BESOS: a Python library that links EnergyPlus with optimization and machine learning tools

Theodor Victor Christiaanse\textsuperscript{1,2}, Paul Westermann\textsuperscript{1,2}, Will Beckett\textsuperscript{2}, Gaëlle Faure\textsuperscript{1,2}, Ralph Evins\textsuperscript{1,2}

\textsuperscript{1}IESVic, Victoria, BC, Canada
\textsuperscript{2}Energy in Cities group, CIVIL Engineering, Victoria, BC, Canada

Abstract
This paper presents the \textit{besos} Python library, which can parameterize EnergyPlus models and integrate these with the Python ecosystem of tools including machine learning and optimization libraries. This library underpins the BESOS (Building and Energy Simulation, Optimization and Surrogate-modelling) platform. The need for a flexible Python-based library that integrates these domains is outlined. A case study is presented to demonstrate the benefits of this integrated approach, and an overview is given of research works that have leveraged these benefits. We also discuss lessons learned while developing, deploying and documenting this open-source library.

Key Innovations
\begin{itemize}
\item A Python library to parameterize EnergyPlus models
\item Aids integration with machine learning and optimization libraries and the wider Python ecosystem
\item Tutorials provided through Jupyter Notebooks
\item Case study demonstrating benefits of integration
\end{itemize}

Practical Implications
EnergyPlus modelling is challenging for many non-experts, and parameterization of these models is cumbersome using built-in tools. This library addresses these issues by making it straight-forward to run and edit EnergyPlus models in a simple computational environment.

Introduction
This paper presents the \textit{besos} Python library that enables parameterization of EnergyPlus models for integration with Python machine learning tools and optimization toolboxes. EnergyPlus is a widely used physics-based whole building energy simulation tool. \textit{besos} also integrates the Energy Hub\textsuperscript{1} mixed-integer linear programming framework for energy system optimization, but that is not the focus of this paper. Parametric energy modelling is becoming increasingly common, since designers need to explore the design space of possible buildings rather than manually simulate single designs. This requires a method of easily defining which elements of a model should be varied and how, the combination of parameters to vary to explore a given problem along with the outputs to be tracked, and methods of sampling across this space, running optimization algorithms over the parametric models, and linking these to machine-learning algorithms for surrogate modelling. Surrogate modelling is the process of fitting a machine-learning model to synthetic data from a detailed simulator to allow rapid exploration of the design space Westermann and Evins (2019a).

Python is the fastest growing programming language by many measures, and has a plethora of libraries spanning scientific topics (the sci-py stack) as well as data science, visualization and web development, all supported by a thriving community. Many machine learning and optimization techniques are most readily available in Python. It is important for the building simulation community to build bridges to this area of rapid development and innovation in order to advance our field as effectively as possible. Linking to other existing codes and libraries is a central tenet of the multi-model ecology paradigm Bollinger et al. (2018), in which many small computational entities are developed together to meet changing needs.

Several tools are available to parameterize EnergyPlus models, including Energy Incites\textsuperscript{1} and Energy Hub\textsuperscript{1}. These tools are designed to help designers explore the design space of possible buildings, but they lack the flexibility and ease of use of a Python-based library.

\textsuperscript{1}\url{https://gitlab.com/energyincities/python-ehub}
Plus models. JEPlus uses the Java programming language and allows for parameterization through a graphical user interface (Zhang and Korolija (2010)). EPLUSR uses the R programming language (Jia and Chong (2020)). EPPy (Philip (2020)) provides EnergyPlus IDF file manipulation capabilities in Python, and we make use of EPPy in the Besos library, adding layers to facilitate parameterization of models and integration with other libraries. As with all these tools, Besos works with a user-supplied EnergyPlus building model, though future work is planned to address this. The Python programming language has many benefits. Python is free to use, unlike MatLab, which has been a major barrier to deployment of MatLab based tools in industry for many years. Python is available for all major operating systems. Due to its strong focus on code readability, it has become a popular choice to teach programming skills (Badică et al. (2020)). The flexibility of the syntax makes it easy to learn Python, however for software development consistent syntax across a codebase is important. For this, automated formatting software\(^2\) to standardize syntax (van Rossum et al. (2001)) across multiple files is available. Python is among the most popular programming languages as of 2020, driven by academics teaching Python and professionals adopting Python (Cass (2020)).

There are many Python libraries for machine learning and optimization. Python is home to two mature machine learning libraries which we leverage in Besos: scikit-learn (Pedregosa et al. (2011)) and TensorFlow (Abadi et al. (2016)). Similarly for optimization, toolboxes have been created that implement standard optimization techniques (Hadka (2021)) as well as novel optimization techniques such as RB-Fopt (Costa and Nannicini (2018)). The Python pandas library provides numerous fast and comprehensive tools to analyze and manipulate data. The pandas DataFrame object is a flexible and adaptive data storage object that all the above tools use (McKinney (2010)).

