POLITECNICO DI TORINO
Repository ISTITUZIONALE

PyPlants: an open-source python package implementing disease and phenological models for plants

Original
PyPlants: an open-source python package implementing disease and phenological models for plants / Colucci, G.P., Salotti, I., Battilani, P., Trinchero, D.. - In: SOFTWAREX. - ISSN 2352-7110. - ELETTRONICO. - 36:(2026), pp. 1-4. [10.1016/j.softx.2026.103062]

Availability:
This version is available at: 11583/3015967 since: 2026-09-27T06:25:40Z

Publisher:
Elsevier

Published
DOI:10.1016/j.softx.2026.103062

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

26 November 2026

SoftwareX 36 (2026) 103062

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

PyPlants: an open-source python package implementing disease and phenological models for plants

Giovanni Paolo Colucci [a,*], Irene Salotti [b], Paola Battilani [b,c], Daniele Trinchero [a,c]

[a] iXem Labs, DET, Politecnico di Torino, Corso Duca degli Abruzzi, 24, Torino, 10129, Italy
[b] Di.Pro.Ve.S., Università Cattolica del Sacro Cuore, Via Emilia Parmense, 84, Piacenza, 29122, Italy
[c] Plant Health Modelling Research Center—PHEM, Università Cattolica del Sacro Cuore, Via Emilia Parmense, 84, Piacenza, 29122, Italy

ARTICLE INFO

Keywords:
Plant health models
Plant disease models
Plant phenology models
Agrometeorological data
Open source
Python package

ABSTRACT

The paper presents PyPlants, an open-source object-oriented Python package, which includes plant models validated and published in the scientific literature, useful for predicting phenology and diseases. These tools are essential in agriculture, as they help to predict growth phases and disease risk, reducing the probability of crop loss. As of today, these models are rarely distributed as software; the few available implementations are developed in different programming languages and provide limited interoperability. PyPlants addresses these limitations by offering a common software interface and a simple framework to facilitate model adoption, for both researchers and developers. Moreover, users can take advantage of this tool to benchmark and consequently share new models. The current release includes eleven different models, ten for diseases and one for phenology, but it is ready to include many more in the near future.

Metadata

Code metadata.

Nr.	Code metadata description	Metadata
C1	Current code version	v0.1.1
C2	Permanent GitHub link to code/repository used for this code version	https://github.com/iXemLabs/pyplants
C3	Legal Code License	Apache-2.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python 3.8 +
C6	Compilation requirements, operating environments & dependencies	None
C7	If available Link to developer documentation/manual	https://pyplants.readthedocs.io/latest/
C8	Support email for questions	giovanni.colucci@polito.it

1. Motivation and significance

Plant health modeling provides tools useful for minimizing crop loss caused by pests, through the prediction of phenology and disease. Disease models aim to evaluate infection risk, whereas phenology models aim to predict plant development stages. Since their introduction in the early 1960 s [1], modeling techniques have evolved significantly: from empirical models, based on expert-rules that encapsulate domain knowledge gained from real world observations and evidence, to mechanistic models that describe biological phenomena by simulating the underlying system as a set of interacting mechanisms, emphasizing direct cause-and-effect relationships. In literature, an ever-increasing number of models exists, with the largest contributions in the last 25 years. This evolution is primarily driven by a greater availability of

* Corresponding author.
Email address: giovanni.colucci@polito.it (G.P. Colucci).

https://doi.org/10.1016/j.softx.2026.103062
Received 7 August 2026; Received in revised form 14 September 2026; Accepted 20 September 2026
Available online 25 September 2026
2352-7110/© 2026 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

agrometeorological data obtained from sensors and the improvements in research tools [2].

From a black-box perspective, depending on the model, agrometeorological data such as air temperature and leaf wetness are often used as input to the system [2,3]. Each model usually runs on an hourly or daily timestep [4]; at each execution, different parameters, such as spore release and infection risk, are generated as output. However, integration of these models is not simple, because of the lack of proper software implementations and interface standardization [4,5]. Moreover, models in the literature are developed without taking interoperability into account. For example, output values such as the spore release could be represented as a dichotomous, categorical or likelihood variable. Finally, building an implementation from scratch requires not only programming skills, but also context and domain knowledge.

In this paper we present PyPlants, an open source object-oriented Python package that implements different disease and phenological predictive models that have been proposed and validated in the scientific literature. The main purpose of this library is to enable both developers and researchers to use those models with ease. Methods included in PyPlants implement a common software interface hiding the implementation complexity; additionally, a basic framework for the development of new models is provided.

