Introduction to Pflacs: Faster load cases and parameter studies with Python

Stephen McEntee

Last edited on 2019-09-24

Contents

Preface

Abstract

The engineering design process has a significant computational component involving analysis of multiple load cases and parameter studies, with the aim of identifying a combination of design parameters that yields an optimal design solution. Traditionally engineering design methodolgies have been very manual and iterative, however recent developments in computer technologies are driving a growing trend towards automation. This article presents Pflacs an open-source Python package that takes advantage of Python's flexible dynamic nature and its introspection tools to provide an object-orientated framework for automating computational studies.

Pflacs binds data and Python functions together in an object-orientated fashion, and uses a tree data structure that is reflective of the hierarchical structure of many design projects. The author has a background in the subsea oil & gas industry, and has applied Pflacs to automating the design of subsea pipelines. Although its origins are in engineering design, Pflacs can be used to manage and automate parameter study type analysis in any domain.

About the Author

Stephen McEntee is a subsea engineer..

1 Introduction to Engineering Design

Reproducibility, data management and workflow management are currently areas of considerable interest and activity in computational scientific research. Recent articles have featured several interesting contributions addressing these issues [(Vyas Ramasubramani et al., 2018), (Greff et al., 2017)]

In the domains of civil and mechanical engineering analysis and design, similar issues arise due to the significant component of computational work involved. There are however also some important differences when engineering computational work is compared with scientific research. Engineering design is a very iterative process, and scheduling issues can be critical. Often engineering workscopes are executed in parallel in order to maintain project schedules, even when a serial, or waterfall, workflow might be more appropriate. This can mean that work commences with incomplete information, requiring assumptions to made in the design basis. As the project progress, more information usually becomes available, and also issues inevitably arise, which can result in changes to the design basis. This leads to frequent re-work being required, and since the engineering design process is still very manual and hands-on, re-work can be a significant burden in terms of cost and schedule delay.

1.1 A historical note

In the 1990s cheap personal computers had become powerful enough to replace the previously more powerful and significantly more expensive class of computers known as engineering “workstations”.
Where previously engineers had only restricted computer access for computational work, and relied on secretaries for typing reports, it became normal for each individual engineer to have their own desktop computer. Before the advent of the personal computer, engineering design was a mostly manual process, based on paper calculation pads, and engineers tended to favour simpler calculations, ostensibly because simpler engineering theory was considered to be more robust and conservative. However, the fact that most computational work was manual may also have been an important factor, since there is a significant cost penalty associated with increasing computational complexity when calculations are carried out by hand on account of the additional “manhours” required, particularly when re-work is required.

The arrival of the personal computer on the engineer’s desk was considered at the time to be a great advancement in terms of productivity improvement, augmentation of computational capability, and general improvement in engineering quality. In reality however, the PC was not so much of a revolution in the engineering office, it was more of an evolution from paper to the desktop computer. Apart from the fact that the actual computational work was now being carried out by computer, the actual work practices and design processes remain much the same to this day. Granted that there has been a significant increase in use of advanced computational technologies, like the finite element method, at the same time a lot of engineering design work continues to be based on the same well-understood, simpler engineering theory, except that now the caculations are carried out on electronic spreadsheets and "worksheets" that greatly resemble their paper predecessors. A time-and-motion study carried out on a junior engineer in most engineering design offices today would reveal that most of their time is spent manually entering data into computer programs, and manually copying-and-pasting the calculated results into word processors, and after that a great deal of time is spend manaully preparing over-formatted "reports" that require constant adjustments to pass quality checks (that are also still carried out manually.)

This at least has been the author's experience in working as a subsea pipeline engineer for more than 20 years in the oil & gas industry. In recent years however, a great change has come to this industry. Since 2014 an unexpected and sustained reduction in the oil price has led to calls to reduce costs, but without compromising safety. It is recognized that oil & gas has fallen behind other industries in terms of productivity and technology uptake, and that digitalization, adopting new technologies and changing the ways we work are widely regarded as means of achieving the goals of sustainably reducing costs without increasing risks.