The Besos library aims to integrate methods from a broad range of research fields which prevent researchers having to reinvent the wheel. Dealing with multiple libraries and making sure they all work well together requires some coordination regarding operation and functionality of each of the different libraries. By controlling software versions, this can be precisely maintained. The control of the dependent libraries are managed using the Python virtual environment. The virtual environment allow library versions to be fixed or set to a version range that is acceptable.

The Besos is release as open-source under the so-called copyleft license GNU GPLv3. This license places few restrictions on using the code, but requires modifications to the code to be published under the same licence so that they remain available to the research community. Open-source libraries may also come with risks: they may be less well maintained and documented compared to commercial software, can contain bugs or even malicious code, or development can cease. These issues can be mitigated by making use of libraries with good community support, keeping copies of critical code and auditing third party code.

In the next section we will cover the structure of the Besos library, methods to parameterize EnergyPlus models, and how inputs and outputs integrate with other Python libraries. Next we showcase some of the functionality of the library through a case study, as well as presenting an overview of recent research facilitated by Besos. Lastly, to aid other members of the community in developing open-source libraries, we detail the lessons learned in developing, documenting and deploying the library.

BESOS

The Besos library\(^3\) underpins the BESOS (Building and Energy Simulation, Optimization and Surrogate-modelling) platform (Faure et al. (2019)). Building simulation refers to standard physics-based building modelling via EnergyPlus. Broader energy system simulation is implemented in the Energy Hub model\(^4\). Optimization using methods like Genetic Algorithms (Evins (2013) is provided by the Python library Platypus (Hadka (2021)). Surrogate modelling is typically accomplished by simulating a large set of samples with a parameterized EnergyPlus model, then fitting a scikit-learn or TensorFlow machine-learning model

\(^2\)https://github.com/psf/black
\(^3\)https://gitlab.com/energyincities/besos/
\(^4\)https://gitlab.com/energyincities/python-ehub
to these results.

A key functionality of besos is to parameterize EnergyPlus models, either to connect with optimization approaches like Genetic Algorithms or to sample the design space as part of a surrogate modelling process. A parameterized problem is composed of a base simulation model, input parameters to vary and outputs of interest to be tracked. The base model is typically an EnergyPlus model (IDF or epJSON file).

The core besos object used to parameterize an EnergyPlus model is the Parameter that defines an aspect of the model to vary and how this should be varied. Each Parameter is composed of a Selector and a Descriptor. The Selector specifies the definitions in the model file that will be changed by the Parameter. The Descriptor provides the range or list of possible values for that Parameter. There are three types of Selectors:

- FieldSelector selects a single field in the model, e.g. the 'Thickness' field of a 'Material' object with name 'Wall Insulation'.
- FilterSelector applies a filter to select multiple fields, e.g. the 'Height' field of all 'Window' objects;
- GenericSelector provides a way to make more complicated changes to the model. It takes three functions: a 'setup' function for any preprocessing, a 'set' function which changes the model based on a single input value, and a 'get' function which returns the input parameter given the model.

These choices allow simple single and multiple points in the model to be specified in a single line using FieldSelector and FilterSelector respectively, while GenericSelector allows much more detailed model modifications. An example of the latter is varying the window to wall ratio. This implementation first defines a single window on each wall surface in the a setup function. The set function then sizes the windows to the required window to wall ratio based on the window and wall vertexes. The get function is defined to find the window to wall ratio based on the window and wall areas calculated from the vertexes definitions.

There are also two forms of Descriptor used to specify acceptable values:

- RangeDescriptor defines a range using maximum and minimum values.
- CategoryDescriptor gives an arbitrary list of items, e.g. strings or numerical values.

The other modules in the besos library that are used to solve parameterized models are shown in Fig. 1. They include:

- Objectives that define the simulation result(s) of interest, for example for use in the objective function of an optimization. These are taken from the EnergyPlus ESO output file based on Variable and Meter definitions. Outputs can be passed directly to a pandas DataFrame, or passed through a custom function for aggregation or transformation.
- Problem objects combine multiple Parameters and Objectives (i.e. inputs and outputs) to define a specific EnergyPlus design space to be explored;
- Evaluator objects associate a Problem with a specific execution method (e.g. EnergyPlus), model (e.g. an IDF file) and context (e.g. EPW weather file). This also includes options for multiprocessing powered by Dask (Dask Development Team (2016)), error handling and output saving.
- The Sampling module executes an Evaluator to generate a specified number of samples across the design space. This includes Latin hypercube and full factorial sampling, and options for distributions for random sampling.
- The Optimizer module links an optimization algorithm to a specific Evaluator. This links to the Platypus optimization library (Hadka (2021)) and the novel RBFOpt method (Costa and Nannicini (2018))
- Surrogate modelling functionality is provided via the machine-learning libraries scikit-learn and TensorFlow. These are trained on a pandas DataFrame containing sampling results consisting of input-output combinations.