2. Software description

PyPlants is implemented entirely in Python; it has no third-party libraries or platform-specific dependencies. This makes the installation straightforward across different operating systems and Python environments.

2.1. Software architecture

The library is composed of two main sub-packages, one for the diseases and one for the phenology. Models are implemented as simple Python classes that inherit from an appropriate parent. In Fig. 1, a simplified UML diagram shows the main base classes. The *update* method is the public interface implementing a common template, thus concrete models must provide the implementation of the private method with the proper logic.

Since input data are heterogeneous, agrometeorological parameters are grouped together into an update context object, implemented as a Python *dataclass*. In this way, a common method signature can be shared across different models.

2.2. Software functionalities

2.2.1. Disease models

Plant disease models predict various aspects of a disease's infection cycle. They output a list of events, where each item in the list is computed during the execution of the update method.

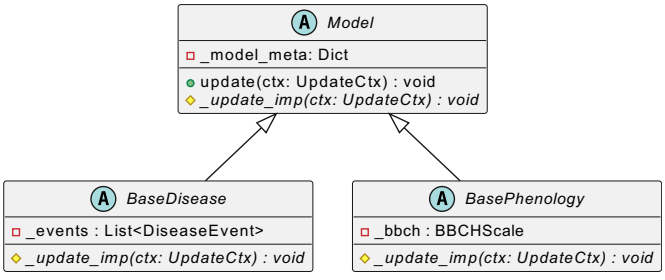


Fig. 1. UML class diagram describing the base classes of PyPlants. Green circles denote public methods, yellow diamonds denote private methods, red squares denote private attributes, and italic labels indicate abstract methods.

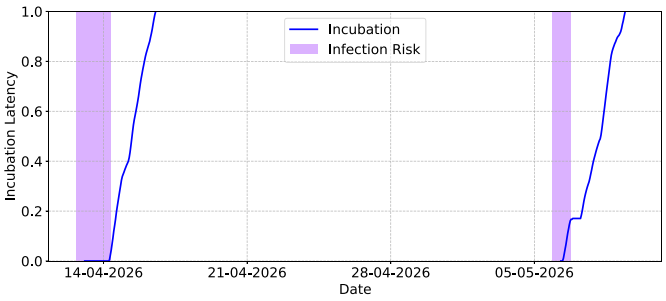


Fig. 2. Example of the Plasmo model output using data collected in a vineyard near the city of Carema in Piedmont (Italy). Purple regions denote predicted infection risk periods; the blue curves represent the incubation latency progression.

An event is composed of a timestamp, stored as a Python *datetime* object, and a number of parameters which represent the output of the model. This approach is general and applicable to different diseases across a wide variety of plants. Normally, the models compute the infection risk, which is stored as a float value, to take into account the heterogeneous types of outputs of different models. Moreover, in case of fungal infections, the spore release can optionally be computed in a dedicated float field. Other output parameters can be stored in an additional field, implemented as a Python dictionary; this is useful to track state variables that can be easily inspected to assess model functionality.

Some plant disease models are not limited to the prediction of infection risk, but they are also capable of tracking the incubation period. To support this functionality, a simple infection manager is included in the library, designed following a slightly simplified version of the strategy pattern [6]. The infection manager is model-agnostic: it takes as input a callable object, delegating the computation of incubation latency to the model itself, avoiding the duplication of the basic logic. Therefore, whenever a model receives new input data, the infection manager updates the latency for the active infections; the incubation period is considered complete when the latency reaches a threshold value of 1. Fig. 2 shows an example of the disease output computed by the Plasmo model [7] for the downy mildew of the Nebbiolo grapevine. The model has been executed using data measured in a vineyard located near the city of Carema in Piedmont (Italy) during the season 2026. In the chart, the purple regions indicate periods with predicted infection risk, whereas the blue curves represent the progression of incubation latency.

Diseases are organized in a hierarchical structure, where subpackages include all models applicable to a specific plant. Inside each sub-package, models of the same disease are grouped in the same Python module. Currently, the majority of available models target grapevine diseases, plus one for apple scab and one for generic plant fungal-based disease. In Table 1, models are reported together with bibliographic references and the relevant output parameters.

