing a waggle dance
% Assumption made based on "Wisdom of the Hive", P. 107f
Prop.Sim.Hive(1).unload_time = 4*60;

% An average of 1min spent to collect food at the flower patch
% Assumption made based on "Wisdom of the Hive", P. 107ff
Prop.Sim.Hive(1).collect_time = 1*60;
```

```
% Maximum amount of bees simulated (if bee count is higher than this,
% aggregated bee clusters will be simulated)
Prop.Sim.Hive(1).max_forager_clusters = 1000;

% Food rate is dynamic (environment simulation)
Prop.Sim.Hive(1).fixed_food_rate = 0;
```

A.2 hive_simulation.m

```
function hive_simulation(proparray)

    fprintf('################### HIVE-SIMULATION ####################\n');

    if(nargin == 0)
        % Ask user to input a properties file for the simulation
        [file_name, path_name] = uigetfile('*.m','Select simulation property
            file');

        fprintf('Loading properties...\n');

        % Construct absolute loading path
        load_properties = strcat(path_name,file_name);

        % Load properties
        run(load_properties);
        proparray = [Prop];
    end

    % Initialize thread pool
    fprintf('Initializing thread pool...\n');
    threads = matlabpool('size');
    cores = feature('numCores');
    if((threads ~= min(cores, proparray(1).Sys.cores)))
        if threads > 0
            matlabpool('close');
        end
        matlabpool('open', min(cores, proparray(1).Sys.cores));
    end

    % Initialize report
    fprintf('Initializing report...\n');
    for k = 1:length(proparray)
        report(k) = Report(proparray(k));
    end

    runtime_start = tic;
    % Parfor loop allows simulating multiple scenarios at once
```

```
parfor k = 1:length(proparray)
    Prop = proparray(k);
    % Initialize world
    fprintf('Initializing world...\n');
    world = World(Prop);

    % Initialize hives
    fprintf('Initializing hives...\n');
    hives = Hive.empty(Prop.Sim.hive_count, 0);
    for i = 1:Prop.Sim.hive_count
        hives(i) = Hive(i, world, Prop);
    end

    % Start simulation
    fprintf('Starting simulation (%d days)...\n', Prop.Sim.eval_time_days);

    reversestr_a = '';
    reversestr_b = '';
    percent_A_old = -1;
    percent_B_old = -1;
    % Counter for indexing data collection
    sim_day = 1;
    for t_d = 1:Prop.Sim.eval_time_days
        % Re-evaluate environment if there is any need for it
        % The need is defined by checking if some flower is active at the
        % moment.
        activity_sum = sum(sum(world.quality_map.array));

        % Check for extinct hives and hives with constant food rate
        dont_calculate_sum = 0;
        for i = 1:Prop.Sim.hive_count
            if(hives(i).fixed_food_rate || hives(i).is_extinct)
                dont_calculate_sum = dont_calculate_sum + 1;
            end
        end

        % There is activity, and there are some hives which are not extinct
        % and don't have constant food rate, so do environment simulation
        if(activity_sum > 0 && dont_calculate_sum ~= Prop.Sim.hive_count)
            dt_s = Prop.Sim.eval_step_seconds;
            for i = 1:Prop.Sim.hive_count
                % Reset environment simulation for new day
                hives(i).reset_s(t_d);
            end
            % Counter for indexing data collection
            sim_second = 1;
            for i = 1:Prop.Sim.hive_count
                % Write corresponding day to this simulated day into the
                % report
                report(k).data.hives(i).day(sim_day).actual_day = t_d;
                % Record data at t_d into field sim_day
                report(k).data.hives(i).day(sim_day).patches_total = hives(
                    i).patches_total;
            end
            for t_s = 1:Prop.Sim.eval_time_seconds
                if(mod(t_s-1,dt_s)==0)
                    for i = 1:Prop.Sim.hive_count
                        if(~hives(i).is_extinct && hives(i).fixed_food_rate
                             == 0)
                            hives(i).simulate_s(t_s, t_d, dt_s);
                        end
                    end
                end
                if(mod(t_s-1,Prop.Sim.record_step_seconds)==0)
                    for i = 1:Prop.Sim.hive_count
                        % Write corresponding time to this simulated time
                            into the
                        % report
                        report(k).data.hives(i).day(sim_day).actual_time(
                            sim_second) = t_s;
                        % Record data at t_s into field sim_second
                        report(k).data.hives(i).day(sim_day)
                            .patches_discovered(sim_second) = hives(i)
                            .patches_discovered;
                        report(k).data.hives(i).day(sim_day).food_sum(
                            sim_second) = hives(i).daily_food_sum(t_d);
                        report(k).data.hives(i).day(sim_day).scouts_count(
                            sim_second) = hives(i).scouts_count;
                        report(k).data.hives(i).day(sim_day).forager_count(
                            sim_second) = hives(i).bees_count;
                    end
                    sim_second = sim_second + 1;
                end
                % TODO: COMMENT OUT AGAIN
%                 if(mod(t_s,240)==0)
%                     hives(1).draw_s();
%                     %world.draw_s();
%                     pause(0.0001);
%                     hives(1).patches_discovered
%                     %t_s
%                 end
                % Print progress
                percent_B = ceil(t_s/Prop.Sim.eval_time_seconds*50);
                if(percent_B ~= percent_B_old)
                    progstr_b = '';
                    for j=1:percent_B
                        progstr_b = strcat(progstr_b,'#');
                    end
                    for j=percent_B:49
                        progstr_b = strcat(progstr_b,'.');
                    end
                    msg_b = sprintf(strcat('Progress (environment): [',
                        progstr_b,']\n'), t_s, Prop.Sim.eval_time_seconds);
                    fprintf([reversestr_b, msg_b]);
                    reversestr_b = repmat(sprintf('\b'), 1, length(msg_b));
                    percent_B_old = percent_B;
                end
            end
            sim_day = sim_day + 1;
        else
           for i = 1:Prop.Sim.hive_count
               % Daily food income (aggregated) is zero if no flower
               % activity
               hives(i).daily_food_sum(t_d) = 0;
           end
        end

        % Daily environment simulation
        world.simulate_d(t_d);
        % Daily hive simulation
        for i = 1:Prop.Sim.hive_count
            hives(i).simulate_d(t_d);
        end

        %world.draw_d();

        % Print progress
        percent_A = ceil(t_d/Prop.Sim.eval_time_days*50);
        if(percent_A ~= percent_A_old)
            progstr_a = '';
            for j=1:percent_A
                progstr_a = strcat(progstr_a,'#');
            end
            for j=percent_A:49
                progstr_a = strcat(progstr_a,'.');
            end
            msg_a = sprintf(strcat('Progress (hive): [',progstr_a,']\n'),
                t_d, Prop.Sim.eval_time_days);
            fprintf([reversestr_a, msg_a]);
            reversestr_a = repmat(sprintf('\b'), 1, length(msg_a));
            percent_A_old = percent_A;
        end

    end
    fprintf('\n');

    % Record simulation data

    for i = 1:Prop.Sim.hive_count
        report(k).data.hives(i).uncapped_brood = hives(i).B;
```

```
            report(k).data.hives(i).hive_bees = hives(i).H;
            report(k).data.hives(i).food_storage = hives(i).f;
            report(k).data.hives(i).foragers = hives(i).F;
            report(k).data.hives(i).daily_food_sum = hives(i).daily_food_sum;
        end

    end
    runtime_stop = toc(runtime_start);

    fprintf('Saving report...\n');
    for k = 1:length(proparray)
        report(k).save();
    end

    fprintf('Simulation finished. Used %fs.\n',runtime_stop);
    fprintf('#########################################################\n');
end
```

A.3 World.m

```
classdef World < handle

    properties
        prop;
        % map with flowers, smog source, dimensions N/10 x N/10
        type_map
        % map with flower quality, smog emission intensity, N/10 x N/10
        % current state
        quality_map
        % peak state
        maxquality_map
        % Flower vitality, 365 values, 4 flowers
        flower
    end

    methods
        % Constructor
        function obj = World(Prop)
            obj.prop = Prop;
            [obj.type_map, obj.quality_map, obj.maxquality_map] = generate_maps
                (Prop);

            % Convert time area vector from properties (2xN) to 1x365 values
               for the year and cancel out negative values:
            obj.flower(1).year_activity = zeros(1,365);
            obj.flower(1).peak = Prop.Sim.Flower(1).peak;
            obj.flower(1).year_activity(Prop.Sim.Flower(1).year_activity(1,:))
                = max(Prop.Sim.Flower(1).year_activity(2,:),0);
            obj.flower(2).year_activity = zeros(1,365);
            obj.flower(2).peak = Prop.Sim.Flower(2).peak;
            obj.flower(2).year_activity(Prop.Sim.Flower(2).year_activity(1,:))
                = max(Prop.Sim.Flower(2).year_activity(2,:),0);
            obj.flower(3).year_activity = zeros(1,365);
            obj.flower(3).peak = Prop.Sim.Flower(3).peak;
            obj.flower(3).year_activity(Prop.Sim.Flower(3).year_activity(1,:))
                = max(Prop.Sim.Flower(3).year_activity(2,:),0);
            obj.flower(4).year_activity = zeros(1,365);
            obj.flower(4).peak = Prop.Sim.Flower(4).peak;
            obj.flower(4).year_activity(Prop.Sim.Flower(4).year_activity(1,:))
                = max(Prop.Sim.Flower(4).year_activity(2,:),0);

            % Looking if this is similar to P. 45 "Wisdom of the Hive", for
            % debug purposes ONLY:
            % bar(obj.flower(1).year_activity*Prop.Sim.Flower(1).peak);
            % hold on
            % bar(obj.flower(2).year_activity*Prop.Sim.Flower(2).peak);
            % hold on
            % bar(obj.flower(3).year_activity*Prop.Sim.Flower(3).peak);
            % hold on
            % bar(obj.flower(4).year_activity*Prop.Sim.Flower(4).peak);
            % pause(100);
        end

        function simulate_d(obj, t_d)
            update_quality_map(obj, t_d);
        end

        function update_quality_map(obj, t_d)
            % Write current quality depending on flower type and time of the
                year
            i = find(obj.type_map.array==1);
            obj.quality_map.array(i) = obj.maxquality_map.array(i) * obj.flower
                (1).year_activity(mod(t_d - 1, 365) + 1);
            i = find(obj.type_map.array==2);
            obj.quality_map.array(i) = obj.maxquality_map.array(i) * obj.flower
                (2).year_activity(mod(t_d - 1, 365) + 1);
            i = find(obj.type_map.array==3);
            obj.quality_map.array(i) = obj.maxquality_map.array(i) * obj.flower
                (3).year_activity(mod(t_d - 1, 365) + 1);
            i = find(obj.type_map.array==4);
            obj.quality_map.array(i) = obj.maxquality_map.array(i) * obj.flower
                (4).year_activity(mod(t_d - 1, 365) + 1);
        end


        function draw_map(obj)
```

```
            %flush
            clf
            %initialize image matrix
            hsv_image_map = ones(obj.prop.Sim.world_size/10,
                obj.prop.Sim.world_size/10,3);

            %fill it with the given values
            hsv_image_map(:,:,1) = obj.type_map.array/6;
            hsv_image_map(:,:,3) = obj.quality_map.array;

            %convert hsv2rgb
            image_map = hsv2rgb(hsv_image_map);

            %correct 'rounding errors' from hsv2rgb
            image_map = min(max(image_map,0),1);

            %display image
            %subplot(1,2,1);
            image(image_map)
            axis square
            %handle = gcf;
            pause(0.001);
        end

        function draw_d(obj)
            draw_map(obj);
        end

    end


end
```

A.4 Hive.m

```
classdef Hive < handle

    properties
        prop;                       % Properties
        hive_ind;                   % Hive index in the world
        world;                      % Handle on the world
        max_sim_time;               % Simulation length

        B;                          % Uncapped brood
        H;                          % Hive bees
        f;                          % Food storage
        F;                          % Foragers
        b;                          % Food effect on brood survival
        v;                          % Hive bees effect on brood survival
        tau;                        % Aging time in days (12 days [2])
        amin;                       % Minimum recruitment
        amax;                       % Food dependent recruitment
        L;                          % Laying rate of the queen
        L_year;                     % Laying rate change over the year
        m;                          % Forager death rate
        sigma;                      % Social inhibition strength
        phi;                        % Adult bee emerging factor
        c;                          % Food per forager and day, will vary with
            advanced model!
        coFH;                       % Food consumption of adult bees (g)
        coB;                        % Food consumption of brood (g)

        S;                          % Brood survival rate
        R;                          % Recruitment function

        extinct_barrier;            % Min. bees required for the hive to survive
        is_extinct;                 % Hive extinct or not

        % For realtime environment simulation
        daily_activity;             % Activity during the day

        x_pos;                      % X-position in the world
        y_pos;                      % Y-position in the world

        foragers;                   % Array of forager bees
        scouts;                     % Array of scout bees

        waggle_paths;               % Available waggles to copy
        waggle_data;                % Array for fast waggle evaluation
        waggle_count;               % Current waggle count

        scouts_count;               % Current scout count
        max_scout_percent;          % Max. percent of active bees that are scouts
        bees_count;                 % Current active bees

        max_forager_clusters;       % Amount of forager bees that can be simulated
             in total (clustering)
        cluster_size                % Size of a bee cluster

        beemap;                     % Rasterized map of bees (N/10xN/10, where N
            is the world size)
        patches;                    % Map of already discovered flower patches

        % We add a barrier to stop scouting bees as soon as all patches
        % have been discovered, which is not something bees would do. But
        % we can do it here as the scout bees don't have any effect after
        % they found every patch, but it's computationally very expensive
        % to keep them in the random walk algorithm.
        patches_total;              % Total patches
        patches_discovered;         % Discovered patches

        daily_food_sum;             % Aggregated collected food per day
        fixed_food_rate;            % Fixed rate (original model) or advanced
            model

    end

    methods
        % Constructor
        function obj = Hive(hive_index, world, Prop)
            obj.prop = Prop;

            obj.hive_ind = hive_index;

            obj.world = world;
            obj.x_pos = Prop.Sim.Hive(obj.hive_ind).x_pos;
            obj.y_pos = Prop.Sim.Hive(obj.hive_ind).y_pos;

            obj.is_extinct = 0;

            obj.fixed_food_rate = Prop.Sim.Hive(obj.hive_ind).fixed_food_rate;
            obj.max_sim_time = Prop.Sim.eval_time_days;

            obj.B = zeros(1,obj.max_sim_time);
            obj.H = zeros(1,obj.max_sim_time);
            obj.f = zeros(1,obj.max_sim_time);
            obj.F = zeros(1,obj.max_sim_time);

            obj.daily_food_sum = zeros(1,obj.max_sim_time);

            obj.B(1) = Prop.Sim.Hive(obj.hive_ind).uncapped_brood;
            obj.H(1) = Prop.Sim.Hive(obj.hive_ind).hive_bees;
            obj.f(1) = Prop.Sim.Hive(obj.hive_ind).food;
            obj.F(1) = Prop.Sim.Hive(obj.hive_ind).foragers;

            obj.b = Prop.Sim.Hive(obj.hive_ind).food_brood_eff;
            obj.v = Prop.Sim.Hive(obj.hive_ind).hive_brood_eff;
            obj.tau = Prop.Sim.Hive(obj.hive_ind).aging_time;
            obj.amin = Prop.Sim.Hive(obj.hive_ind).min_recruitment;
            obj.amax = Prop.Sim.Hive(obj.hive_ind).max_recruitment;
            obj.L = Prop.Sim.Hive(obj.hive_ind).laying_rate;

            % Load x values of measure points for laying rate change
            L_year_x = Prop.Sim.Hive(obj.hive_ind).laying_function(1,:);

            % Load y values of measure points for laying rate change
            L_year_y = Prop.Sim.Hive(obj.hive_ind).laying_function(2,:);
```

```
% Interpolate values of laying rate change for 365 days
obj.L_year = interpolate_values(L_year_x,L_year_y,1,1,365,0,1);

obj.m = Prop.Sim.Hive(obj.hive_ind).mortality;
obj.sigma = Prop.Sim.Hive(obj.hive_ind).social_inhibition;
obj.phi = Prop.Sim.Hive(obj.hive_ind).adult_bee_emerging;
obj.c = Prop.Sim.Hive(obj.hive_ind).food_per_forager;

obj.coFH = Prop.Sim.Hive(obj.hive_ind).food_consumption_adult;
obj.coB = Prop.Sim.Hive(obj.hive_ind).food_consumption_brood;

obj.extinct_barrier = Prop.Sim.Hive(obj.hive_ind).extinct_barrier;
obj.max_scout_percent = Prop.Sim.Hive(obj.hive_ind).scout_count;

obj.max_forager_clusters = Prop.Sim.Hive(obj.hive_ind)
    .max_forager_clusters;
obj.cluster_size = 1;

obj.scouts = Bee.empty(round(obj.F(1)*obj.max_scout_percent), 0);

obj.foragers = Bee.empty(obj.max_forager_clusters, 0);

