University of Auckland
Department of Accounting & Finance

FINANCE 788: Research Essay

Author: Connor McDowall
Supervisor: Rosalind Archer

August 1, 2021

Abstract

Acknowledgements

Paul Geertsema

Declaration of Contribution

Contents

List of Figures

List of Tables

1 Introduction

2 Literature Review

Insert

3 Project Scope and Research Question

4 Theory

5 Methodology and Implementation

5.1 Project organisation

GOCPI adopted Data Science best practice, as described by Wilson et al [**J:10**]. Although these practices are mostly reserved for data science projects, their principles are suitable for product development and version control. All data and results were saved regularly and reproducible. The retention of data in all forms received high levels of attention. Project files were synched continuously to Google Drive [**Google˙Drive**]. Git [**Git**] was used to manage version control for GOCPI's source code, data, documentation and results. Git stores a complete history of versions using Git hashes. These hashes are strings unique to each state of the publicly available GOCPI repository[1]. Git hashes enabled the discretisation of GOCPI's development over time, enabling the accessibility and recollection of all previous states given a unique git hash. This functionality enabled reproducibility, error correction and the ability to revert to previous models.

5.1.1 Version Control

Git, hosted by GitHub, provided a comprehensive set of version control technologies. These technologies provided a range of benefits. Firstly, Git is excellent at providing and supporting collaborative functionalities. The master version of a project is accessible for all who have access to the repository. Each contributor could create custom copies of branches through pull requests on the master branch. Contributors could commit changes to custom branches and push these changes to the master branch through push requests. The product manager could review these push requests, approving suitable requests to integrate changes to the master branch. Collaborative efforts were possible with commit messages describing the contributions from each contributor. This project had one contributor. Git ensured the histories of code, work and authors are stored. The descriptive nature of the commit log ensured an accurate journal is kept.

5.1.2 Folder Structure

GOCPI maintained the file folder structure recommended in Wilson et al [**J:10**]. Project organisation was paramount as the modelling of energy systems involves integrating a range of optimisation models, data files and documents. Wilson et al's recommendations were appropriate as data science projects require similar organisational rigor. Subsequently, file management and structure was most efficient and comprehensive. **GOCPI** is the root directory of this project and contains several sub directories: bin, data, doc, src

[1] https://github.com/CMCD1996/GOCPI

and results. The **bin** sub directory contained external scripts and compiled programmes related to the GOCPI project. The **data** sub directory contained all raw data associated with the project. This data included energy statistics, energy balance datasets, partitioned geographies, standardised optimisation models and TIMES modelling frameworks. The **doc** sub directory stored GOCPI's user guides, academic resources, research reports and project deliverables. The **results** sub directory contained the output from optimisation simulations and processed data to display on dashboards and websites to inform investment and policy decisions. The **src** sub directory stores the source code for preparing raw data, partitioning sets of geographies with varying granularities and the GOCPI python package available to download using PyPI[2] and install using pip[3]. All files were continuously backed up using Google Drive.

5.1.3 Python

Python 3.7 was the primary coding language for the GOCPI project. GOCPI's objective is to enable any user to design and model their own energy system to inform investment and policy decisions. The intention is to empower users to discuss energy investment and policy decisions made by public and private parties. Additionally, GOCPI intends to reduce misinformation regarding energy policies and help assess the feasibility of meeting the International Energy Agency's Sustainable Development Scenario [**IEA˙WEM**]. Python is omnipresent, widespread in software development. Python's language design makes the language highly productive and simple to use. Python can hand off computationally straining tasks to C/C++ and has first-class integration capabilities with these two languages. The language also has a very active and supportive community [**Python˙Features**]. In addition, Python is the most popular coding language on the planet defined by the PYPL PopularitY of Programming Language Index. As at August 2020, Python had 31.59% of all language tutorial search instances on Google [**PYPL˙Pop**]. Python has many useful packages for creating the GOCPI package such as NumPy, Scikit-learn, os, csv and Pandas. Programming is quick due to Python's dynamic nature. The language is also open-source with no cost. Subsequently, Python was the best language to ensure the GOCPI model is accessible for many users to use and extend.

5.1.4 Package Management

The Anaconda package management platform for Python [**Anaconda**] was the chosen coding environment. Anaconda is a well defined, free platform with known versions of python packages such as matplotlib, numpy and pip. The use of this environment ensured both reproducibility and consistency across infrastructure. Although this project required no collaboration, the use of Anaconda will inform future developers on how to manage collaborative processes, especially for packages which are less well-maintained. Anaconda allows you to create custom environments which was necessary for creating scalable linear optimization problems to express energy systems. Pip is Python's default package manager and is included in the Anaconda package. Pip was used to install and update packages for python not available on Anaconda such as twine and the custom GOCPI package developed for this project.