2.2.2. Phenology models

Phenology models predict the evolution of plant growth over time. Phenology can be described in different ways, with the BBCH scale being the most widely used one [17]. This scale provides a common way to track growth stages for different crop types. Each stage of the scale is represented by a combination of two numeric codes; the former represents the main growth stage, the latter the secondary one. The code progression is not linear and values cannot be sorted as regular numbers. Every plant is characterized by two main phases: vegetative and reproductive, represented by the first code value being between 0 and 4, or between 5 and 9 respectively. The two phases can overlap in time: when the reproductive scale starts, the vegetative growth (such as leaf

Table 1
List of plant disease models implemented, each model can predict infection risk and, in case of fungal-based diseases, spore release; additionally, some models track the infection incubation period.

Crop	Disease	Model	Spore Release	Infection
Grape	Downy mildew[b]	Plasmo [7]	No	Yes, with incubation
Grape	Powdery Mildew[a]	Gadoury [8]	Yes	Yes
Grape	Powdery Mildew[a]	Moyer [9]	Yes	No
Grape	Powdery Mildew[a]	Davis [10]	Yes	No
Grape	Powdery Mildew[a]	Caffi [11]	Yes	Yes, with incubation
Grape	Gray Mold	Broome [12]	No	Yes
Grape	Gray Mold	González-Fernández [13]	No	Yes
Grape	Gray Mold	González-Domínguez [14]	No	Yes
Apple	Scab	Mills [15]	No	Yes
Generic	Generic	Magarey [16]	No	Yes

[a] Implementation focused on ascospore infections
[b] Implementation focused on oospore primary infections

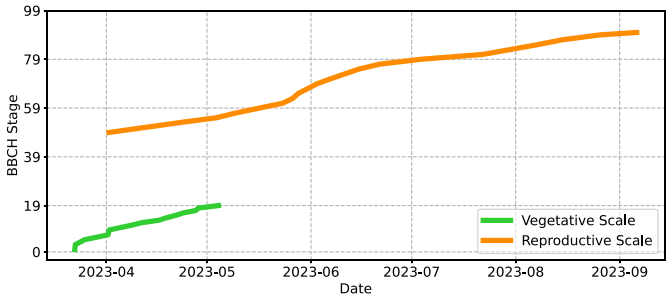


Fig. 3. Example of BBCH scale computed for grapevine cultivar, in south of Alessandria in Italy during the season 2023. The plot highlights the vegetative and reproductive phases in green and orange respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

development) may still be in progress. In Fig. 3, an example of BBCH scale progression for grapevine cultivar, computed in 2023 season in Piedmont (Italy) is shown.

In PyPlants, an implementation of the BBCH scale is provided. The scale is implemented as two lists, one for the vegetative phase and the other for the reproductive phase; each item in these two arrays is an object, composed of a timestamp and a BBCH stage. A single stage is represented as an object wrapping an integer value. The BBCH scale implementation allows comparison only between stages within the same growth phase; otherwise, an exception is raised. This is convenient because the scale is often used by plant disease models to define their running periods during the season.

Currently, only one model is included in the library: the Iphen model developed for grapevine [18], which uses the normal heating hours (NHH) [19], to track the growth evolution of the plant and consequently to predict the phenological stage. The NHH is the normalized sum of the mean hourly temperatures, using three cardinal values: the minimum, the optimum and the maximum. Since growth rates vary across different cultivars, this model takes as input the grapevine variety, in order to select the correct parameters used to run the model.

3. Illustrative examples

The library is used by initializing the models and invoking the update method as soon as new agrometeorological data are available. While phenological models usually run autonomously, plant disease models often require phenological data, which can eventually be generated by means of a phenological model. For this reason, phenology models must be updated prior to disease ones. Furthermore, according to each model requirements, the update can be on a daily or hourly basis. While hourly models are restricted to hourly data, daily models may accommodate both input types. To streamline the processing, a utility, implementing the same update interface used by the models, is included. It allows users to perform bulk updates across multiple models using granular hourly data, bridging the gap between different temporal resolution requirements. An example of library usage is presented in Listing 1, where the classic workflow for model initialization and execution is demonstrated.

```python
from pyplants.core.context import UpdateCtx
from pyplants.utils.plants import PlantEnum
from pyplants.utils.helpers import DailyUpdater
from pyplants.phenology.iphen import Iphen
from pyplants.diseases.grape.pm import Gadoury, Moyer, DavisRI

# Init the phenology model
iphen = Iphen.build_from_plant(PlantEnum.GRAPE, "CHARDONNAY")
# Models with daily timestep
moyer = Moyer()
gadoury = Gadoury(iphen)
# Models with hourly timestep
davis = DavisRI(iphen)

updater = DailyUpdater([gadoury, moyer])

uctx = UpdateCtx(...) # A sample data

iphen.update(uctx)
davis.update(uctx)
updater.update(uctx)
```

Listing 1. Usage example of models with different timestep. The daily updater utility takes as input a list of models with daily timestep. Additionally, some plant disease models require a phenology model, as input; in this example, the Iphen method is initialized selecting the Chardonnay as cultivar.