2 Introduction to Pflacs

Pflacs (Stephen McEntee, 2016) is a pure Python module that has been developed to manage and expedite computational studies that are typically carried out as part of the engineering design process. The inspiration for Pflacs came from the author's work as a pipeline engineer in the subsea oil & gas industry :cite:`CDDsubsea`, where design and analysis work has tended to be an intensively manual and iterative process. The geometrical simplicity of pipelines, effectively 1-dimensional structures, means that it is comparatively straightforward to parameterize, compartmentalize and automate their design. That means that pipeline design is the ideal domain in which to demonstrate the capabilities and useage of Pflacs.

Computational studies, whether in engineering design or scientific research or generally, tend to be hierarchical in structure, with an over-arching fundemental *base-case* study at the root of the project, and multiple, various *load cases* or *parameter studies* that explore variations on the *base-case*. This project hierarchical structure is often exploited by computational analysts by organizing study components in directories or folders in the computer file system.

The limitions of using the computer file system to manage large computational projects quickly become evident, as it gets harder to maintain a consistent naming scheme for parameters and load cases, and other scaling issues arise as the project grows. Typically, even the best organized analyist can quickly fall into an ad-hoc approach to managing data and work flows, and this makes it more difficult to resume work at a later point or for another analyst to take over the project workscope.

The objective of PFLACS is to address these issues in a familiar `Python` computational environment. PFLACS inherits from a companion `Python` module called `vntree` (Stephen McEntee, 2019), and that makes PFLACS a tree data structure. Study input parameters become attributes of the nodes in a PFLACS tree, and when a node requires a parameter it can ascend the tree to find its value, if the parameter is not an attribute of that node. So effectively parameters can be inherited from higher levels in the tree structure.

Computational functionality is added by *plugging-in* (or *patching*) external `Python` functions, turning the functions into class attributes, or methods, that are available to all the nodes in the tree. The plugged-in functions are bound to the parameter attributes, and this means that it is not necessary to explicity specify the function arguments when a function is invoked on a `pflacs` node. If an argument is not specified in a function call, `pflacs` will substitute the value of the parameter attribute it finds with the same name as the required argument. Binding node parameter attributes to functions in this manner facilitates automation of computations.

`pflacs` achieves this by using the introspection tools provided by Python's `inspect` module. Then an external function is plugged in, `pflacs` uses the `inspect.Signature` class to obtain the call signature of the function. When the function is invoked on a `pflacs` node, the function call signature is used to match any unspecified arguments with the appropriate parameters.

`pflacs` is a lightweight and unopinionated environment, the only requirement is that the user adopts their own naming naming scheme for parameters, and maintains consistency with that scheme within the project. The idea behind this approach is to allow the user to re-use, or re-purpose, existing code without the need to alter or adapt the original code. There is no requirement to decorate or modify plug-in functions, which means that external functions can continue to be used in their original form as standalone code, or in another computational environment outside of `pflacs`. The only restriction on this is that `pflacs` plugin functions must be pure Python code, due to the dependency on `inspect.Signature` which has this limitation. In order to use `pflacs` with compiled libraries, like the functions in Python's built-in `math` module, the work-around would be to wrap the compiled function inside a Python wrapper function which can be accessed by `inspect.Signature`.

3 Basic usage

We will now further explore the use and capabilities of `pflacs` through two examples, first a very simple study that showcases basic usage, and after that a real-life example is presented demonstrating the engineering design of a subsea pipeline using `pflacs`.

References

Greff, K., Klein, A., Chovanec, M., Hutter, F., and Schmidhuber, J. (2017). The sacred infrastructure for computational research. In Huff, K., Lippa, D., Niederhut, D., and Pacer, M., editors, *Proceedings of the 16th Python in Science Conference*, pages 49–56, Austin, TX.

Stephen McEntee (2016). Cost, Design and Data in Subsea.

Stephen McEntee (2019). «vntree» a simple python tree data structure.

Vyas Ramasubramani, Carl S. Adorf, Paul M. Dodd, Bradley D. Dice, and Sharon C. Glotzer (2018). signac: A Python framework for data and workflow management. In Fatih Akici, David Lippa, Dillon Niederhut, and Pacer, M., editors, *Proceedings of the 17th Python in Science Conference*, pages 152 – 159.