Computational Materials Science 146 (2018) 220–229

Contents lists available at ScienceDirect

Computational Materials Science

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

Simple data and workflow management with the `signac` framework

Carl S. Adorf[a], Paul M. Dodd[a], Vyas Ramasubramani[a], Sharon C. Glotzer[a,b,c,*]

[a] Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109, United States
[b] Department of Materials Science and Engineering, University of Michigan, Ann Arbor, MI 48109, United States
[c] Biointerfaces Institute, University of Michigan, Ann Arbor, MI 48109, United States

ARTICLE INFO

Article history:
Received 18 October 2017
Received in revised form 14 January 2018
Accepted 15 January 2018

Keywords:
Data management
Database
Data sharing
Provenance
Computational workflow

ABSTRACT

Researchers in the fields of materials science, chemistry, and computational physics are regularly posed with the challenge of managing large and heterogeneous data spaces. The amount of data increases in lockstep with computational efficiency multiplied by the amount of available computational resources, which shifts the bottleneck in the scientific process from data acquisition to data processing and analysis. We present a framework designed to aid in the integration of various specialized data formats, tools and workflows. The `signac` framework provides all basic components required to create a well-defined and thus collectively accessible and searchable data space, simplifying data access and modification through a homogeneous data interface that is largely agnostic to the data source, *i.e.*, computation or experiment. The framework's data model is designed to not require absolute commitment to the presented implementation, simplifying adaption into existing data sets and workflows. This approach not only increases the efficiency with which scientific results can be produced, but also significantly lowers barriers for collaborations requiring shared data access.

1. Introduction

Improved software [1–5] and increased resources available to computational researchers [6,7] have led to significant increases in the quantities of data generated [8]. This makes a highly systematic data management approach crucial to preserving data provenance and ensuring reproducibility. To address this problem, researchers often employ data organization practices such as using human-readable file-naming conventions. Although such solutions address the problem at a superficial level, they suffer from numerous drawbacks with respect to efficiency and flexibility. Here, we introduce `signac`, named after Paul Signac (see Fig. 1), a simple and robust framework for the management of complex and heterogeneous data spaces as well as the efficient implementation of workflows. Data spaces managed with `signac` are immediately searchable and sharable.

The capabilities of `signac` are best illustrated by example. Consider a typical, albeit trivial, research task in which we are given data about the pressure, volume, and temperature of a noble gas and wish to develop a simple theory to explain these data. As a first hypothesis, we might test Boyle's law, $pV = \text{const.}$, by iterating over values of p and storing the corresponding values for V in text files named for those values of p. Upon finding that the data appears to be temperature-dependent, we then could choose to test a more general equation, $pV = NkT$.

We are now faced with a dilemma: how do we efficiently adapt our data space for this extension? We could provide the existing files with new names incorporating temperature, but this could quickly become intractable if we had to further increase the complexity of our equation of state. Alternatively, we might determine that storing data in a (relational) database would be a more flexible solution to accommodate any future schema changes; however, that could be much less efficient for a generally file-based workflow and could introduce a significant bottleneck in downstream data processing and analysis.

The `signac` framework resolves this by abstracting away the details of file-based data storage while simultaneously functioning like a lightweight, semi-structured database. Using `signac`, files are directly stored on the file system *along with the associated metadata* in a well-defined storage layout. The metadata is parsed and indexed on-the-fly whenever we use `signac`'s interface to access and search for data. By using `signac` to manage the data in the above example, the tasks of adding a parameter such as temperature and searching for data associated with a particular p, T pair can both be easily realized with only a few commands.

* Corresponding author at: Department of Chemical Engineering, University of Michigan, Ann Arbor, MI 48109, United States.

E-mail addresses: csadorf@umich.edu (C.S. Adorf), pdodd@umich.edu (P.M. Dodd), vramasub@umich.edu (V. Ramasubramani), sglotzer@umich.edu (S.C. Glotzer).

https://doi.org/10.1016/j.commatsci.2018.01.035
0927-0256/© 2018 Elsevier B.V. All rights reserved.

Fig. 1. The Pointillist style was invented by Paul Signac (1863–1935) and Georges Seurat (1895–1891) and describes paintings in which images are composed from collections of individual dots, each containing a single color. This style serves as a metaphor for `signac`'s data model, in which the data is dependent on both individual data points *and* their position within the larger parameter space. The painting underlying this artistic illustration *Cassis, Cap Lombard* was created by Paul Signac in 1889 and is owned by the Gemeentemuseum Den Haag.

This paper is organized as follows. First, the general design principles of `signac` are presented. We then delve into greater detail about how the core `signac` functionality is implemented in keeping with these principles, followed by a more in-depth comparison to closely related solutions. Finally, the practicality of this system is demonstrated through numerous examples indicating how `signac` can be used to manage a variety of disparate, heterogeneous data sets.