% Instantiate bees
for i=1:round(obj.F(1)*obj.max_scout_percent)
    obj.scouts(i) = Bee(obj, obj.world, obj.prop);
end

for i=1:obj.max_forager_clusters
    obj.foragers(i) = Bee(obj, obj.world, obj.prop);
end

% Interpolate daily activity
daily_activity_x = Prop.Sim.Hive(obj.hive_ind).daily_activity(1,:);
daily_activity_y = Prop.Sim.Hive(obj.hive_ind).daily_activity(2,:);
obj.daily_activity = interpolate_values(daily_activity_x,
    daily_activity_y,1,1,12,0,1);

% Initialize empty beemap
obj.beemap = Map(0,obj.prop.Sim.world_size_10,
    obj.prop.Sim.world_size_10);

% Initialize patches map
obj.patches = Map(0,obj.prop.Sim.world_size_10,
    obj.prop.Sim.world_size_10);

% Brood survival rate
obj.S = @(H, f) f^2/(f^2 + obj.b^2) * H/(H+obj.v);

% Recruitment function
```

```
            obj.R = @(H, F, f) obj.amin + obj.amax*(obj.b^2/(obj.b^2+f^2)) -
                obj.sigma * (F/(F+H));
        end

        % Count active patches
        function count = count_patches(obj)
            % Maps to logic maps conversion
            logic_quality_map = logical(obj.world.quality_map.array);
            logic_type_map = logical(obj.world.type_map.array);
            logic_map = logic_quality_map & logic_type_map;
            [sm,sn] = size(logic_map);
            check_map = zeros(sm,sn);
            count = 0;
            % Go through map
            for y = 1:sm
                for x = 1:sn
                    % Check map to avoid double counting
                    if(check_map(y,x) == 0 && logic_map(y,x) == 1)
                        patch = bwselect(logic_map,x,y);
                        ind = find(patch);
                        check_map(ind) = 1;
                        count = count + 1;
                    end
                end
            end
        end

        % Compare own food source against all other food sources
        function prob = food_source_compare(obj, path)
            if(obj.waggle_count > 0)
                % Calculating relative probability compared to other
                % patches
                prob = path.patch_quality/(sum(obj.waggle_data(1:
                    obj.waggle_count,2))/obj.waggle_count) * ...
                    path.patch_size/(sum(obj.waggle_data(1:obj.waggle_count,3))
                        /obj.waggle_count) * ...
                    path.distance/(sum(obj.waggle_data(1:obj.waggle_count,4))/
                        obj.waggle_count);
            else
                % It's the first waggle we test against, so it's the best
                prob = 1;
            end
        end

        % Register a waggle dance after arriving at the hive
        function register_waggle_dance(obj, path, prob)
            obj.waggle_count = obj.waggle_count + 1;
            % Enter path
            obj.waggle_paths(obj.waggle_count) = path;
            % Enter data in separate array (for speed)
            obj.waggle_data(obj.waggle_count, :) = [prob, path.patch_quality,
                path.patch_size, path.distance];
            % Normalize probabilities of accepting a waggle dance
            if(prob > 1)
                obj.waggle_data(1:obj.waggle_count,1) = obj.waggle_data(1:
                    obj.waggle_count,1)/prob;
            end
        end

        % Mark a flower patch, report quality and size of the patch
        function [psize, pquality, ptype] = mark_flower_patch(obj, x, y)
            % Get flower patch that our scout bee discovered
            patch = bwselect(logical(obj.world.type_map.array),x,y);
            % Find nonzero fields, get indexes
            ind = find(patch);
            % Calculate patch size in 100m^2 units, 10^2m^2 per index
            psize = length(ind);
            % Calcualte average quality of the patch
            pquality = sum(obj.world.quality_map.array(ind))/psize;
            % Get flower type
            ptype = obj.world.type_map.array(y,x);
            % Mark entire adjacent flower patch (segment, object)
            obj.patches.array(ind) = 1;
            % Add flower patch to counting variable
            obj.patches_discovered = obj.patches_discovered + 1;
        end

        % Watch a waggle dance and maybe use it
        function [got_path, path] = watch_waggle(obj)
            path = 0;
            % Select one, randomly
            if(obj.waggle_count > 0)
                i = randi([1,obj.waggle_count],1);
                % Bee decided that the patch is good enough
                if(rand() < obj.waggle_data(i,1))
                    got_path = 1;
                    path = obj.waggle_paths(i);
                else
                    got_path = 0;
                end
            else
                got_path = 0;
            end
        end


        % Iterative daily simulation step
        function simulate_s(obj, t_s, t_d, dt_s)
            obj.waggle_count = 0;
%             for i = 1:round(obj.F(t_d)*obj.max_scout_percent)
%                  % Don't record bees that don't work
%                  if(obj.scouts(i).work_mode ~= 0)
%                      % Add bee record at current position
%                      x = ceil(obj.scouts(i).x_pos);
%                      y = ceil(obj.scouts(i).y_pos);
%                      obj.beemap.array(y,x) = obj.beemap.array(y,x) - 1;
%                  end
%              end
%              for i = 1:obj.max_forager_clusters
%                  % Don't record bees that don't work
%                  if(obj.foragers(i).work_mode ~= 0)
%                      % Add bee record at current position
%                      x = ceil(obj.foragers(i).x_pos);
%                      y = ceil(obj.foragers(i).y_pos);
%                      obj.beemap.array(y,x) = obj.beemap.array(y,x) -
    obj.cluster_size;
%                  end
%              end

            % Scouts working loop
            % Work only if there is a chance to discover a new
            % flower patch (not realistic but doesn't numerically
            % change the foraging result and saves a lot of
            % computation power)
            if(obj.patches_total > obj.patches_discovered)
                for i = 1:round(obj.F(t_d)*obj.max_scout_percent)
                    % Assign jobs to scouts
                    if(obj.scouts(i).work_mode == 0)
                         % More active bees possible?
                         if(obj.bees_count < obj.F(t_d)*obj.daily_activity(ceil
                             (t_s/3600)))
                            % More scouts possible?
                            if(obj.scouts_count < obj.F(t_d)*obj.daily_activity
                                (ceil(t_s/3600))*obj.max_scout_percent)
                                obj.scouts_count = obj.scouts_count + 1;
                                obj.bees_count = obj.bees_count + 1;
                                % Assign scout job
                                obj.scouts(i).work_mode = 1;
                                obj.scouts(i).path = Path();
                                % Starting point in the hive
                                obj.scouts(i).path.append(obj.x_pos,obj.y_pos);
                                % Random starting direction from the hive
                                obj.scouts(i).alpha = rand()*2*pi;
                            end
                         end
                    end
                    obj.scouts(i).work(t_d, t_s, dt_s);
                end
            end
            % Foragers working loop
            for i = 1:obj.max_forager_clusters
                % Assign jobs to scouts
                if(obj.foragers(i).work_mode == 0)
                     % More active bees possible?
                     if(obj.bees_count < obj.F(t_d)*obj.daily_activity(ceil(t_s
                         /3600)))
                        obj.foragers(i).work_mode = 11;
                        obj.bees_count = obj.bees_count + obj.cluster_size;
                     end
                else
                    obj.foragers(i).work(t_d, t_s, dt_s);
                end
            end

%             for i = 1:round(obj.F(t_d)*obj.max_scout_percent)
%                 % Don't record bees that don't work
%                 if(obj.scouts(i).work_mode ~= 0)
%                     % Add bee record at current position
%                     x = ceil(obj.scouts(i).x_pos);
%                     y = ceil(obj.scouts(i).y_pos);
%                     obj.beemap.array(y,x) = obj.beemap.array(y,x) + 1;
%                 end
%             end
%             for i = 1:obj.max_forager_clusters
%                 % Don't record bees that don't work
%                 if(obj.foragers(i).work_mode ~= 0)
%                     % Add bee record at current position
%                     x = ceil(obj.foragers(i).x_pos);
%                     y = ceil(obj.foragers(i).y_pos);
%                     obj.beemap.array(y,x) = obj.beemap.array(y,x) +
    obj.cluster_size;
%                 end
%             end
        end

        % Reset daily simulation parameters
        function reset_s(obj, t_d)
            % Reset bees
            for i=1:obj.max_forager_clusters
                obj.foragers(i).work_mode = 0;
                obj.foragers(i).work_time = 0;
                obj.foragers(i).food = 0;
                obj.foragers(i).time_counter = 0;
                obj.foragers(i).x_pos = obj.x_pos;
                obj.foragers(i).y_pos = obj.y_pos;
            end
            for i=1:round(obj.F(t_d)*obj.max_scout_percent)
                obj.scouts(i).work_mode = 0;
                obj.scouts(i).work_time = 0;
                obj.scouts(i).food = 0;
                obj.scouts(i).time_counter = 0;
                obj.scouts(i).x_pos = obj.x_pos;
                obj.scouts(i).y_pos = obj.y_pos;
            end
            % Reset hive
            obj.cluster_size = round(obj.F(t_d)/obj.max_forager_clusters);
            obj.bees_count = 0;
            obj.scouts_count = 0;
            obj.daily_food_sum(t_d) = 0;
            % Allocate enough space for waggle dances
            obj.waggle_paths = Path.empty(round(obj.F(t_d)*
                obj.max_scout_percent) + obj.max_forager_clusters,0);
            obj.waggle_data = zeros(round(obj.F(t_d)*obj.max_scout_percent) +
                obj.max_forager_clusters,4);
            obj.waggle_count = 0;
            obj.patches = Map(0,obj.prop.Sim.world_size_10,
                obj.prop.Sim.world_size_10);
            % Count total patches, discovered patches = 0
            obj.patches_total = obj.count_patches();
            obj.patches_discovered = 0;
        end

        % Iterative simulation step
        function simulate_d(obj, t_d)
            if(~obj.is_extinct)
                % Function handles
                % Brood change rate
                dB = @(t) obj.L_year(mod(t_d - 1, 365) + 1) * obj.L * obj.S(
                    obj.H(t), obj.f(t)) - obj.phi * obj.B(t);

                % Rate of change of hive bees > tau
                dH = @(t) obj.phi * obj.B(t - obj.tau) - obj.H(t) * obj.R(obj.H
                    (t), obj.F(t), obj.f(t));

                % Rate of change of hive bees <= tau
                dHs = @(t) - obj.H(t) * obj.R(obj.H(t), obj.F(t), obj.f(t));

                % Rate of change of foragers
                dF = @(t) obj.H(t) * obj.R(obj.H(t), obj.F(t), obj.f(t)) -
                    obj.m*obj.F(t);

                % Food change rate
                if(obj.fixed_food_rate == 1)
                    % Fixed rate mode from original model
                    df = @(t) obj.c * obj.F(t) - obj.coFH * (obj.F(t) + obj.H(t
                        )) - obj.coB * obj.B(t);
                else
                    % Environment dependent mode from advanced model
                    df = @(t) obj.daily_food_sum(t) - obj.coFH * (obj.F(t) +
                        obj.H(t)) - obj.coB * obj.B(t);
                end

                % Calculate t+1
                obj.B(t_d + 1) = max(obj.B(t_d) + dB(t_d), 0);
                obj.F(t_d + 1) = max(obj.F(t_d) + dF(t_d), 0);

                % Hive bees can only emerge if starting brood was more than tau
                     days ago
                if(t_d > obj.tau)
                    obj.H(t_d + 1) = max(obj.H(t_d) + dH(t_d), 0);
                else
                    obj.H(t_d + 1) = max(obj.H(t_d) + dHs(t_d), 0);
                end
                obj.f(t_d + 1) = max(obj.f(t_d) + df(t_d), 0);

                % Check if enough bees instantiated, increase if needed
                if(length(obj.scouts) < round(obj.F(t_d+1)*
                    obj.max_scout_percent))
                    new_scouts = Bee.empty(round(obj.F(t_d+1)), 0);
                    new_scouts(1:length(obj.scouts)) = obj.scouts(:);
                    for i=length(obj.scouts)+1:round(obj.F(t_d+1))
                        new_scouts(i) = Bee(obj, obj.world, obj.prop);
                    end
                    obj.scouts = new_scouts;
                end

                % Hive extinct due to a too low bee count that cannot run a
                    hive
                if(obj.B(t_d+1)+obj.F(t_d+1) < obj.extinct_barrier)
                    obj.is_extinct = 1;
                end
            else
                % Hive extinct. No bees in the hive
                obj.B(t_d + 1) = 0;
                obj.F(t_d + 1) = 0;
                obj.H(t_d + 1) = 0;
                % Food does not change anymore
                obj.f(t_d + 1) = obj.f(t_d);
            end
        end

        function draw_s(obj)
            clf
            colormap('hot')
            %subplot(1,2,1);
            %imagesc(obj.patches.array);
            %subplot(1,2,2);
            imagesc(logical(obj.beemap.array));
```

```
            colorbar
        end


        function plot(obj)
            subplot(2,1,1);
            plot(obj.B,'r-','LineWidth', 2);
            hold on
            plot(obj.F,'g-','LineWidth', 2);
            hold on
            plot(obj.H,'b-','LineWidth', 2);
            hold on
            legend('Uncapped Brood','Forager Bees','Hive Bees')
            xlabel('days')
            ylabel('count')
            subplot(2,1,2);
            plot(obj.f,'k-','LineWidth', 2);
            hold on
            legend('Stored Food')
            xlabel('days')
            ylabel('grams')
        end
    end
end
```

A.5 Bee.m

```
classdef Bee < handle
    properties
        prop;                      % Properties
        hive;                      % Hive the bee belongs to
        world;                     % The world the bee lives in
        % Current work mode
        % 0: no job

        % 1: scout, searching flower patch
        % 2: scout, evaluating flower patch
        % 3: scout, returning to hive
        % 4: scout, waggle dance

        % 11: forager, getting job
        % 12: forager, going to food source
        % 13: forager, collecting food at source
        % 14: forager, way back home
        % 15: forager, food unload + waggle dance

        work_mode;
        % work time, time spent in current work state
        work_time;
        % wait time, time to spend in next work mode
        wait_time;
        % time stepping counter
        time_counter;
        % current food carrying
        food;
        % cluster size of this bee
        % This bee can carry the cluster size times the normal food of a
        % bee. One cluster bee stands for this amount of normal bees
        x_pos;                      % X—position in the world
        y_pos;                      % Y—position in the world
        alpha;                      % Flight direction angle
        path;                       % Flying path (waypoints)
        speed;                      % Flying speed
        max_food;                   % Max. food carry capacity
        rotate_scale;               % Angle scale for scouting
        max_dist;                   % Max. flight distance
        optimize_prob;              % Path optimization probability
        change_waypoint;            % Time after which to set a new waypoint (s)
        current_waypoint;           % Current waypoint for calculations
        scouting_eval_time;         % Time for evaluating a flower patch
        unload_time;                % Time to unload and get a new job
        collect_time;               % Time spent to collect food
    end


    methods
        % Constructor
        function obj = Bee(hive, world, Prop)
            obj.prop = Prop;
            obj.world = world;
            obj.hive = hive;
            obj.x_pos = hive.x_pos;
            obj.y_pos = hive.y_pos;
            obj.work_mode = 0;
            obj.work_time = 0;
            obj.time_counter = 0;
            obj.speed = Prop.Sim.Hive(obj.hive.hive_ind).Bee.flight_speed;
            obj.max_food = Prop.Sim.Hive(obj.hive.hive_ind).Bee.max_food;
            obj.max_dist = Prop.Sim.Hive(obj.hive.hive_ind).Bee.max_dist;
            obj.rotate_scale = Prop.Sim.Hive(obj.hive.hive_ind)
                .Bee.rotate_scale;
            obj.optimize_prob = Prop.Sim.Hive(obj.hive.hive_ind)
                .Bee.optimize_prob;
            obj.change_waypoint = Prop.Sim.Hive(obj.hive.hive_ind)
                .Bee.change_waypoint;
            obj.scouting_eval_time = Prop.Sim.Hive(obj.hive.hive_ind)
                .scouting_eval_time;
            obj.unload_time = Prop.Sim.Hive(obj.hive.hive_ind).unload_time;
            obj.collect_time = Prop.Sim.Hive(obj.hive.hive_ind).collect_time;
            obj.food = 0;
        end