There are also more advanced ways to combine Evaluator objects. Evaluators can also be nested inside a GenericEvaluator object, if for example an optimizer
needs to run multiple EnergyPlus models or combine an EnergyPlus model with an Energy Hub model. A surrogate model consisting of a fitted machine-learning model can also be wrapped inside an Evaluator object so that an Optimizer can be easily reused between different problems, including when EnergyPlus object names change if FilterSelector and GenericSelector functions are configured appropriately. Problem definitions can group together commonly-used Parameter and Objective definitions, and can be reused across different IDF files and EPW files very easily. Different sampling and optimization approaches can be applied to specific Evaluators.

In the following section, we demonstrate the utility of the besos library and of this approach with a case study. We define a problem consisting of parameters with ranges, link these to an EnergyPlus model, then run an optimization algorithm over the problem, and finally sample the problem space and fit a surrogate model to the results. This is accomplished in 10 lines of code, something that would not be possible without the flexibility and power of the Python language and the integration with the Python ecosystem.

Case Study

The process in Fig. 2 shows an example of how to use a single Parameter to vary the window conductivity in an EnergyPlus model. This model uses a theoretical glass material in the IDF file as a window glazing material. The FieldSelector is used to select this material using the class name ‘WindowMaterial:Glazing’, object name ‘Theoretical Glass [176]’ and field to change ‘Conductivity’. The conductivity range is specified using the Descriptor to be between 0.9 and 6 W/m²·K. Combining these gives a Parameter called ‘Conductivity [W/m²-K]’, which could now be reused in other contexts. For this example, the Problem is setup using window conductivity and multiple parameters that include wall insulation thickness and window to wall ratio. These are setup using the FilterSelector and the GenericSelector as seen in Fig. 3. The Objectives are set to be the total heating and total cooling demands. Finally, an Evaluator is defined to link the IDF file to the Problem.

In Fig. 4 we present several possible next steps after the setup of the Evaluator. We can use an optimization algorithm to find the Pareto front of optimal designs based on the design space defined by the Parameters (I). We are can sample the design space randomly to understand the correlations between the input parameters and the objectives. We can then use this dataset to create a surrogate model. The figures on the right show examples of plots that can be generated using plotting libraries available in Python. The results from the optimization of this example problem show that there is a clear trade-off between heating and cooling demand. The scatter plot from the sampling of the problem also shows that there are any designs for which both heating and cooling demand are very far. Using the correlations plot of the sampled dataset we can see there is a clear correlation between window to wall ratio and heating demand which is not present between cooling demand.

Figure 1: An overview of the structure and flow of the besos library objects
and window to wall ratio. The surrogate model results show the results from the E+ model as dots and the results from the surrogate model as the surface plot.

It would much more challenging to implement a parametric analysis and surrogate modelling process like this without a Python library that allows us to easily parameterize the underlying EnergyPlus model and to link this to the optimization and machine-learning algorithms available in Python.

Examples of use in research

A wide range of research applications have used the besos library. Westermann and Evins (2019b) explored the use of adaptive sampling for surrogate modelling, in which successive generations of samples are performed to refine the model in areas where precision is lacking. Westermann and Evins (2020) extended the analysis of errors in surrogate models by using a Bayesian formulation that can identify areas of poor accuracy and trigger the evaluation of the high-fidelity simulation. Westermann et al. (2020) applied surrogate models over a wide range of weather data, using Convolutional Neural Networks in TensorFlow to fit a model which can take weather data as an input and produce hourly data as an output. Bhatta et al. (2020) trained surrogate models that takes building geometry parameters as an input. Baasch et al. (2019) generated a large synthetic data set to compare data-driven building characterization methods. The library has been used to generate the surrogate models that are linked to a web dashboard for early stage building design (Westermann et al. (2020)).