The library provides basic functionality to build custom models. In Listing 2, the skeleton of an empty plant disease model is provided as a reference. Besides inheriting from the appropriate parent, a model must define specific metadata: the timestep, the required fields in the update context, and, optionally, a BBCH range. Thanks to this definition, the library can activate some automatic consistency checks. Concerning the timestep, the system prevents updates from being executed with a mismatched one. About the update context, an automatic procedure verifies the presence of the required fields. The BBCH range is only required for plant disease models using phenological data; in this case, the system executes the update only when the BBCH stage falls within the specified range. If correctly defined, a custom model can be used exactly as any other model defined inside the package. Finally, to define a custom phenological model, the same procedure can be used, by simply changing the parent class.

4. Impact

The proposed package has potential impact on agronomists, plant pathologists, researchers, students and software developers. Agronomists and plant pathologists can find a multitude of models in the scientific literature, rarely paired with a code implementation. Even when a software tool is available, it might be distributed as closed source, thus making it difficult to verify. The package fills this gap, offering immediate usability and an open source distribution.

Researchers may use models under simulated future weather conditions to evaluate the impact of climate change on plant growth and pests [20]. Moreover, the availability of a common software interface across models, allows an easier comparison of different methods; consequently, researchers can choose the most suitable one for the specific case under study. The open implementation offers the possibility to fine tune model's internal parameters, potentially allowing its

```python
from pyplants.core.base import BaseDisease
from pyplants.core.context import UpdateCtx, CtxField

class CustomDisease(BaseDisease):
    _model_meta = {
        "timestep": "d",
        "use_ctx_fields": {CtxField.TMEAN, CtxField.RAIN}
    }

    def _update_imp(self, update_ctx: UpdateCtx):
        """Model logic goes here."""
        self._events.append(...)

uctx = UpdateCtx(...) # A sample data
# Init and run the model
model = CustomDisease()
model.update(uctx)
# Read the events property for output
print(model.events[0])
```

Listing 2. Template of a custom plant disease model that requires mean temperature and rainfall data as input parameters, running on a daily timestep.

adaptation to peculiar scenarios. For example, the pathogens response to temperature variations is described by parameters called cardinal temperatures. These values are peculiar to the pedoclimatic and morphological characteristics of each single region; therefore, they are evaluated experimentally. In this context, the package could be exploited to study models performance, when applied in different environments, and to assess, thanks to additional observations, the pests' ability to adapt to new climate conditions.

As far as students are concerned, the library could serve as a learning resource, by providing transparent reference implementations that can be inspected and modified by students.

For the developer community this package may represent a useful instrument, being a consolidated basis for further implementations of decision support systems (DSS), by reducing the amount of required code and avoiding the repetition of control logic. This simplifies the implementation of tools that allow agronomists to optimize pesticide application, reduce chemical usage, and adapt crop management to climate variability. Additionally, users developing their own models, may adopt the package as a framework, taking advantage of the available core engine.

5. Conclusion

PyPlants provides a collection of disease and phenological models for plants. The package aligns with the FAIR (Findable, Accessible, Interoperable, and Reusable) principles: the source code is publicly available on GitHub [21] under the Apache License 2.0, and the documentation is hosted on the Read the Docs platform.

Although the library currently includes a limited number of models, compared with the vast availability in the scientific literature, the package represents a starting point for new contributions. For this reason, future development will focus on expanding the library to include additional models and crop types, alongside more robust data management logic. Furthermore, to facilitate adoption by users with less familiarity with subjects such as object oriented programming, alternative model declaration constructs could be proposed.

CRediT authorship contribution statement

Giovanni Paolo Colucci: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Irene Salotti:** Writing – review & editing, Investigation, Formal analysis, Conceptualization. **Paola Battilani:** Writing – review & editing, Supervision, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Daniele Trinchero:** Writing – review & editing, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by CSR 2023-2027, Bando SRG01 - Sostegno ai Gruppi Operativi PEI-AGRI Regione Piemonte with the following projects: "Advanced, Green and Reliable Instruments for Climate Change Adaptation and Plant Monitoring"gs1 (AgriCAM) - project CUP J17F26000070009; "INnovazione per il cambiamento climatico nella VIticoltura dei Territori Eroici Piemontesi" (IN-VITE) - project CUP J17F26000060009.

