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

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.

2 Literature

Here is a review of existing methods.

This chapter is an overview of the methods that we propose to solve an **important problem**.

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 1. If you do not manually label them, there will be automatic

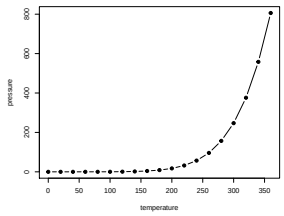

Figure 1: Here is a nice figure!

Table 1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 1.

```
knitr::kable(
  head(iris, 5), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2019) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

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.

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.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2019). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.13.