        % Most complicated part of the simulation,
        % bee agent based work algorithm with clustering
        function work(obj, t_d, t_s, dt_s)
            obj.work_time = obj.work_time + dt_s;
            switch obj.work_mode
                case 0
                    % Do nothing until job gets assigned
                case 1
                    % Scouting
                    if(obj.path.distance < obj.max_dist)
                        if(obj.time_counter >= obj.change_waypoint)
                            obj.time_counter = 0;
                            obj.path.append(obj.x_pos,obj.y_pos);
                            % Change in angle
                            obj.alpha = obj.alpha+obj.rotate_scale*randn();
                        end
                        % Last (rounded) position
                        y_min = ceil(obj.y_pos);
                        x_min = ceil(obj.x_pos);
                        % Random walk with border protection
                        obj.x_pos = min(max(obj.x_pos + cos(obj.alpha)*
                            obj.speed*dt_s,1),obj.prop.Sim.world_size_10);
                        obj.y_pos = min(max(obj.y_pos + sin(obj.alpha)*
                            obj.speed*dt_s,1),obj.prop.Sim.world_size_10);
                        obj.time_counter = obj.time_counter + dt_s;
                        % Current (rounded) position
                        y_max = ceil(obj.y_pos);
                        x_max = ceil(obj.x_pos);
                        % Get all points we passed on our way (bresenham line
                        % algorithm)
                        [y_points,x_points] = bresenham(x_min,y_min,x_max,y_max
                            );
                        % Flower patch discovered. A patch is discovered when:
                        % - Quality is above zero (flowers are currently
                            growing)
                        % - Typemap indicates that there is a flower patch
                        % - No other bee has marked this patch already
                        inds = sub2ind([obj.prop.Sim.world_size_10,
                            obj.prop.Sim.world_size_10],y_points,x_points);
                        if(sum(obj.world.quality_map.array(inds)) > 0)
                            ind = inds(find(obj.world.quality_map.array(inds)
                                ,1,'first'));
                            if(obj.world.type_map.array(ind) > 0 && ...
                                obj.world.type_map.array(ind) < 5 && ...
                                obj.hive.patches.array(ind) == 0)
                                % Mark the new patch, transition to new working
                                % state, report quality and size
                                [y,x] = ind2sub([obj.prop.Sim.world_size_10,
                                    obj.prop.Sim.world_size_10], ind);
                                [psize, pquality, ptype] =
                                    obj.hive.mark_flower_patch(x,y);
                                obj.path.patch_size = psize;
                                obj.path.patch_quality = pquality;
                                obj.path.patch_type = ptype;
                                % Last path point is the point where the flower
                                % patch has been discovered
                                obj.path.append(x,y);
                                obj.work_mode = 2;
                                % Reset work time for patch evaluation,
                                % include random (normal distributed) variation
                                obj.wait_time = (abs(randn())+0.5) *
                                    obj.scouting_eval_time;
                                obj.work_time = 0;
                            end
                        end
                    else
                        % Scouting unsuccessful, return to hive
                        obj.work_mode = 2;
                        % Wait time = 0, so that no patch evaluation time
                        % is taken into account in the next work mode
                        obj.wait_time = 0;
                        obj.work_time = 0;
                    end
                case 2
                    % Return to hive as soon as evaluating time is over
                    if(obj.work_time >= obj.wait_time)
                        % Optimize path if probability says so
                        if(rand() <= obj.optimize_prob)
                            obj.path = obj.path.copy(1);
                        end
                        obj.current_waypoint = obj.path.length;
                        obj.work_mode = 3;
                    end

                case 3
                    % Fly back to hive
                    obj.move(−1,dt_s);
                    % Hive is reached, change work mode
                    if(obj.current_waypoint == 1)
                        obj.work_mode = 4;
                        obj.work_time = 0;
                        obj.wait_time = (abs(randn())+0.5) * obj.unload_time;
                    end

                case 4
```

```
        % Scout does waggle dance
        % At hive, waggle dance and continue
        prob = obj.hive.food_source_compare(obj.path);
        % Bee decides if waggle dance or not (if scouting was
        % successful)
        if(obj.path.patch_type > 0)
            % Do waggle dance
            obj.hive.register_waggle_dance(obj.path, prob);
        end
        if(obj.work_time >= obj.wait_time)
            % Bee becomes unemployed again
            obj.work_mode = 0;
            obj.hive.scouts_count = obj.hive.scouts_count - 1;
            obj.hive.bees_count = obj.hive.bees_count - 1;
        end

    case 11
        [got_path, obj.path] = obj.hive.watch_waggle();
        % Bee decided that the patch is good enough
        if(got_path == 1)
            obj.current_waypoint = 1;
            obj.work_mode = 12;
        end

    case 12
        % Fly to flower patch
        obj.move(1,dt_s);
        % Flower patch is reached, change work mode
        if(obj.current_waypoint == obj.path.length)
            obj.food = min(obj.max_food,obj.world.flower(
                obj.path.patch_type).peak*obj.path.patch_quality
                /100);
            obj.work_mode = 13;
            % Reset work time for foraging at flower patch,
            % include random (normal distributed) variation
            obj.work_time = 0;
            obj.wait_time = (abs(randn())+0.5) * obj.collect_time;
        end

    case 13
        % Return to hive as soon as foraging time is over
        if(obj.work_time >= obj.wait_time)
            % Optimize path if probability says so
            if(rand() <= obj.optimize_prob)
                obj.path = obj.path.copy(1);
            end
            obj.current_waypoint = obj.path.length;
            obj.work_mode = 14;
        end
```

```
            case 14
                % Fly back to hive
                obj.move(-1,dt_s);
                % Hive is reached, change work mode
                if(obj.current_waypoint == 1)
                    % Unload food to hive
                    obj.hive.daily_food_sum(t_d) = obj.hive.daily_food_sum(
                        t_d) + ...
                    obj.hive.cluster_size * obj.food;
                    obj.food = 0;
                    obj.work_mode = 15;
                    obj.work_time = 0;
                    obj.wait_time = (abs(randn())+0.5) * obj.unload_time;
                end

            case 15
                % At hive, waggle dance and continue
                prob = obj.hive.food_source_compare(obj.path);
                % Bee decides if waggle dance or not
                if(rand() < prob)
                    % Do waggle dance
                    obj.hive.register_waggle_dance(obj.path, prob);
                end
                if(obj.work_time >= obj.wait_time)
                    % Move on to next working state
                    if(rand() > prob)
                        % Abandon food source, become unemployed
                        obj.work_mode = 0;
                        obj.hive.bees_count = obj.hive.bees_count -
                            obj.hive.cluster_size;
                    else
                        % Continue using this food source
                        obj.work_mode = 12;
                        % Optimize path if probability says so
                        if(rand() <= obj.optimize_prob)
                            obj.path = obj.path.copy(1);
                        end
                    end
                end

            otherwise
        end
    end


    function move(obj, direction, dt_s)
        % Calculate distance that can be passed in the calculation step
        % time (dt_s with speed)
        total_dist = obj.speed * dt_s;
        % As long as not all 'total_distance' is used & target not
```

```
            % reached
            while(total_dist > 0 && ...
                    ((direction == 1 && obj.current_waypoint < obj.path.length)
                         || ...
                    ((direction == -1 && obj.current_waypoint > 1))))
                % Calculating distance from current position to the next
                % waypoint
                distance = norm([obj.x_pos;obj.y_pos] - obj.path.waypoints(:,
                    obj.current_waypoint+direction),2);
                % Calculating remainder after reaching that point
                remainder_dist = total_dist - distance;
                if(remainder_dist > 0)
                    % Remainder is bigger than distance to next waypoint, we
                    % can go one further and loop
                    obj.current_waypoint = obj.current_waypoint + direction;
                    total_dist = remainder_dist;
                else
                    % Remainder not sufficient to reach the next waypoint,
                    % move closer to the next waypoint
                    factor = total_dist/distance;
                    obj.x_pos = obj.x_pos + factor * (obj.path.waypoints(1,
                        obj.current_waypoint+direction) - obj.x_pos);
                    obj.y_pos = obj.y_pos + factor * (obj.path.waypoints(2,
                        obj.current_waypoint+direction) - obj.y_pos);
                    total_dist = 0;
                end
            end
        end
    end
end
```

A.6 Map.m

```
classdef Map < handle

    properties
        array
    end

    methods
        % Constructor
        function obj = Map(is_sparse, m, n)
            if(is_sparse)
                obj.array = sparse(m, n);
            else
                obj.array = zeros(m, n);
            end
        end
    end

end
```

A.7 Path.m

```
classdef Path < handle

    properties
        waypoints               % 2 x n matrix
        patch_size              % Patch size (m^2)
        patch_quality           % Average patch quality (normalized percents)
        patch_type              % Flower type of the patch
        length                  % Save length of path explicitly
        distance                % Compute and save distance
    end

    methods
        % Constructor
        function obj = Path()
            obj.length = 0;
            obj.distance = 0;
            obj.patch_quality = 0;
            obj.patch_size = 0;
            obj.patch_type = 0;
        end

        % Copy the path to a new path and copy properties, probably with
        % optimization, leave old path untouched
        function new_obj = copy(obj, optimize)
            [~, old_n] = size(obj.waypoints);
            % skip every second waypoint to shorten the path
            if((optimize == 1) && (obj.length ~= 2))
                new_obj = Path();
                new_obj.patch_size = obj.patch_size;
                new_obj.patch_quality = obj.patch_quality;
                new_obj.patch_type = obj.patch_type;
                new_n = floor((old_n-2)/2)+2;
                new_obj.waypoints = zeros(2,new_n);
                % copy every 2nd waypoint, preserve start and end point
                inserts = obj.waypoints(:,1:2:old_n);
                [~,ins_size] = size(inserts);
                new_obj.waypoints(:,1:1:ins_size) = inserts;
                new_obj.waypoints(:,new_n) = obj.waypoints(:,old_n);
                new_obj.length = new_n;
                new_obj.distance = 0;
                for i = 2:new_obj.length
                    new_obj.distance = new_obj.distance + norm(
                        new_obj.waypoints(:,i-1)-new_obj.waypoints(:,i),2);
                end
            else
                new_obj = obj;
            end
        end

        % Append a waypoint to the path
        function append(obj, x, y)
            [~, old_n] = size(obj.waypoints);
            obj.waypoints(:,old_n+1) = [x; y];
            obj.length = obj.length + 1;
            if(obj.length == 1)
                obj.distance = 0;
            else
                obj.distance = obj.distance + norm(obj.waypoints(:,obj.length
                    -1)-obj.waypoints(:,obj.length),2);
            end
        end


    end

end
```

A.8 Report.m

```
classdef Report < handle
    % Collects data

    properties
        data
        prop
    end

    methods

        % Constructor
        function obj = Report(prop)
            obj.prop = prop;
        end

        % Save to file
        function save(obj)
```

```
            save(strcat(pwd,'\results\',obj.prop.Sys.identifier,'_report.mat'),
                 'obj','-mat');
        end

    end

end
```

A.9 generate_maps.m

```
function [type_map,quality_map,maxquality_map] = generate_maps(Prop)
%   Generates a map from a png file.
%   creates a 1000x1000 map with 0:4 values, 4:1 for rbgk and 0 for
%   anything else.

% When drawing in GIMP, choose for H value:
% 330-360 and 0-30: Nothing     MEAN: 0
% 30-90: Flower 1               MEAN: 60
% 90-150: Flower 2              MEAN: 120
% 150-210: Flower 3             MEAN: 180
% 210-270: Flower 4             MEAN: 240
% 270-330: Smog                 MEAN: 300
% for S, choose 100, for V choose quality between 0 and 100%


    image = imread(Prop.Sim.world_file);
    image = rgb2hsv(image);
    size_image = size(image);

    %resize if necessary
    if  size_image(1:2) == Prop.Sim.world_size/10
    else
        image = imresize(image, [Prop.Sim.world_size/10,Prop.Sim.world_size
            /10]);
    end

    %initalize empty maps
    type_map = Map(0,Prop.Sim.world_size/10,Prop.Sim.world_size/10);
    quality_map = Map(0,Prop.Sim.world_size/10,Prop.Sim.world_size/10);
    maxquality_map = Map(0,Prop.Sim.world_size/10,Prop.Sim.world_size/10);

    %fill arrays
    type_map.array = image(:,:,1);
    maxquality_map.array = image(:,:,3);
    %scale the values of h to 0:5
    type_map.array = round(mod((type_map.array)*6,6));
    % Use this in debug mode to test if the map has been read correctly:
    %figure
    %imagesc(type_map.array);
    %colormap(hsv);
end
```

A.10 interpolate_function.m

```
function f = interpolate_function(xin, yin, degree, ymin, ymax)
    % Interpolate values into function handle
    % xin       1xN vector with x values
    % yin       1xN vector with y values
    % degree    polynom degree (must be smaller than N)
    % ymin      low cap for y values
    % ymax      high cap for y values
    [a,~,mu] = polyfit(xin,yin,degree);
    f = @(x) min(max(polyval(a, (x-mu(1))/mu(2)),ymin),ymax);
end
```

A.11 interpolate_values.m

```
function y = interpolate_values(xin, yin, xmin, xstep, xmax, ymin, ymax)
    % Interpolate values into vector
    % xin       1xN vector with x values
    % yin       1xN vector with y values
    % xmin      min. value for interpolation range
    % xstep     step between two interpolation points
    % xmax      max. value for interpolation range
    % ymin      low cap for y values
    % ymax      high cap for y values
    y = min(max(interp1(xin, yin, (xmin:xstep:xmax),'spline'),ymin),ymax);
end
```

A.12 bresenham.m

(From MVTB [2])

```
%BRESENHAM Generate a line
%
% P = BRESENHAM(X1, Y1, X2, Y2) is a list of integer coordinates for
% points lying on the line segement (X1,Y1) to (X2,Y2).  Endpoints
% must be integer.
%
% P = BRESENHAM(P1, P2) as above but P1=[X1,Y1] and P2=[X2,Y2].
%
% See also ICANVAS.
% Copyright (C) 1993-2011, by Peter I. Corke
%
% This file is part of The Machine Vision Toolbox for Matlab (MVTB).
%
% MVTB is free software: you can redistribute it and/or modify
% it under the terms of the GNU Lesser General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% MVTB is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU Lesser General Public License for more details.
%
% You should have received a copy of the GNU Leser General Public License
% along with MVTB.  If not, see <http://www.gnu.org/licenses/>.

 function [x,y] = bresenham(x1, y1, x2, y2)
    if nargin == 2
        p1 = x1; p2 = y1;
        x1 = p1(1); y1 = p1(2);
        x2 = p2(1); y2 = p2(2);
    elseif nargin ~= 4
        error('expecting 2 or 4 arguments');
    end
    x = x1;
    if x2 > x1
        xd = x2-x1;
        dx = 1;
    else
        xd = x1-x2;
        dx = -1;
    end
    y = y1;
    if y2 > y1
        yd = y2-y1;
        dy = 1;
    else
        yd = y1-y2;
        dy = -1;
    end
    p = [];
    if xd > yd
      a = 2*yd;
      b = a - xd;
      c = b - xd;
      while (1)
        p = [p; x y];
        if all([x-x2 y-y2] == 0)
            break
        end
        if  b < 0
            b = b+a;
            x = x+dx;
        else
            b = b+c;
            x = x+dx; y = y+dy;
        end
      end
    else
      a = 2*xd;
      b = a - yd;
      c = b - yd;
      while 1
        p = [p; x y];
        if all([x-x2 y-y2] == 0)
            break;
        end
        if  b < 0
            b = b+a;
            y = y+dy;
        else
            b = b+c;
            x = x+dx; y = y+dy;
        end
      end
    end