This breadth of research applications demonstrates the benefits of the integrated Python-based approach used in besos. Researchers and software developers have worked iteratively together to provide a workable structure that makes use of best-practices like object-oriented programming. The library has evolved to meet the needs of researchers, and this has been relatively straight-forward as it builds on the strong integration to other libraries available in Python and the flexibility of the core besos functionality. In Westermann et al. (2020), an undergraduate student new to simulation was able to easily perform advanced parametric analyses, and to incorporate cutting-edge developments in machine-learning via the TensorFlow library. It would not have been possible to achieve the range of novel developments achieved in a short time without a library that bridges simulation tools and the wider Python ecosystem.

Figure 2: Setup of an Evaluator from an IDF model. The Parameter varies the thickness of the insulation within the wall assembly.

Figure 3: The setup of selecting all insulation thicknesses used in various wall assemblies and the use of the GenericSelector for setup a window to wall ratio parameter.

https://netzeronavigator.ca/
Development recommendations

In this section, we share our experiences in developing the besos library, for the assistance of other researchers who might wish to develop open-source software to help bring their tools and processes to other researchers.

The use of a version control system is essential. We use Git to track different versions of code, allowing changes to be merged where needed, and also multiple 'branches'. The main branch contains the most recent stable version, and can be linked with package managers like pip so you can install besos in Python using "pip install besos". We use a development branch for testing new features, and specific branches for each feature as it is being developed.

Our codebase in a Git repository is hosted on the Gitlab cloud-based hosting service (Github and BitBucket are alternatives) which provides remote storage of the codebase for collaborative development by many users plus productivity and team management tools. We use the Gitlab Issues Board to track problems with the code and assign them to users.

Testing is critical to ensure that functionality is not impacted by changes elsewhere in the codebase. Continuous Integration (CI) on runs tests automatically every time a change is made. The pytest library is used to define specific tests.

The Journal of Open Source Software (Westermann et al. (2021)) is a great place to submit your code and to receive tips on how to to improve it.

Clear and comprehensive documentation takes a lot of work, and readers will see that this is still in progress. Most of our documentation is on ReadTheDocs⁶, which is automatically generated based on restructured text documents and docstrings⁷, which are descriptions added to classes etc directly in the source code. This helps keep the documentation up to date when code changes are made, and can also populate contextual help like code completion suggestions and popups that list method arguments.

We also maintain the BESOS platform⁸ (Faure et al. (2019)), an online environment that provides Jupyter Notebooks containing lots of examples of use of the besos library. Jupyter Notebooks are executable Python documents in which you can interweave code and text. We currently provide storage and computing following a fair-use policy. New users can start exploring the examples without the need of installing Python or EnergyPlus, and all users have

---

⁷https://www.python.org/dev/peps/pep-0257/
⁸http://besos.uvic.ca
a unified environment with the latest stable version of the besos library. While this has been a lot of work to implement and maintain, it has proven to be a stable platform for all group members to conduct research, with more computational power than would be available on local machines but without the barrier of transitioning to a cluster.

The initial code development on the besos library stems from graduate research work, but there are limits on the time students should commit to software development, and it may be hard to maintain code quality if development is entirely performed by people unfamiliar with software development best-practices. The code structure and modules were largely developed by undergraduate students with a background in computer science and software development. This brings other issues in ensuring that the purpose of the code is clearly explained to non-experts. It is critical to discuss frequently with research students so that misunderstandings are caught early. Involvement of a group leader who has an overall vision for the codebase and awareness of all the moving parts is important. A permanent employee with knowledge of the code is also needed to maintain institutional knowledge as students and developers leave the team.

Developing, deploying, documenting and maintaining open-source code is time intensive, but having the right systems in place can improve the process. It is regrettable that standard academic metrics like publication numbers do not reflect the effort that goes into developing such codes or the value that can be gained from them. However, we encourage others to find efficient ways to make their codes available to others, which can dramatically increase research impact and will hopefully lead to collaborations and citations that boost traditional academic metrics.

Conclusion

The software available for building energy modellers is constantly evolving and expanding. This progress, particularly in other domains like machine-learning, has great potential to advance the capabilities of the energy modelling community. In this paper we have presented the besos library as an example of a powerful open-source codebase that provides a link between parameterized EnergyPlus and the huge variety of libraries available in Python. A case study is shown to demonstrate the power of this approach, along with examples of research works that have leveraged these capabilities. We have also summarized some salient points to help other community members to develop such a library.

The development of open-source libraries that link existing energy simulation tools with the Python ecosystem provides many benefits to the energy modelling community in terms of breadth of application and integration, speed of development of new approaches and ease of adoption by new users and other researchers.

Acknowledgements

The besos software library was supported by the work of many graduate and undergraduate students and grant funding from CANARIE via the BESOS project (CANARIE RS-327).

References


Bollinger, L. A., C. B. Davis, R. Evins, E. J. L. Chap...