References

[1] Van der Plank JE. Plant diseases: epidemics and control. Elsevier; 2013.
[2] González-Domínguez E, Caffi T, Rossi V, Salotti I, Fedele G. Plant disease models and forecasting: changes in principles and applications over the last 50 years. Phytopathol ® 2023;113(4):678–93.
[3] Bregaglio S, Donatelli M, Confalonieri R, Acutis M, Orlandini S. Multi metric evaluation of leaf wetness models for large-area application of plant disease models. Agric for Meteorol 2011;151(9):1163–72.
[4] Donatelli M, Magarey RD, Bregaglio S, Willocquet L, Whish JP, Savary S. Modelling the impacts of pests and diseases on agricultural systems. Agric Syst 2017;155:213–24.
[5] Triki HE, Ribeyre F, Pinard F, Jaeger M. Coupling plant growth models and Pest and disease models: an interaction structure proposal, MIMIC. Plant Phenomics 2023;5:0077.
[6] Gamma E. Design patterns: elements of reusable object-oriented software. Pearson Education India; 1995.
[7] Orlandini S, Gozzini B, Rosa M, Egger E, Storchi P, Maracchi G, Miglietta F. Plasmo: a simulation model for control of plasmopara viticola on Grapevine 1. EPPO Bull 1993;23(4):619–26.
[8] Gadoury DM, Pearson RC. Ascocarp dehiscence and ascospore discharge in uncinula necator. Phytopathology 1990;80(4):393–401.
[9] Moyer MM, Gadoury DM, Wilcox WF, Seem RC. Release of erysiphe necator ascospores and impact of early season disease pressure on vitis vinifera fruit infection. Am J Enol Vitic 2014;65(3):315–24.
[10] Gubler W, Rademacher M, Vasquez S, Thomas C. Control of powdery mildew using the UC Davis powdery mildew risk index, APSnet feature. In: Published online. Paul, MN: The American Phytopathological Society, St.; 1999.
[11] Caffi T, Rossi V, Legler SE, Bugiani R. A mechanistic model simulating ascosporic infections by erysiphe necator, the powdery mildew fungus of Grapevine. Plant Pathol 2011;60(3):522–31.
[12] Broome J, English J, Marois J, Latorre B, Aviles J. Development of an infection model for botrytis bunch rot of grapes based on wetness duration and temperature. Phytopathology 1995;85(1):97–102.
[13] González-Fernández E, Piña-Rey A, Fernández-González M, Aira M, Rodríguez-Rajo FJ. Identification and evaluation of the main risk periods of botrytis cinerea infection on Grapevine based on phenology, weather conditions and airborne conidia. J Agric Sci 2020;158(1–2):88–98.
[14] González-Domínguez E, Caffi T, Ciliberti N, Rossi V. A mechanistic model of botrytis cinerea on grapevines that includes weather, vine growth stage, and the main infection pathways. PLoS One 2015;10(10):e0140444.
[15] N. Y. S. C. of Agriculture. Extension Service, Mills W. Efficient use of sulfur dusts and sprays during rain to control apple scab; 1944.
[16] Magarey R, Sutton T, Thayer C. A simple generic infection model for foliar fungal plant pathogens. Phytopathology 2005;95(1):92–100.
[17] Meier U, Bleiholder H, Buhr L, Feller C, Hack H, Heß M, Lancashire PD, Schnock U, Stauß R, Boom TVD, et al. The bbch system to coding the phenological growth stages of plants–history and publications. J für Kult 2009;61(2):41–52.
[18] Mariani L, Alilla R, Cola G, Monte GD, Epifani C, Puppi G, Osvaldo F. Iphen—a real-time network for phenological monitoring and modelling in Italy. Int J Biometeorol 2013;57(6):881–93.
[19] Cola G, Mariani L, Parisi S, Failla O, et al. Tempo termico e fenologia della Vite. ITALUS HORTUS 2012;3:31–4.
[20] Raza MM, Bebber DP. Climate change and plant pathogens. Curr Opin Microbiol 2022;70:102233.
[21] iXemLabs. pyplants: a collection of phenological and disease models for plants; 2026 https://github.com/iXemLabs/pyplants [accessed: 20 July 2026].