    x = p(:,1);
    y = p(:,2);
end
```

B Additional Graphics

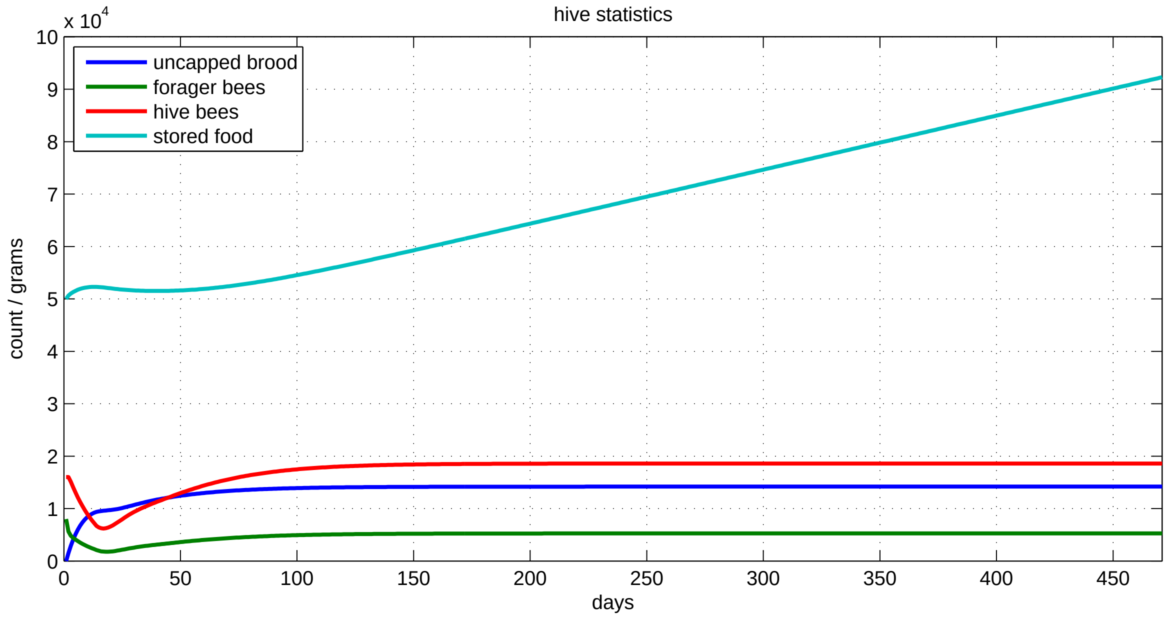
B.1 Simulation run without environment simulation

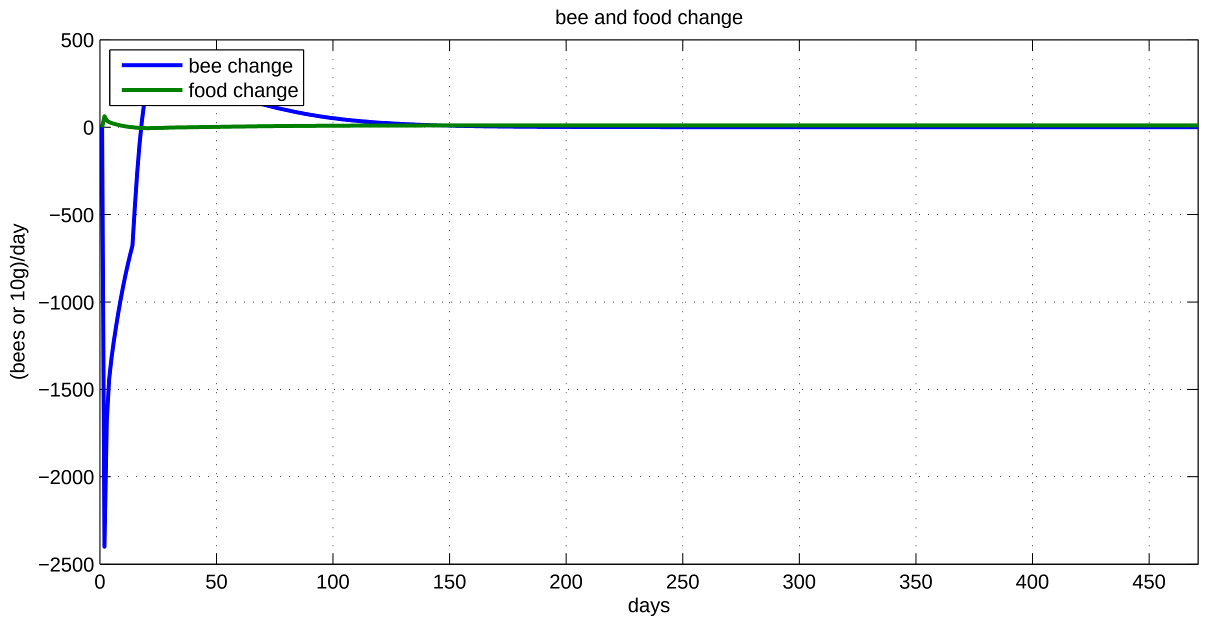
B.1.1 Runscript with changes relative to the base properties

```
% This is a runscript for automatized simulation on any system

% Simple test cases based on Properties_Base
```

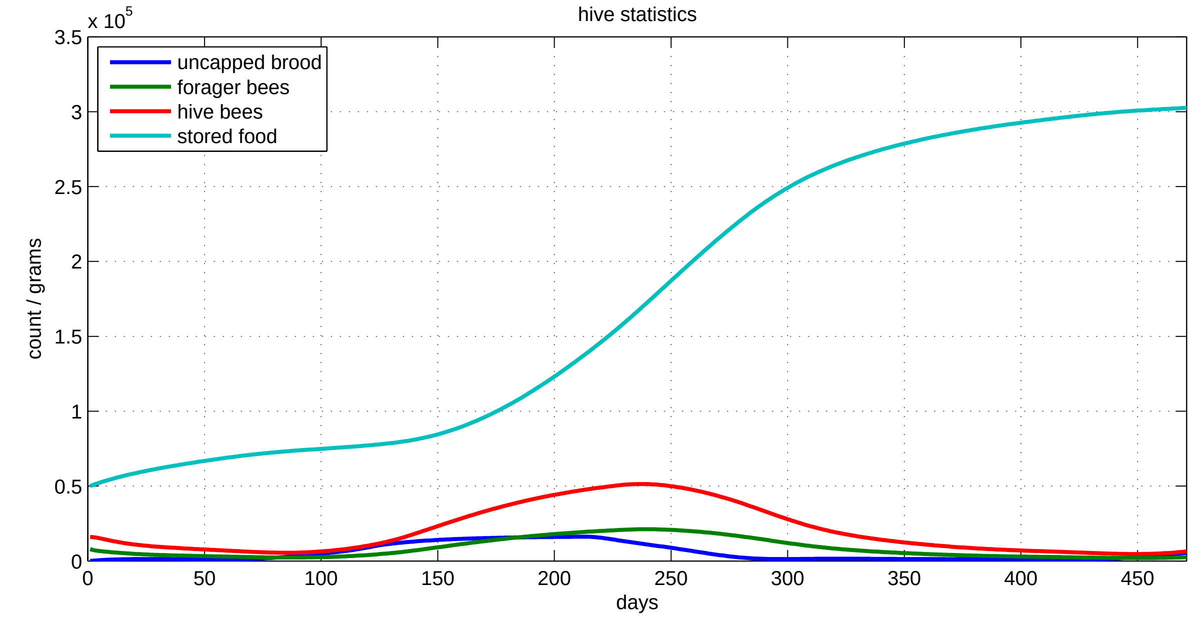
```
run('data\Properties_Base.m');

% Load the properties into memory, make copies and modify them on the go

% The testcases:
% 1. Original model (constant food, constant laying rate, original mortality)
% 2. Intermediate model (constant food, varying laying rate, adapted mortality)

Prop1 = Prop;
Prop1.Sim.Hive(1).fixed_food_rate = 1;
Prop1.Sim.Hive(1).mortality = 0.3;
Prop1.Sim.Hive(1).laying_function = [0,1,2;1,1,1];
Prop1.Sys.identifier = 'Properties_Base_R0_1';

Prop2 = Prop;
Prop2.Sim.Hive(1).fixed_food_rate = 1;
Prop2.Sys.identifier = 'Properties_Base_R0_2';


proparray=[Prop1,Prop2];

% Start simulation
hive_simulation(proparray);
```

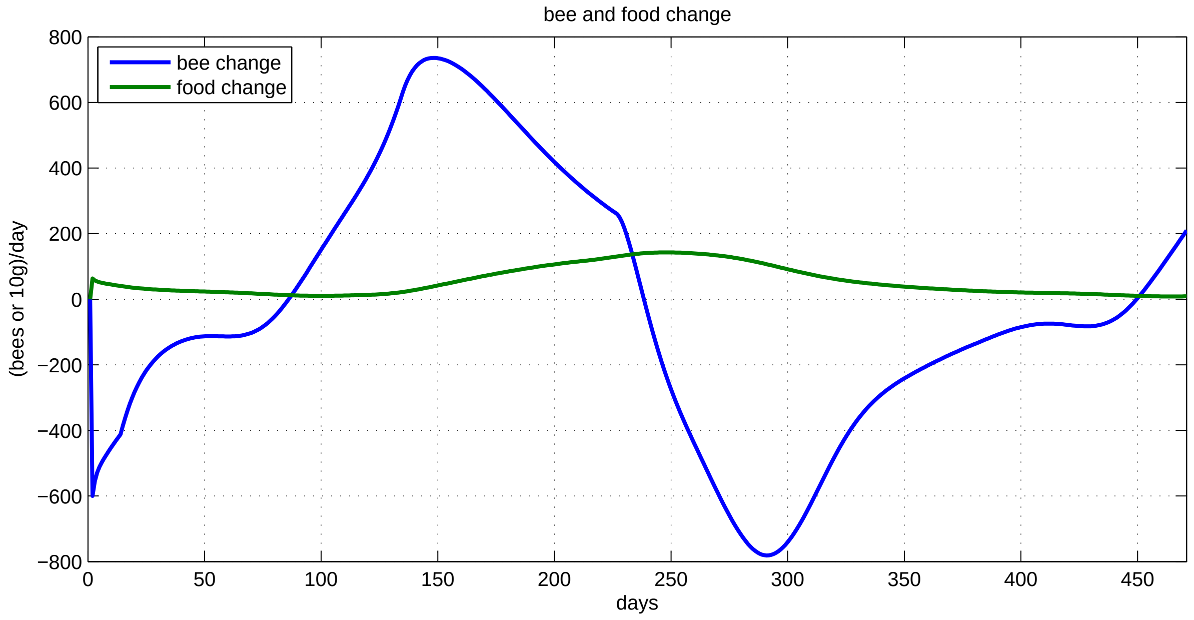
B.1.2 Constant laying rate, constant food income





B.1.3 Varying laying rate, constant food income



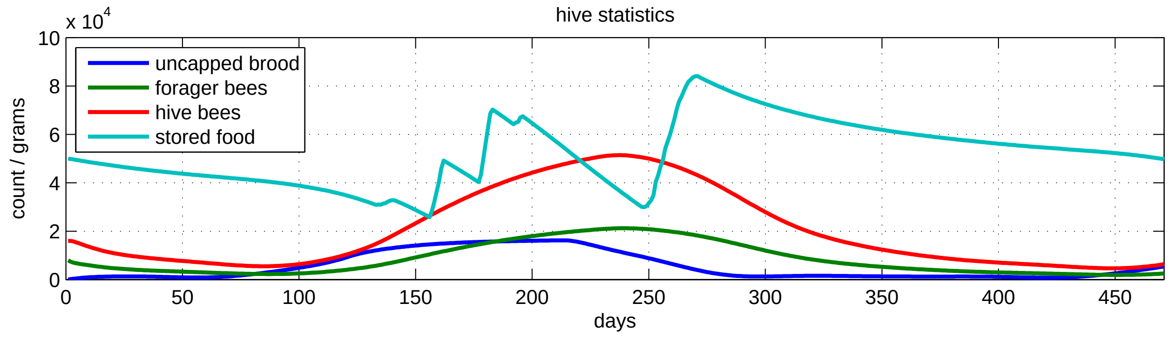


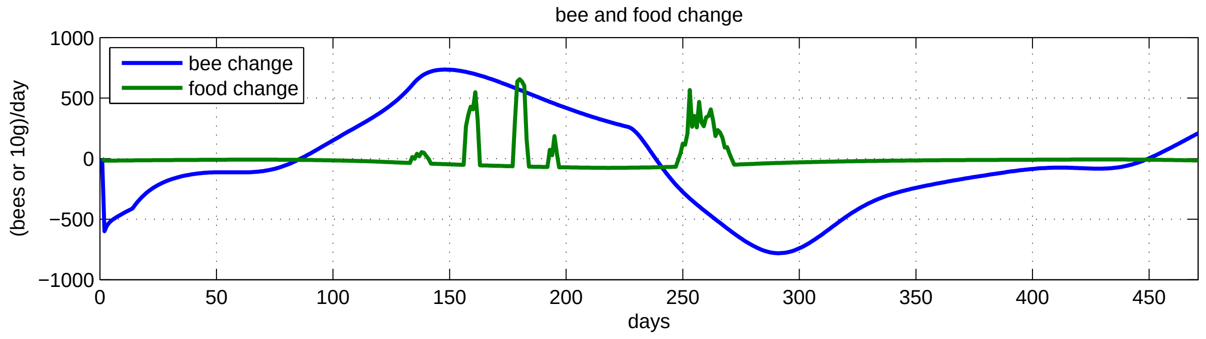
B.2 Simulation run with missing flower patches

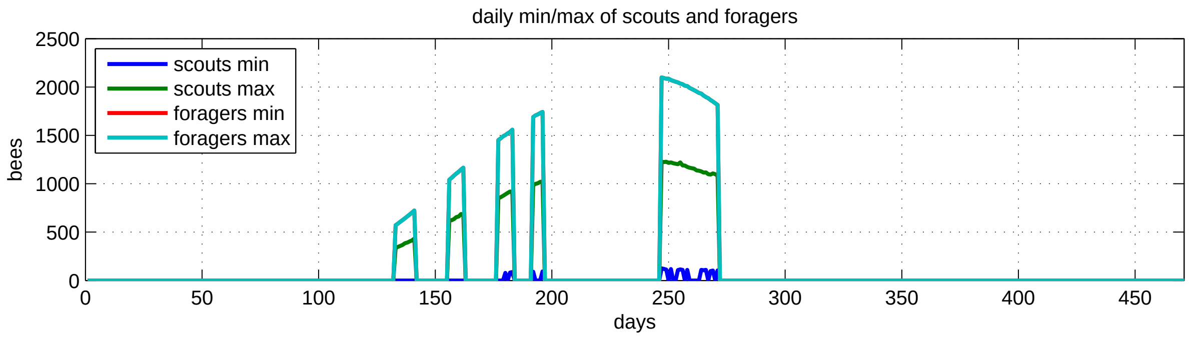
B.2.1 Runscript with changes relative to the base properties

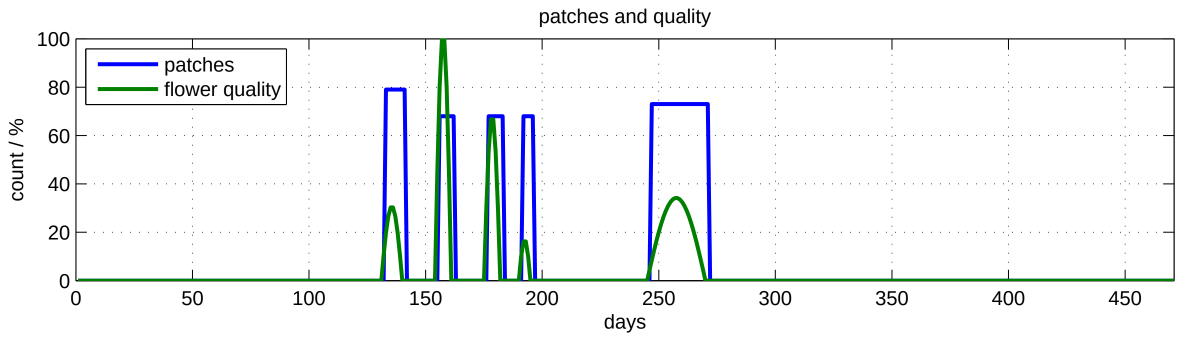
```
% This is a runscript for automatized simulation on any system

% Simple test cases based on Properties_Base
run('data\Properties_Base.m');

% Load the properties into memory, make copies and modify them on the go

% The testcases:
% 1. all flowers
% 2. no spring flowers
% 3. no summer flowers
% 4. no autumn flowers

Prop1 = Prop;
Prop1.Sys.identifier = 'Properties_Base_R1_1';

Prop2 = Prop;
Prop2.Sim.Flower(1).year_activity = [1:2;0,0];
Prop2.Sys.identifier = 'Properties_Base_R1_2';

Prop3 = Prop;
Prop3.Sim.Flower(2).year_activity = [1:2;0,0];
Prop3.Sys.identifier = 'Properties_Base_R1_3';

Prop4 = Prop;
Prop4.Sim.Flower(3).year_activity = [1:2;0,0];
Prop4.Sys.identifier = 'Properties_Base_R1_4';


proparray=[Prop1,Prop2,Prop3,Prop4];