[2]https://pypi.org/
[3]https://pypi.org/project/pip/

5.1.5 Excel

It is important users are comfortable with using the GOCPI model. Energy modelling can be quite complex. The modelling process must be transparent to inform users how to build their own models. Excel is ubiquitous across academic and professional communities. Excel's omnipotence makes the software well-suited for describing the components of the GNU Mathprog energy system model. The **GOCPI OseMOSYS Structure.xlsx** file describes the sets, paramaters, constraints and objective function of a scalable energy system model. The User may toggle statement sets, parameters and constraints to adjust the complexity of the model. The model file was imported to a text file. However, data related to these energy systems was stored using Python dictionaries, lists and NumPy arrays. This Python formulation was later transcribed to a text file. Excel is best for two dimensional variables or data stored in Codd-Boyce relational databases [**CBNF**]. The majority of parameters in energy systems were three or more dimensions. Therefore, Excel was not suitable to store these parameters. Python dictionaries, lists and NumPy arrays were preferred alternatives.

5.1.6 IBM ILOG CPLEX Optimization

The OseMOSYS methodology (see **??**) translates energy systems into linear programming problems. A solver was required to optimise these user-defined energy systems. The IBM ILOG Optimization Studio [**IBM˙ILOG**], more commonly known as CPLEX, was chosen to be this solver. CPLEX solves very large linear programming problems using the Barrier Interior-point method [**IPM**] or primal/dual variants of the Simplex Method [**Simplex**]. GOCPI's user-defined energy systems could be scaled up to model very large systems, creating large linear programming problems.

The IBM ILOG CPLEX Optimization Studio has an interface with the Python language based on a C programming interface. Subsequently, Python APIs were available to run the CPLEX solver when installed either locally or on a cloud service. The python packages are **cplex** and **docplex**. The cplex package contains classes for accessing CPLEX for the Python programming language. The Cplex class is the most important class in this package as provides methods for creating, modifying, querying, or solving optimisation problems. Docplex also enables the formulation of new linear programmes where one creates the model, defines the decision variables, sets the constraints and expresses the objective function. The user uses docplex to solve the linear programme on a local solver. Alternatively, the model can be solved on a private cloud using Decision Optimisation on Cloud service through the provision of a service url and personal API key. The CPLEX Python APIs were most attractive as provided the user with a powerful commercial solver in an accessible format.

There is a caveat to the use of the CPLEX solver. The IBM ILOG CPLEX Optimization Studio is commercial by nature and requires a license to use. Fortunately IBM have the IBM Academic Initiative [**IBM˙AI**], granting students access to commercial software for free. This commercial nature creates accessibility issues for users who are not enrolled at an academic institution or can afford to pay for the software. Accessibility issues caused by the need for commercial solvers must be addressed to enable the distribution of the GOCPI product.

5.1.7 IBM Watson Machine Learning Service

The IBM CPLEX Optimisation Cplex python API is suitable for smaller models that can be solved locally. As the model increases in complexity, the docplex Python API did enable the ability to solve larger linear programmes. Unfortunately, IBM phased out the docplex Python API by incorporating the Decision Optimisation on Cloud services into the IBM Watson Machine Learning cloud services [**IBM˙WML**]. This change occurred during September 2020. This service uses IBM Cloud to access assets through credentials, create model deployments in IBM's servers and execute jobs to solve models. The model deployments must be Python-based models with jobs specifying a payloads containing input data and output formats.

5.1.8 PyPI

PyPI[1] is the Python Package Index, a repository of software for the python programming language. This repository helps you find and install software developed by the Python community who have decided to share their work. The GOCPI package is distributed from this platform to enable as many as possible the ability to model their own energy systems to inform and question energy policy and investment. Enter command: **pip install GOCPI** in the terminal to install the package using pip package management software.

5.1.9 Code Style

The GOCPI project was developed as the GOCPI package. All development code is organised within this package. The PEP8 style for Python Code was the formatting style for development code [**PEP8**]. All code was formatted with **yapf**, a formatter maintained by Google to format Python files. Standardised formatting is important as makes the code easy to read, helps optimise the code and promotes consistency. Docstrings and commenting were most important in documentation. A docstring is a Python inline comment. Each class and function has an unique docstring, a one sentence description of the function, inputs with data types and types of outputs. The Google style docstring was most appropriate because of it's readability, ease to write and consistency with the Google Style Guide. Additionally, automated documentation generators (**pdoc3, Sphinx** etc.) can parse this format to create documentation. This self-consistent code style facilitated best practise maintenance and enabled reproducibility.

5.1.10 Infrastructure

GOCPI creates scalable energy system optimisation models with complexity size dependent. Computations either took place locally on a 128 GB, four core Apple MacBook Pro or remotely using a cloud service.

5.2 Documentation

The GOCPI project is well documented to keep an accurate record of key design decisions. The commit history described in **??** was the most important form of document. Other explicit documentation methods were applied to supplement this commit history. These

[1] https://pypi.org/

methods, in addition to in-code documentation, include project updates and meeting minutes nested within a project logbook.

5.2.1 Project updates

Project updates were recorded as itemized lists. Each item is a brief description of the work completed during that day, week or month. Items include, but are not limited to, completing GOCPI submodules, researching energy system statistics, building websites or writing sections of this research report. These updates were pivotal to exploring new options, monitoring progress and making decisions to drive forward development. For example, the decision to adopt the OseMOSYS methodology in favour of the TIMES modelling methodology. Project updates were transcribed to the project logbook held in this project's research compendium.

5.2.2 Meeting minutes

Project meetings took place for half an hour once a week. These meetings included discussions on energy markets, modelling methodologies, project progress and key design decisions. The minutes from these meetings accompanies project updates in the project logbook nested within the research compendium.