2. Overview

2.1. Design

In the following section we lay out the core design principles behind `signac`, which necessitates making a clear distinction between the `signac` *framework* and the `signac` *application*. The primary focus of this paper is the `signac` application (henceforth simply `signac`), which implements the core data management functions discussed throughout this paper. The `signac` framework is a collection of applications and modules that are built on top of the core `signac` application, such as the `signac-flow` application, which will be introduced in Section 3.3.

At its core, `signac` is a database built directly on top of the file system, leveraging the many advantages of direct file system access while also providing functions to efficiently index and search the data space. As a database system, `signac` makes only one central assumption: that all data may be discretized within a high-dimensional parameter space (see Fig. 2). Once the user provides the parameters and associated data, `signac` is responsible for managing both the storage of data and its association with the parameters through the maintenance of metadata files encoded in the open JavaScript Object Notation (JSON) format. Through this division, `signac` can ensure both data integrity and searchability.

The database functions of `signac` are modeled after those provided by well-tried database management systems (DBMS) such as MongoDB [9] or MySQL [10]. Typically, such DBMS are very efficient when it comes to the execution of complex query and

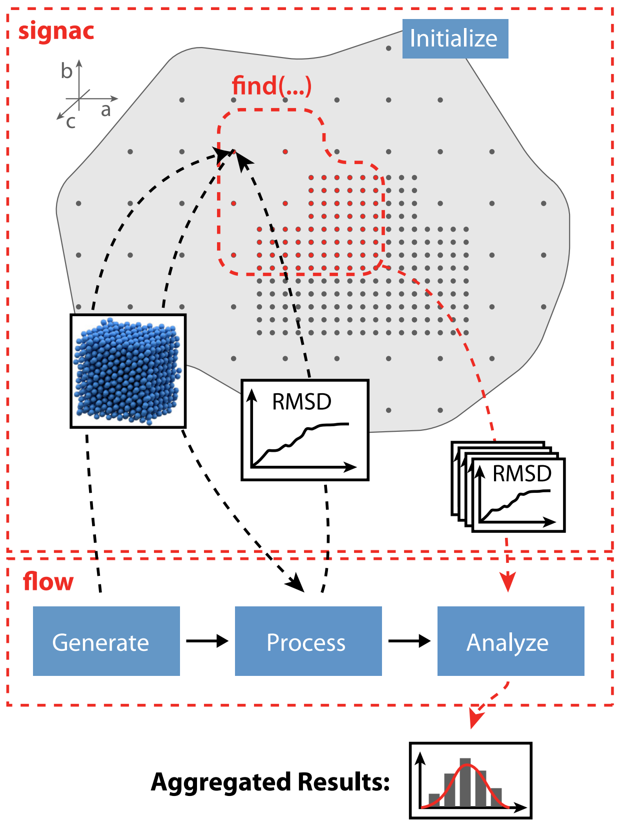

Fig. 2. This conceptual example demonstrates how we manage and operate on a data space using the `signac` and the `signac-flow` applications. We use `signac` to *initialize* a discrete data space (represented by dark grey dots), where each dot represents a discrete data point and may be associated with anything from a single number to a large set of data. The data space is coordinated within a higher-dimensional parameter space (light grey shape), in this case spanned by the three vectors *a*, *b*, and *c*. Manipulations of the data space (addition, modification, or removal of data), can be divided into *operations*, where each operation must be a function of one or more data points. The operations shown in the example deposit and extract data (dashed arrows) and are organized into a specific workflow using `signac-flow`. Specifically, after initialization, we first *generate* particle configurations, then *post-process* these configurations to extract the root-mean squared displacement (RMSD). Finally, we aggregate results *via* the *analysis* of a subset of our data space that we *find* using a `signac` search query. This example shows the clear division of responsibilities between the different applications. The signac application manages and provides access to the data space and allows us to perform complex search queries. The `signac-flow` application assists in the definition and execution of reproducible workflows comprised of individual data space operations.

aggregation operations; however, there are two main issues that render these tools suboptimal for managing the large amounts of (binary) data typically generated by massively parallelized scientific applications within high-performance computing (HPC). First, unless a database is specifically set up to handle peak loads originating from many instances (potentially numbering in the thousands) hitting them in parallel, reading and writing to files distributed on the file system will usually scale more efficiently. Setting up a partitioned or replicated database system to handle higher loads is non-trivial, and this task becomes even more complicated if we care about proper authentication and authorization among different nodes. Secondly, data may need to be serialized for ingestion into the database, which may pose another performance bottleneck, particularly if the data are large binary files.

With `signac`, files are managed directly on the file system and performance is mainly determined by the latency and scalability of the file system. This technique fully exploits the existing file systems on supercomputers, which are commonly designed to process highly parallel, computationally intensive input/output (I/O)