% Start simulation
hive_simulation(proparray);
```

B.2.2 All flower patches

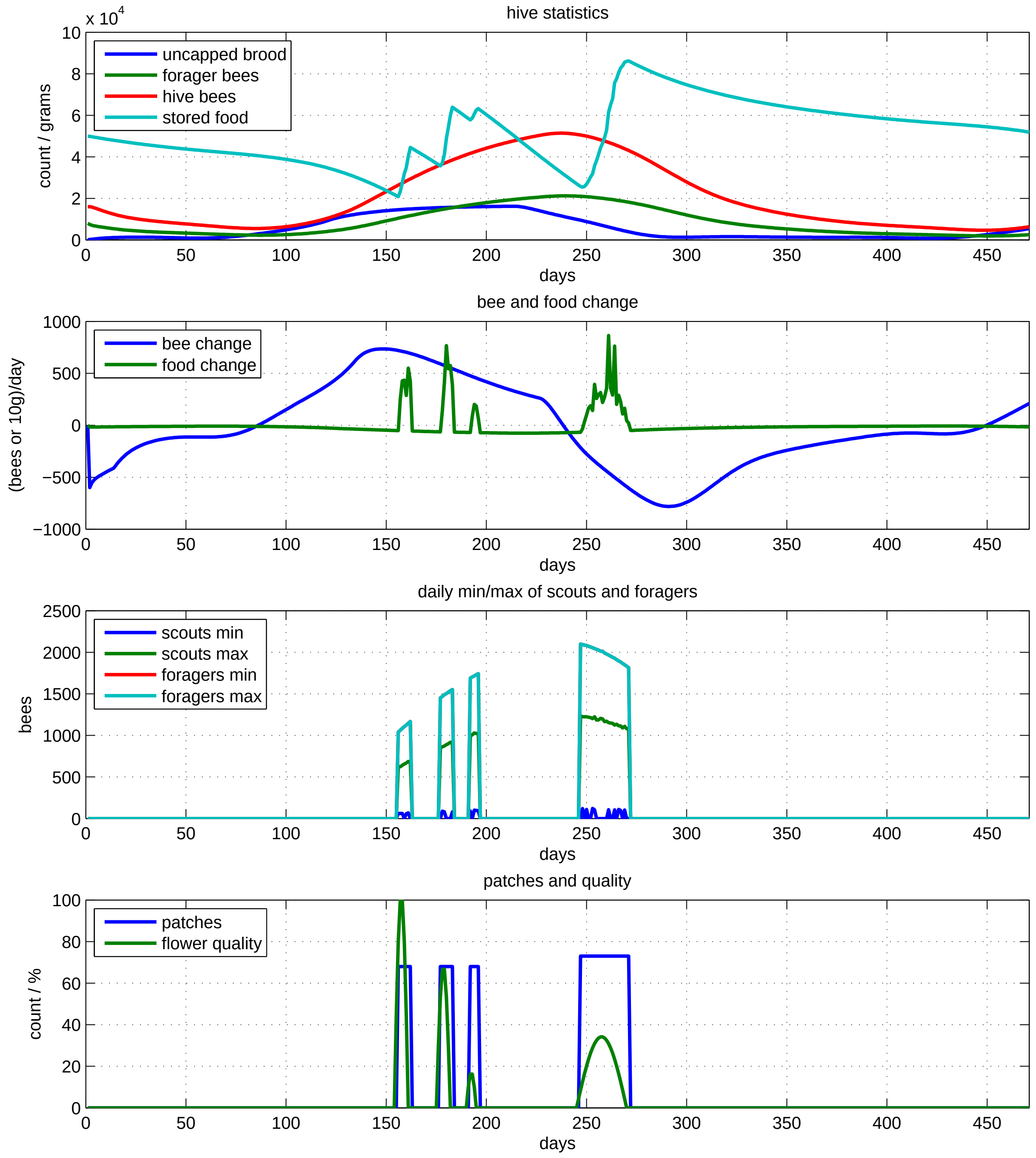




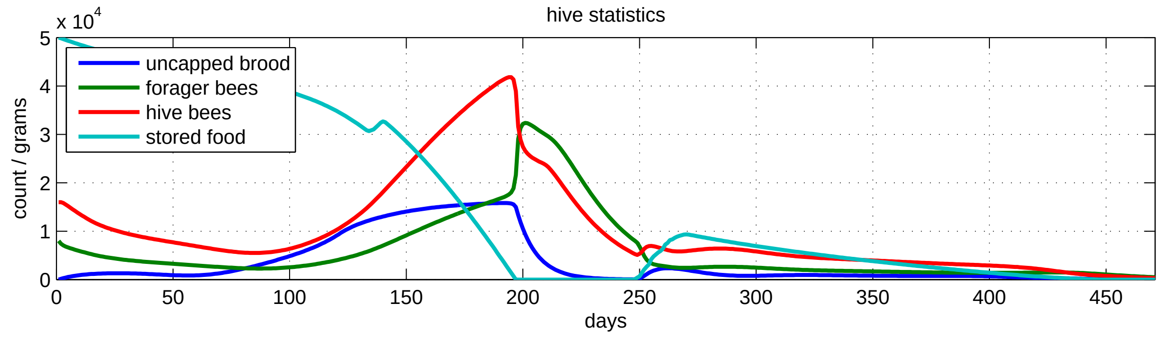


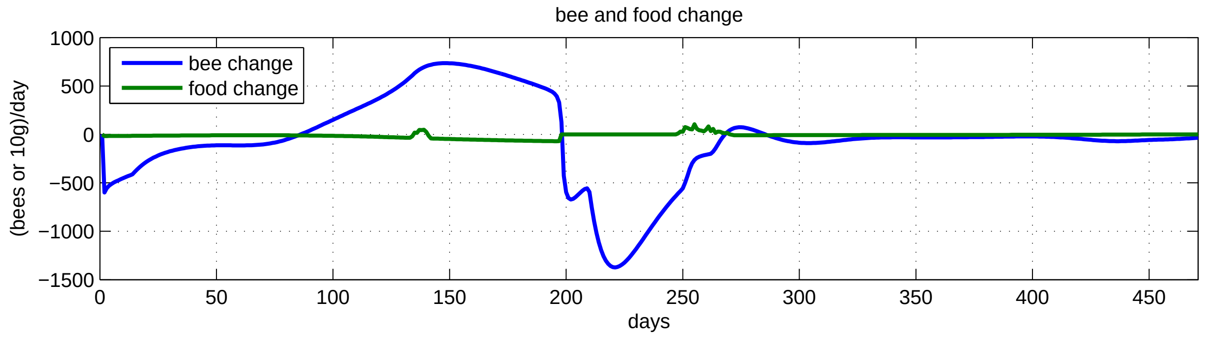


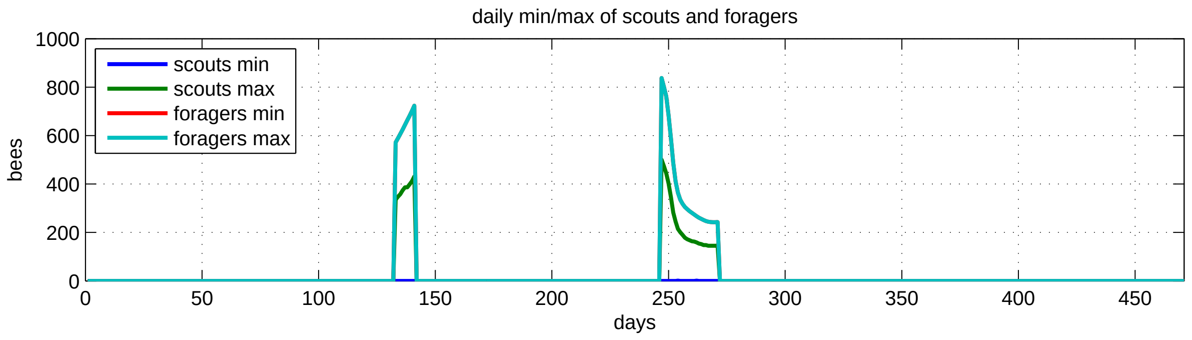
B.2.3 Missing spring flowers

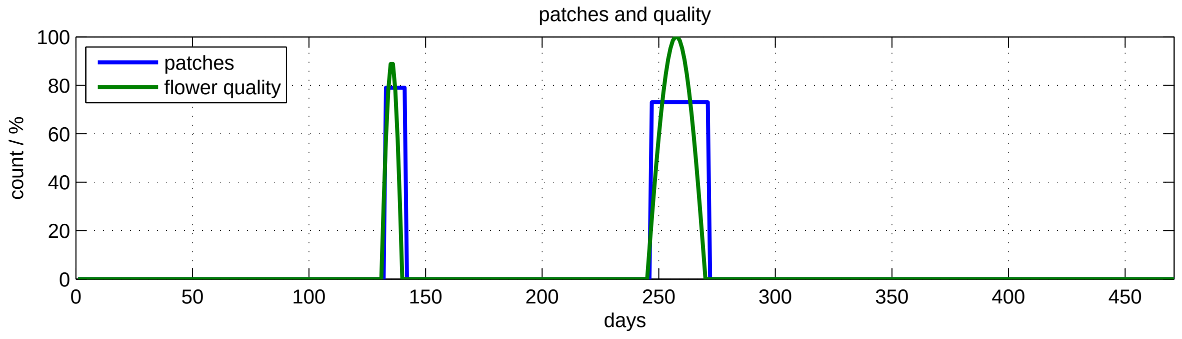


B.2.4 Missing summer flowers

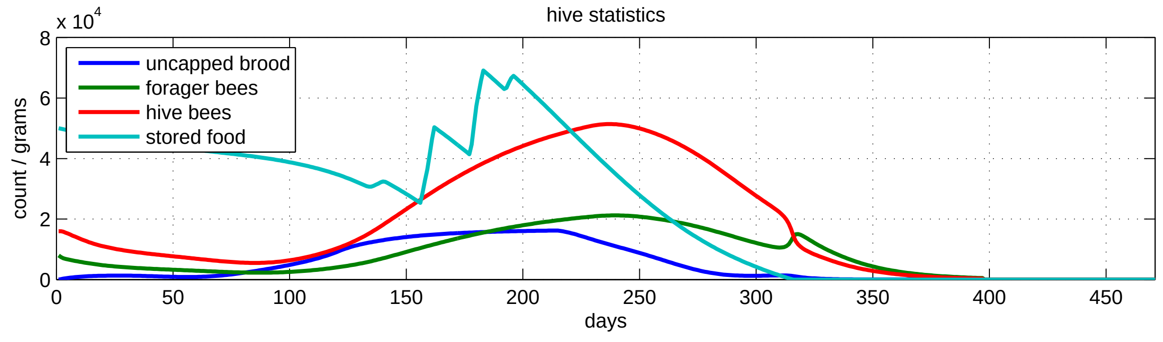


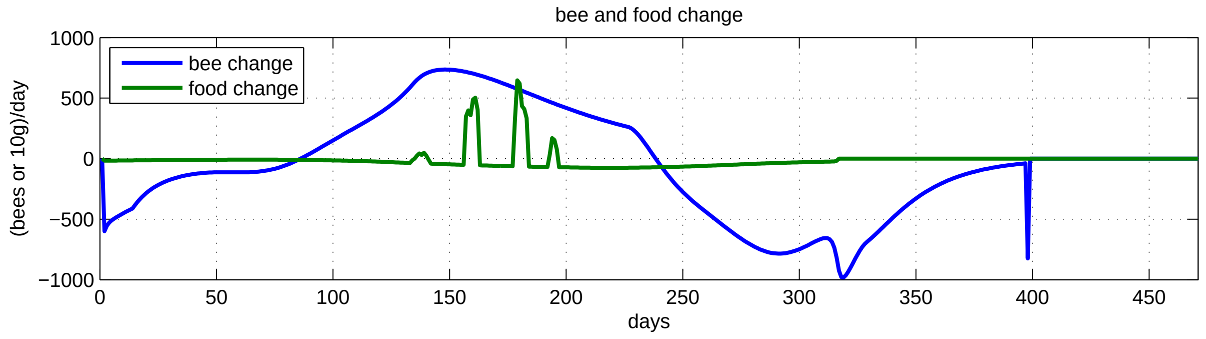


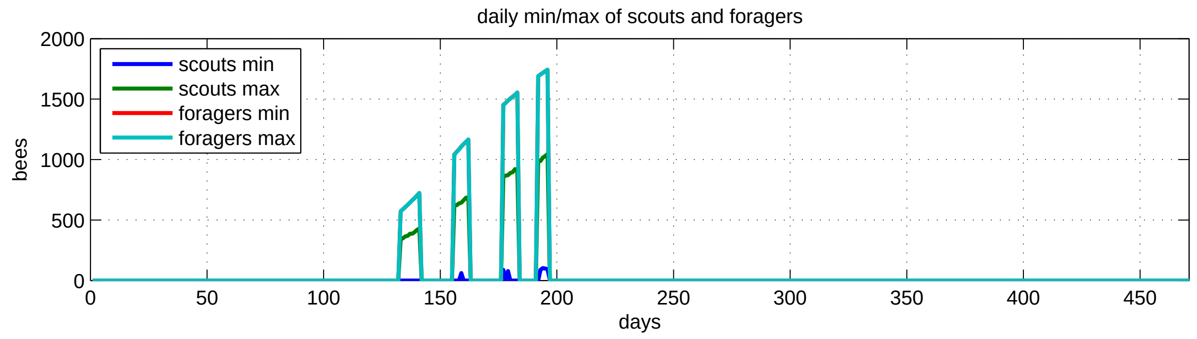


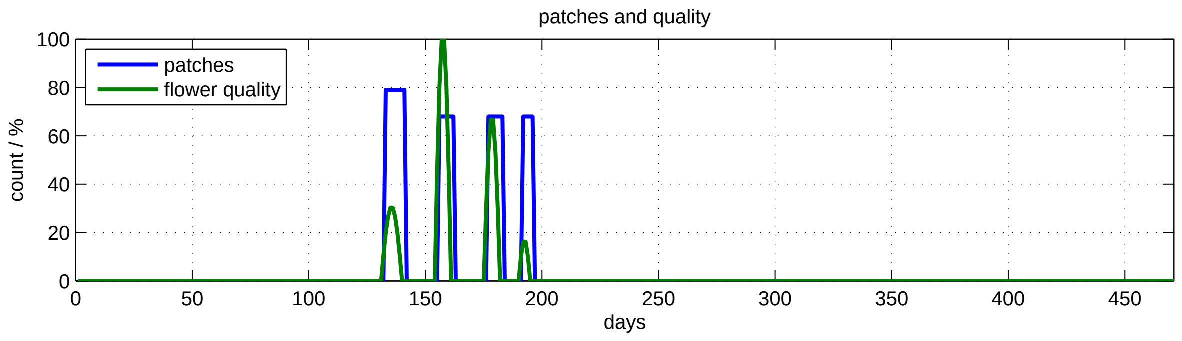


B.2.5 Missing autumn flowers







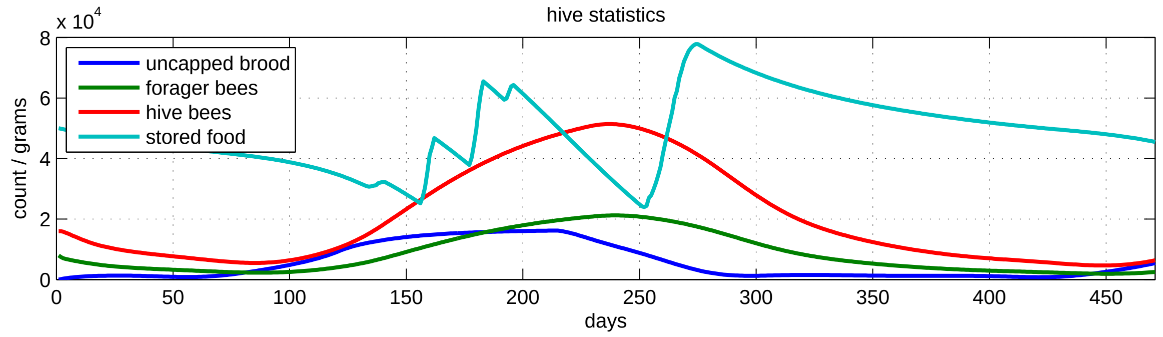


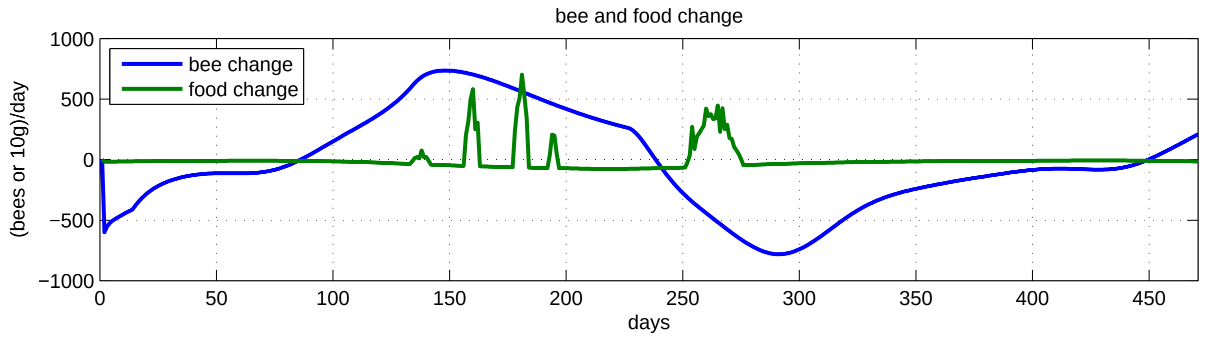
B.3 Simulation run with varying autumn flowers

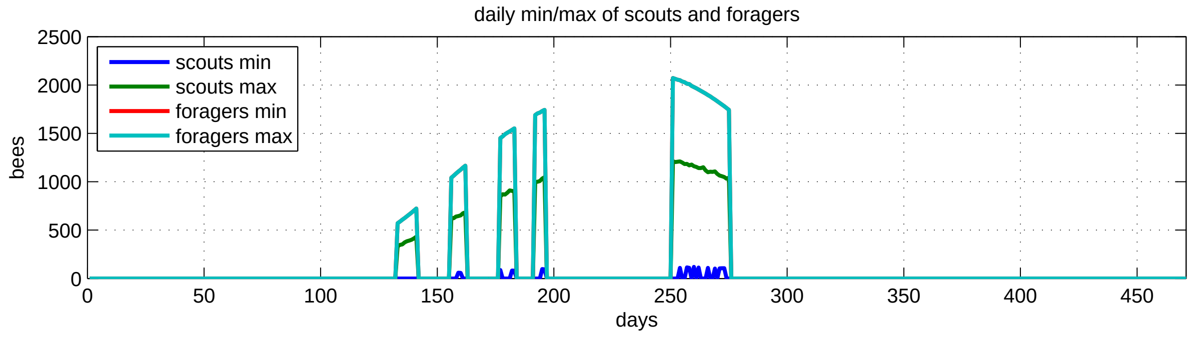
B.3.1 Runscript with changes relative to the base properties

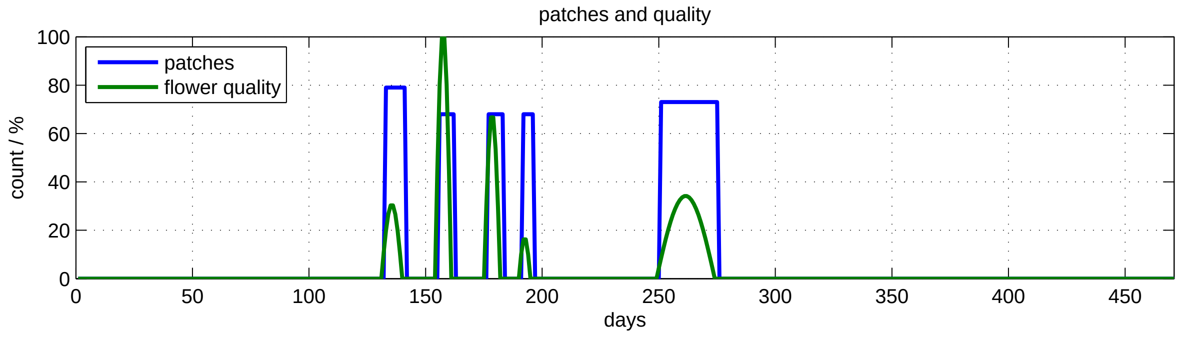
```
% This is a runscript for automatized simulation on any system

% Simple test cases based on Properties_Base
run('data\Properties_Base.m');

% Load the properties into memory, make copies and modify them on the go

% The testcases, variable change 1:
% 1. Autumn flower peak = 2
% 2. Autumn flower peak = 1.5
% 3. Autumn flower peak = 1
% 4. Autumn flower peak = 0.5

% The testcases, variable change 2:
% 1. Autumn flowers shifted by 4 days
% 2. Autumn flowers shifted by 8 days
% 3. Autumn flowers shifted by 12 days
% 4. Autumn flowers shifted by 16 days
% 5. Autumn flowers shifted by 20 days

% Empty proparray at first
proparray = [];

% 4 possible peaks to test
peaks = [2,1.5,1,0.5];

for j=1:5
    for i=1:4
        identifier = strcat('Properties_Base_R2_',num2str(i),'_',num2str(j));
        tprop = Prop;
        tprop.Sys.identifier = identifier;
        tprop.Sim.Flower(3).year_activity(1,:) = tprop.Sim.Flower(3)
            .year_activity(1,:) + j * 4;
        tprop.Sim.Flower(3).peak = peaks(i);
        proparray = [proparray, tprop];
    end
end

% Start simulation
hive_simulation(proparray);
```

B.3.2 Peak = 2, delayed by 4 days

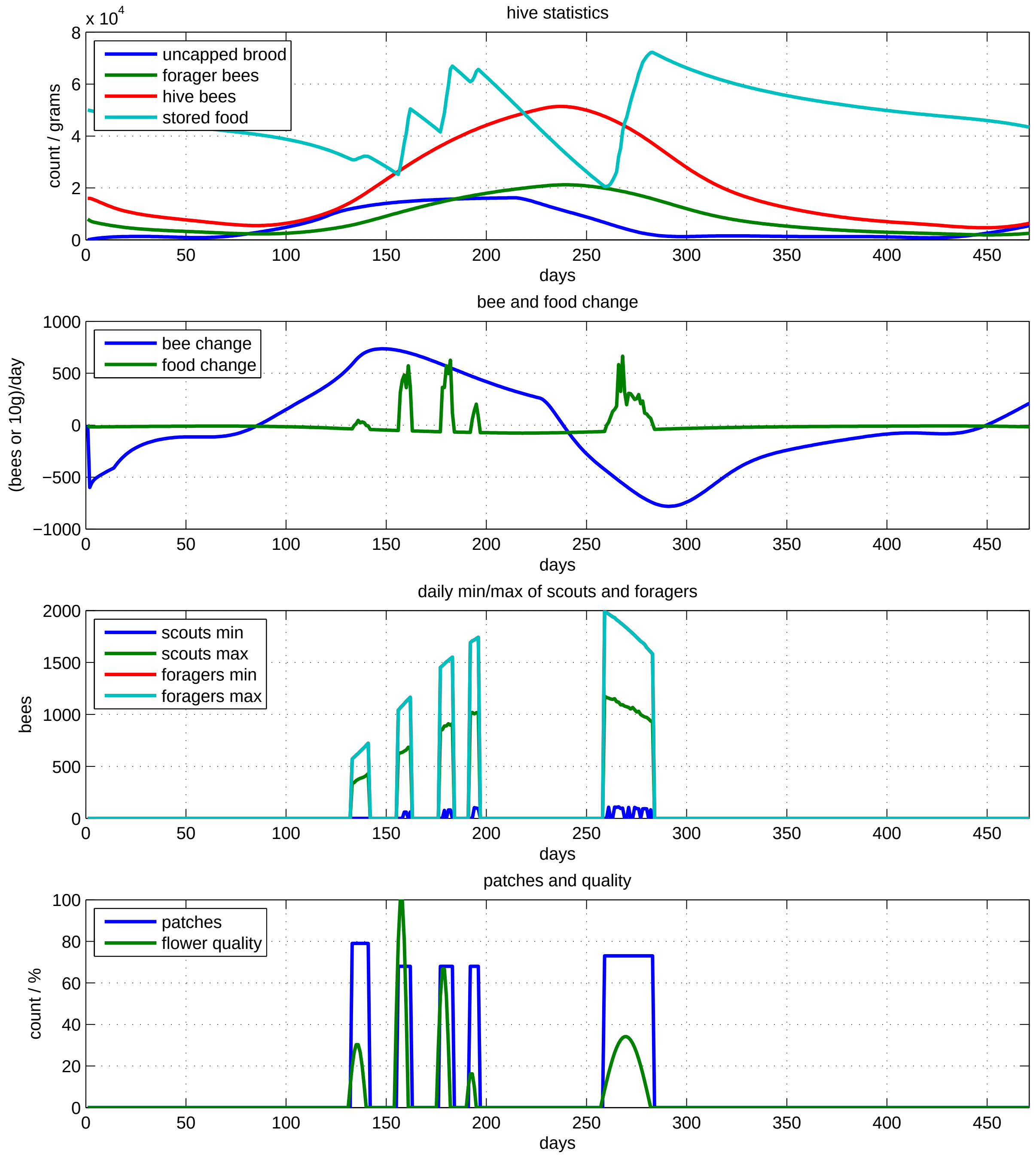




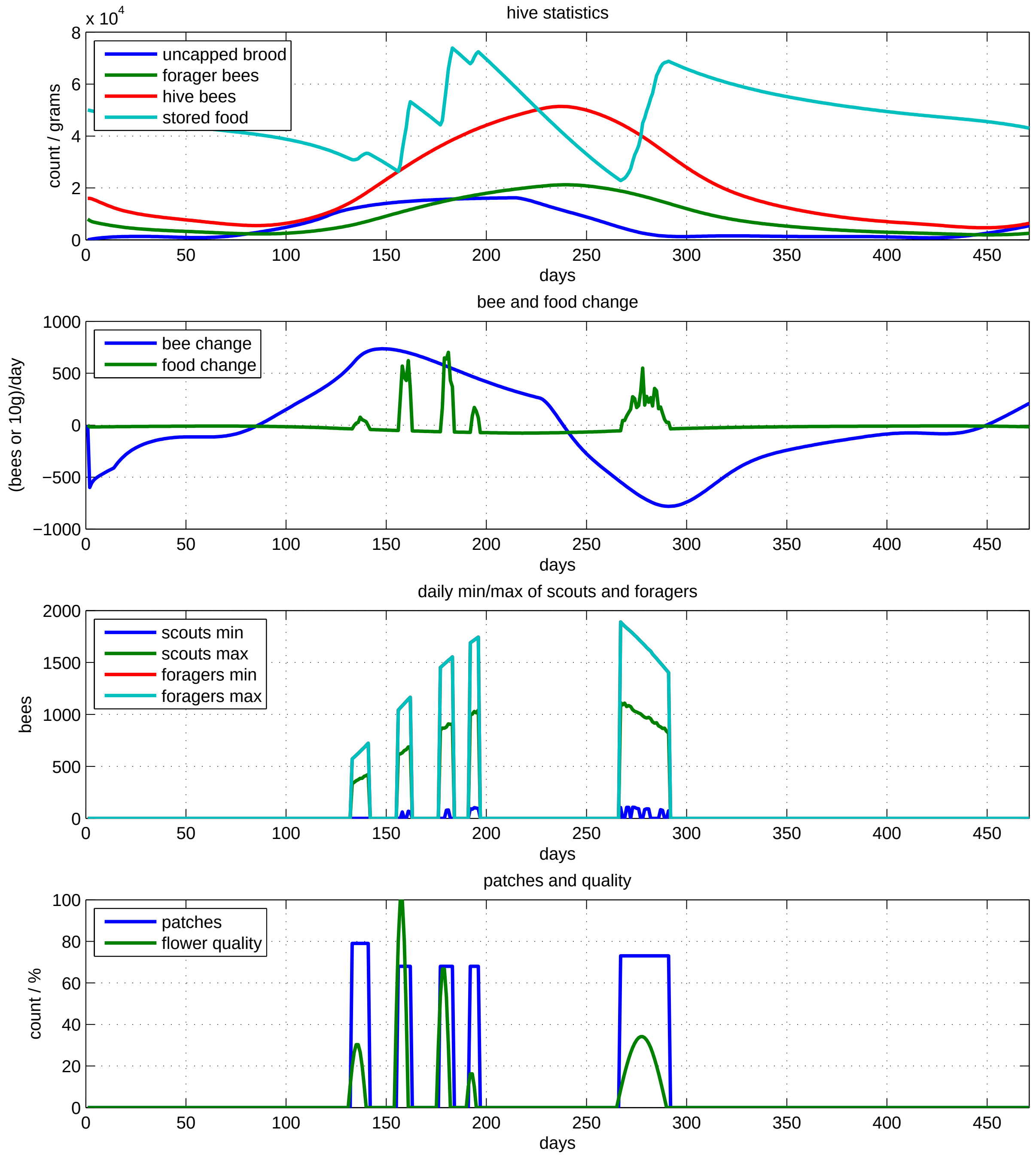




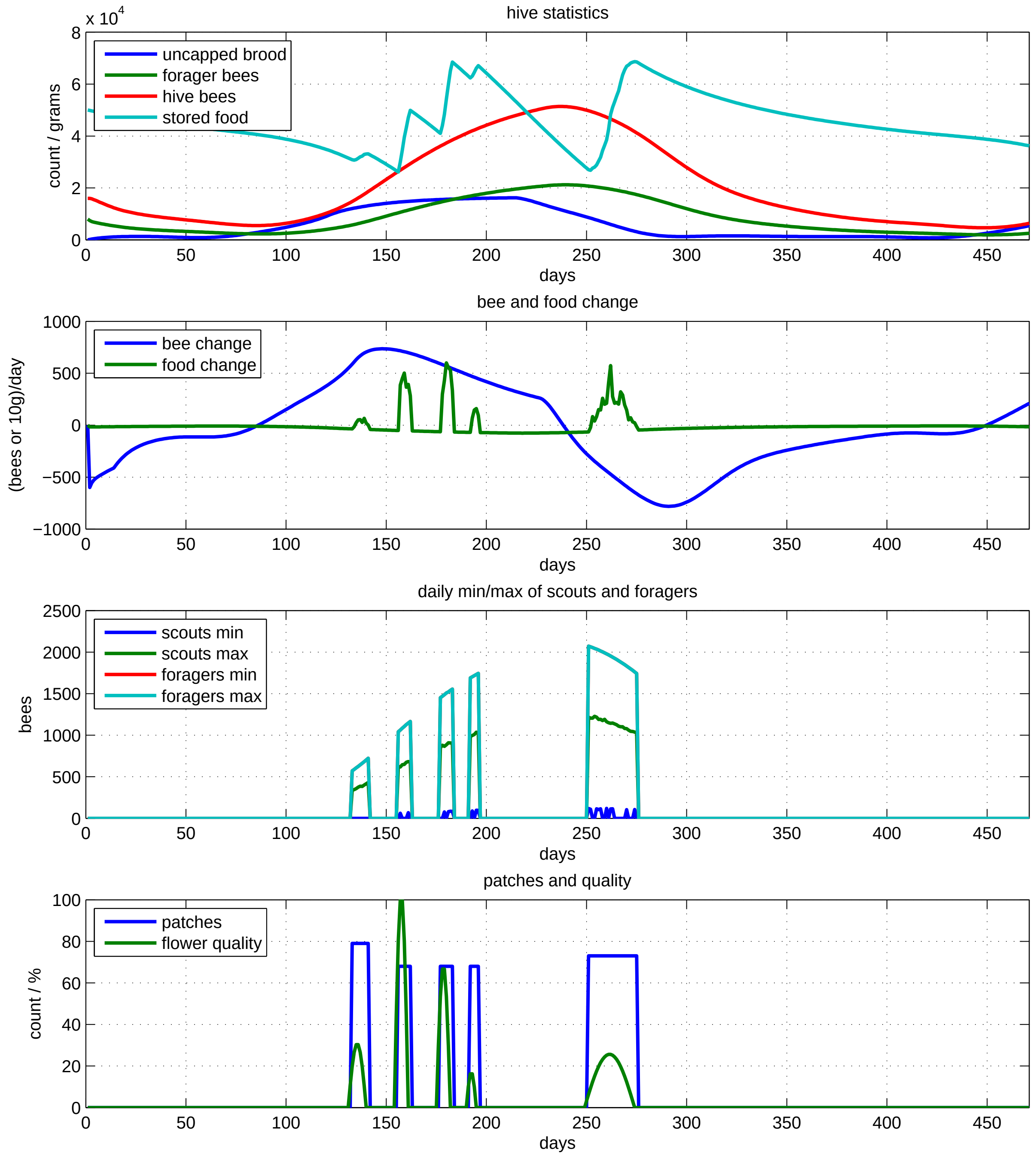
B.3.3 Peak = 2, delayed by 12 days



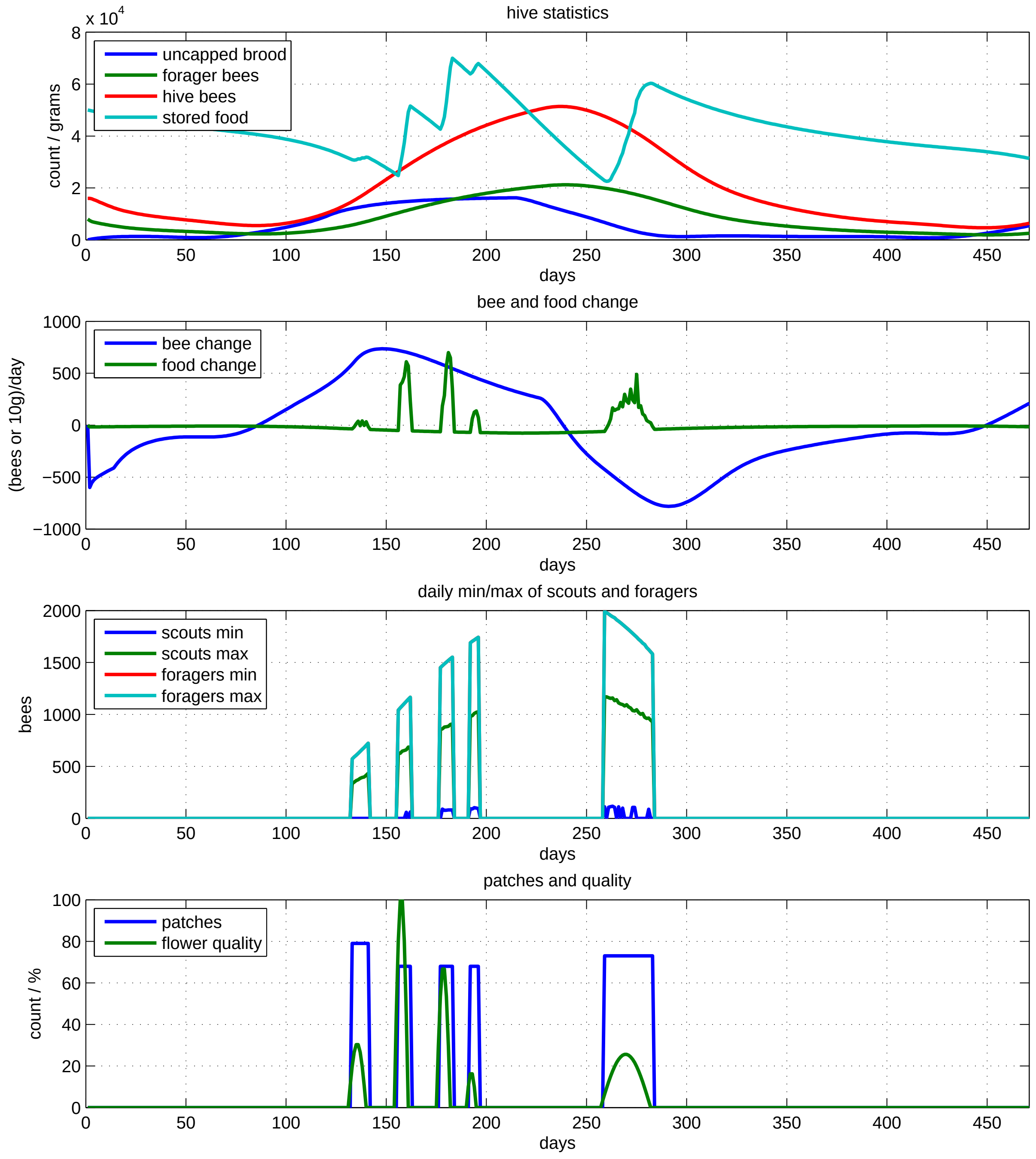
B.3.4 Peak = 2, delayed by 20 days



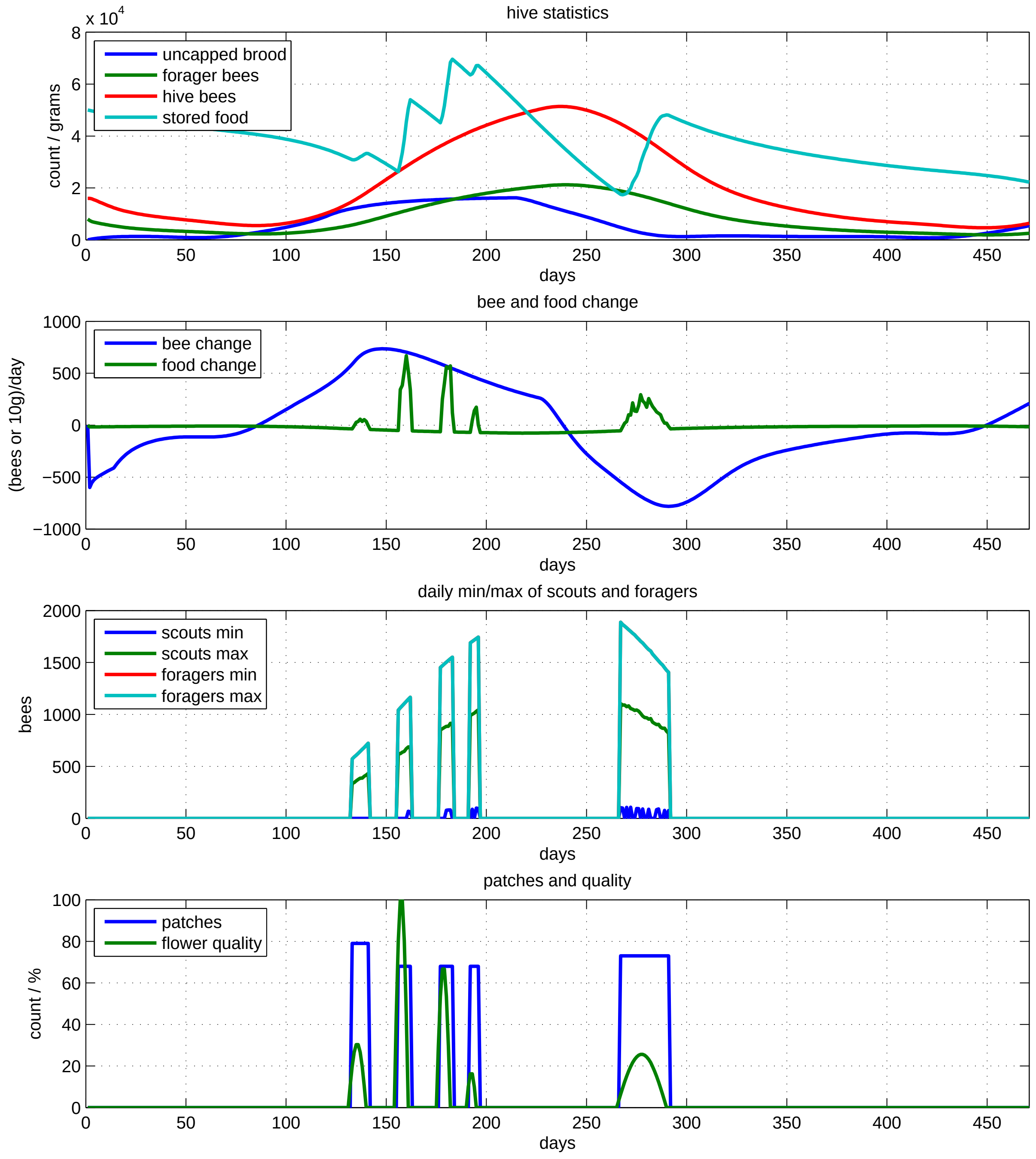
B.3.5 Peak = 1.5, delayed by 4 days



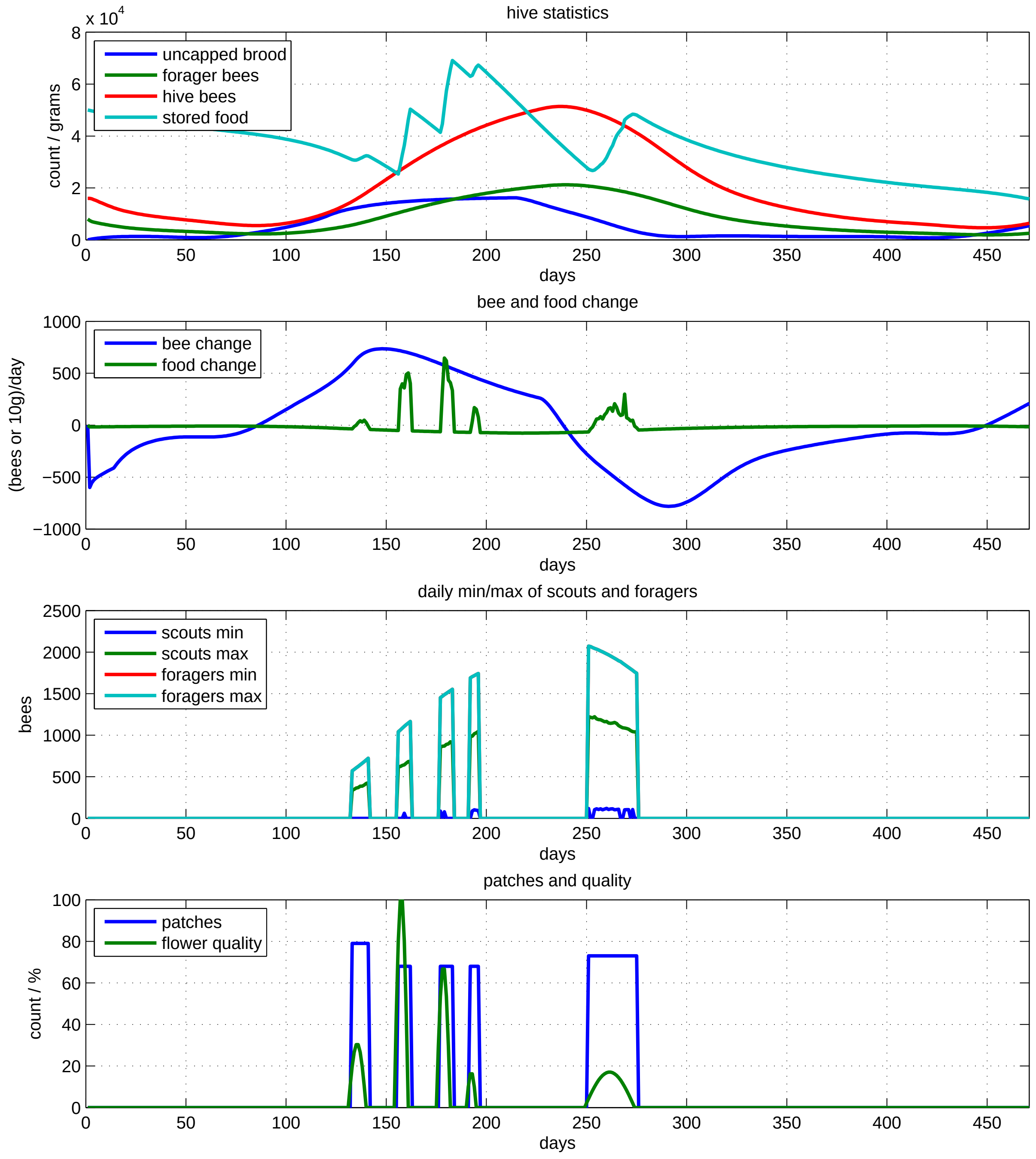
B.3.6 Peak = 1.5, delayed by 12 days



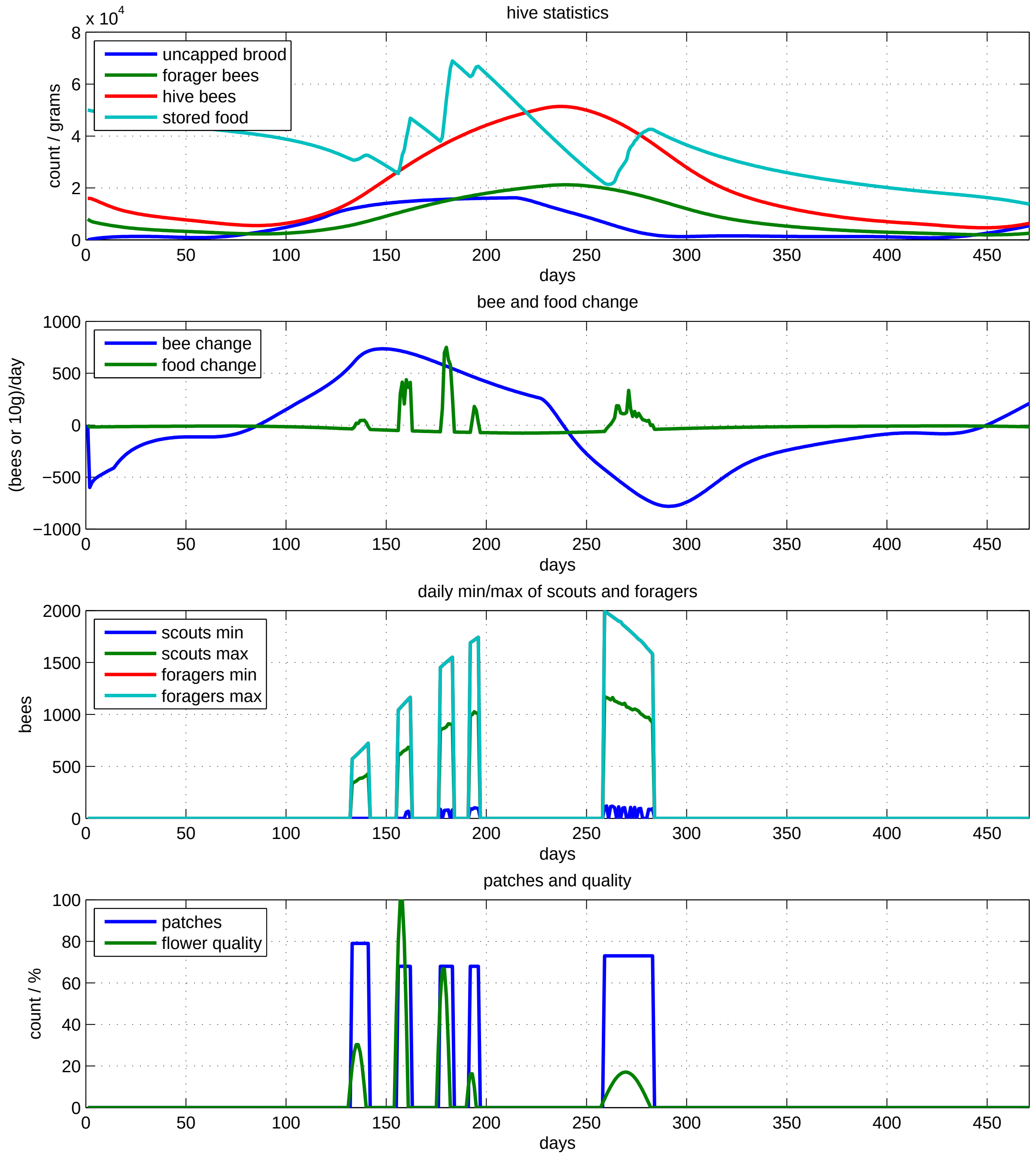
B.3.7 Peak = 1.5, delayed by 20 days



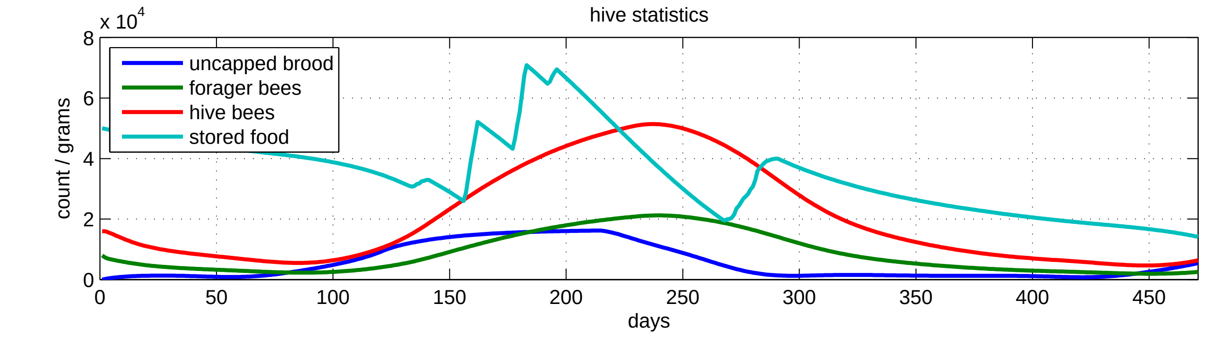
B.3.8 Peak = 1, delayed by 4 days

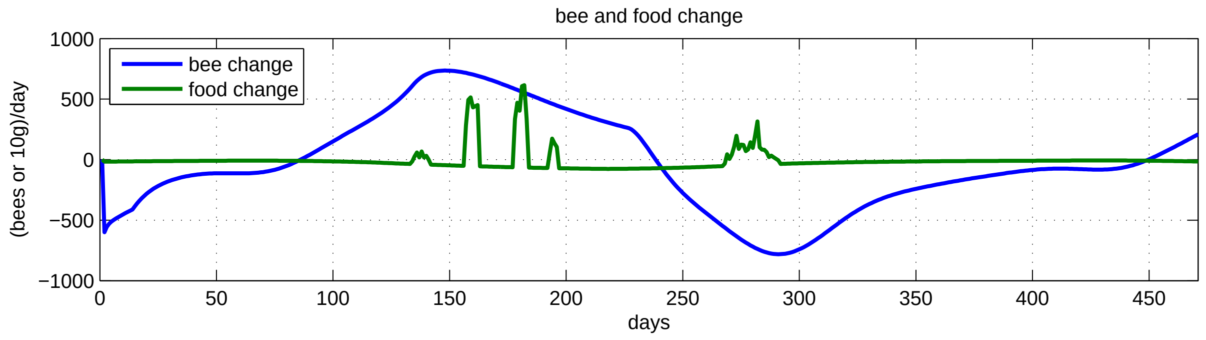


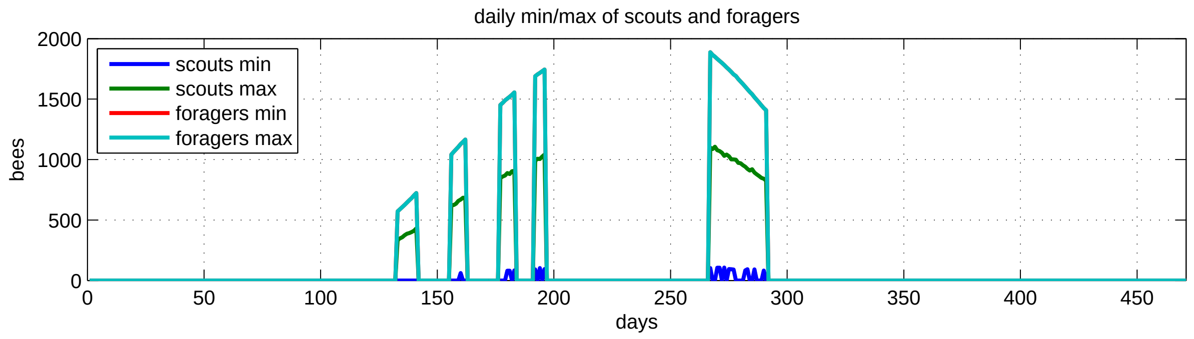
B.3.9 Peak = 1, delayed by 12 days

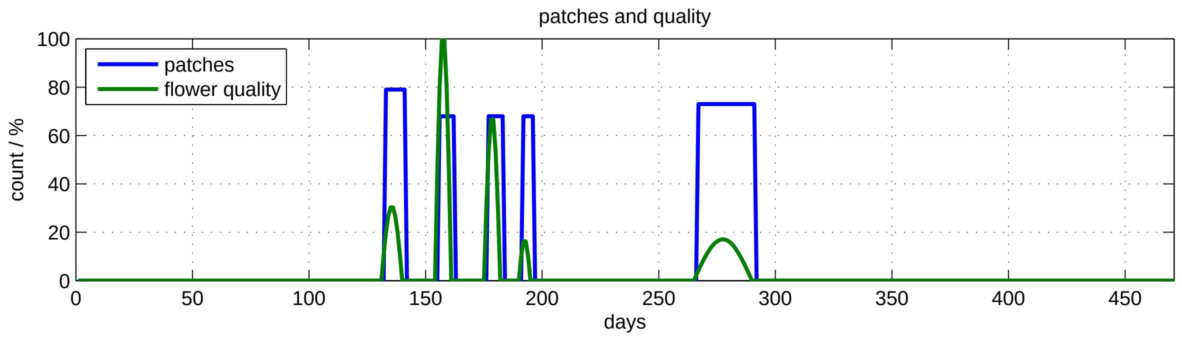


B.3.10 Peak = 1, delayed by 20 days

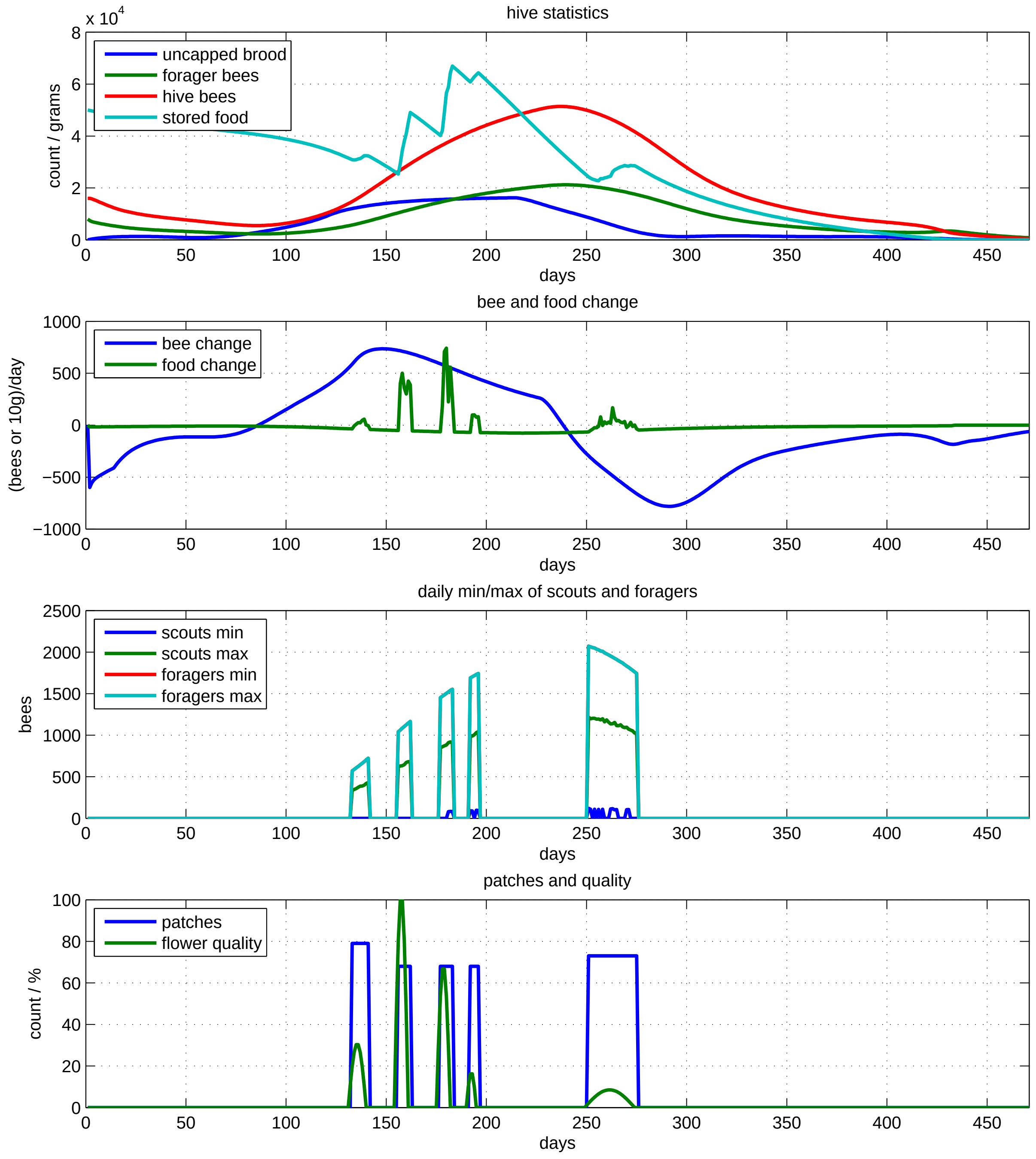




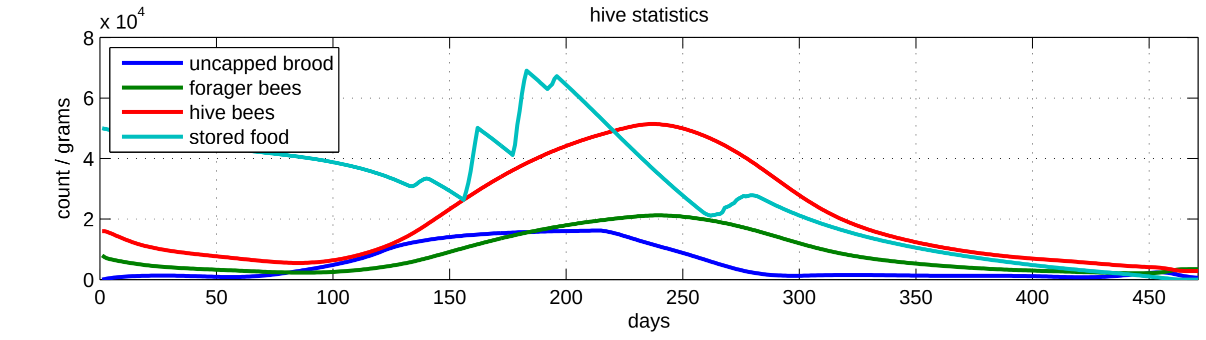


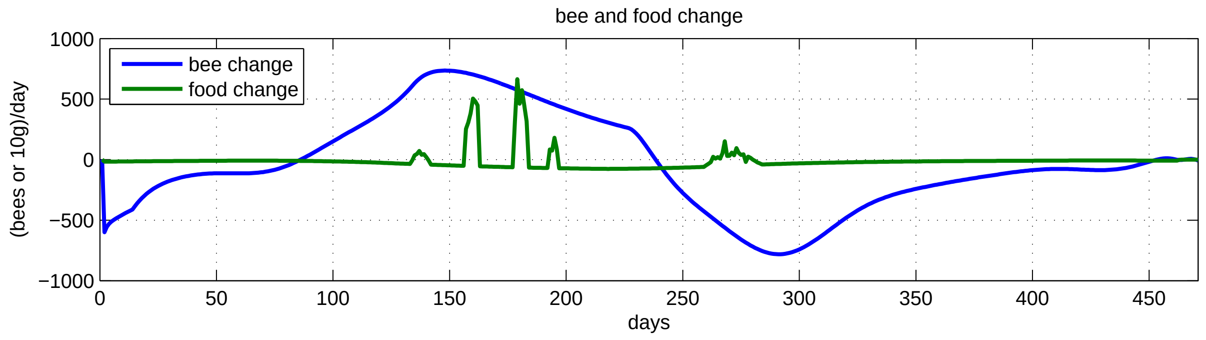


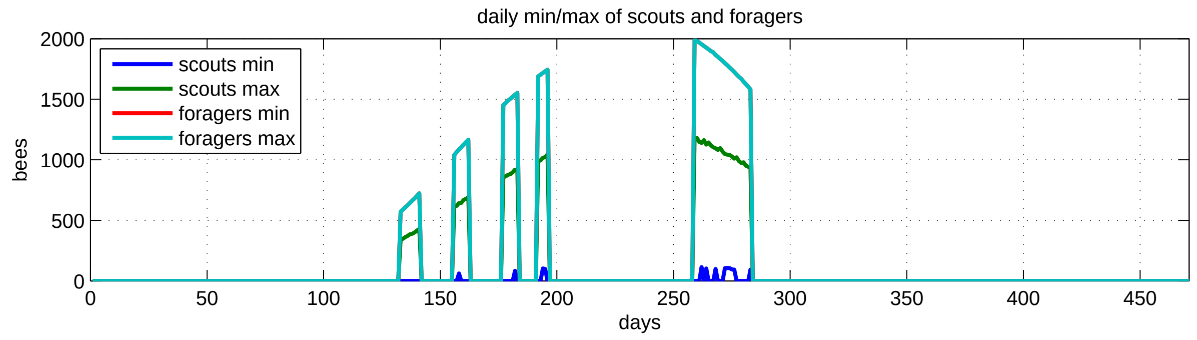
B.3.11 Peak = 0.5, delayed by 4 days

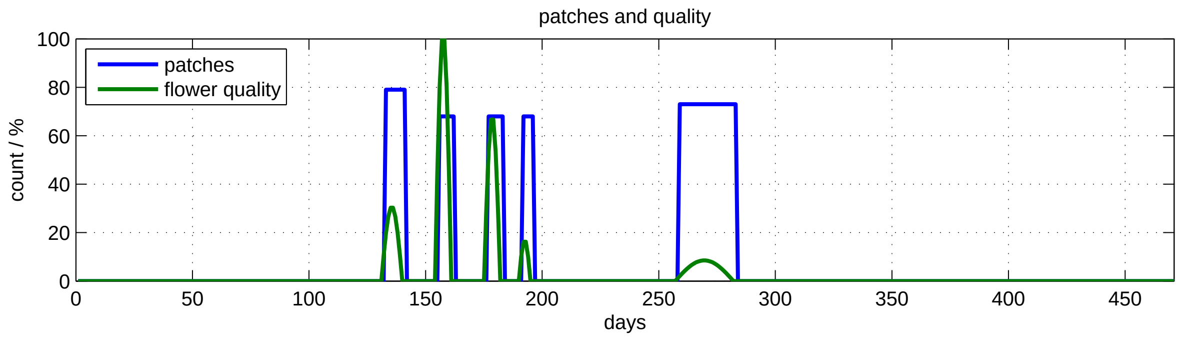


B.3.12 Peak = 0.5, delayed by 12 days

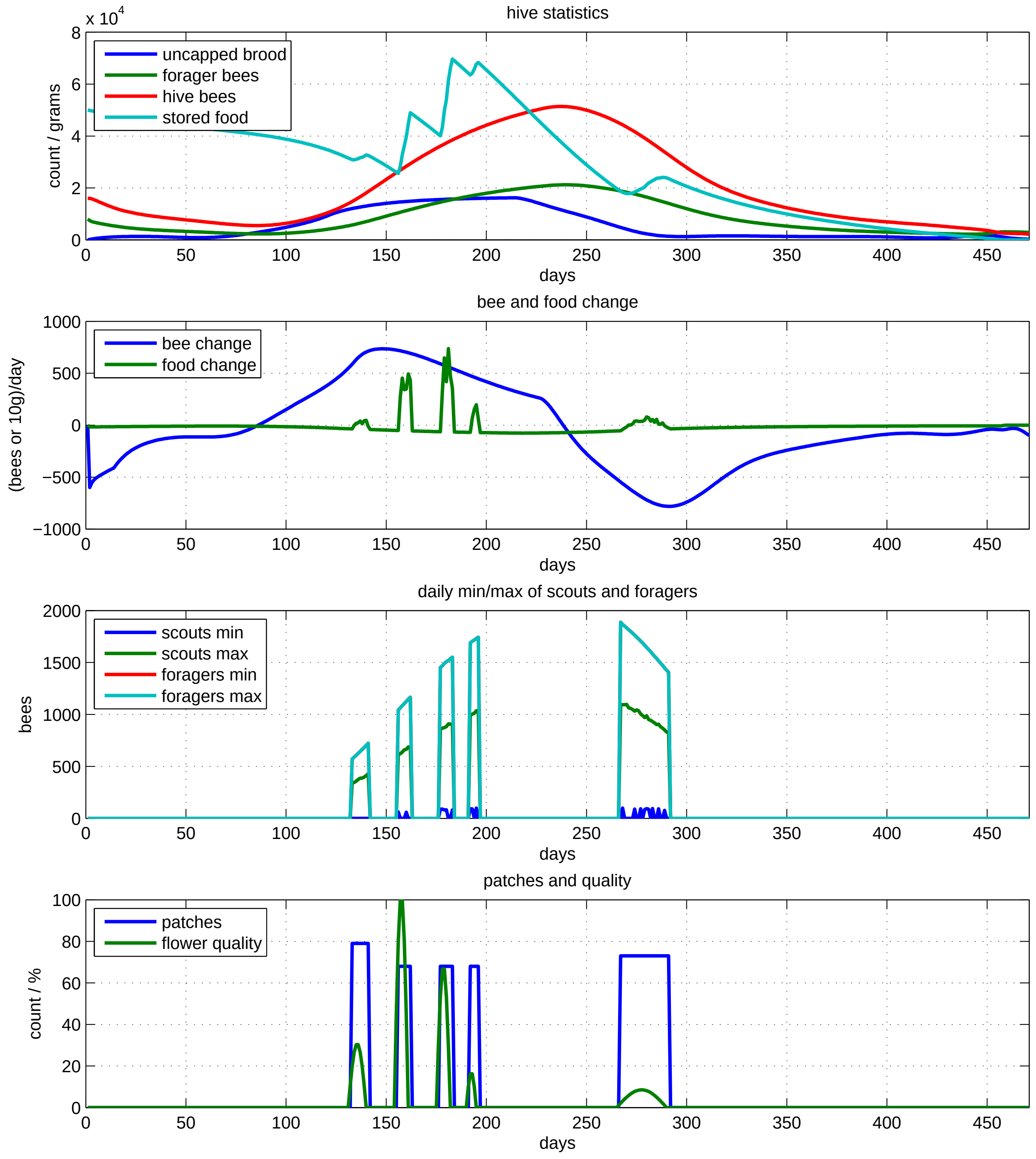








B.3.13 Peak = 0.5, delayed by 20 days


hive statistics
x 10^4
count / grams
uncapped brood
forager bees
hive bees
stored food
days
bee and food change
(bees or 10g)/day
bee change
food change
days
daily min/max of scouts and foragers
bees
scouts min
scouts max
foragers min
foragers max
days
patches and quality
count / %
patches
flower quality
days