Biomolecular Crystallographic Computing with Jupyter

Blaine H. M. Mooers[‡§¶‖*]

Abstract—The ease of use of Jupyter notebooks has helped biologists enter scientific computing, especially in protein crystallography, where a collaborative community develops extensive libraries, user-friendly GUIs, and Python APIs. The APIs allow users to use the libraries in Jupyter. To further advance this use of Jupyter, we developed a collection of code fragments that use the vast *Computational Crystallography Toolbox* (*cctbx*) library for novel analyses. We made versions of this library for use in JupyterLab and Colab. We also made versions of the snippet library for the text editors VS Code, Vim, and Emacs that support editing live code cells in Jupyter notebooks via the GhostText web browser extension. Readers of this paper may be inspired to adapt this latter capability to their domains of science.

Index Terms—literate programming, reproducible research, scientific rigor, electronic notebooks, JupyterLab, Jupyter notebooks, Colab notebook, OnDemand notebooks, computational structural biology, computational crystallography, biomolecular crystallography, protein crystallography, biomolecular structure, computational molecular biophysics, biomedical research, data visualization, scientific communication, GhostText, text editors, snippet libraries, SciPy software stack, interactive software development

Introduction

Biomolecular crystallography involves the determination of the molecular structure of proteins and nucleic acids and their complexes by using X-rays, neutrons, or electrons. The molecular structure determines the protein's biological function, so the experimentally determined structures provide valuable insights vital for understanding biology and developing new therapies in medicine. The recent *resolution revolution* in cryo-electron microscopy (cryo-EM) [1] and the breakthrough in protein structure prediction with neural networks now provide complementary sources of insights into biomolecular structure [2], [3], [4]. However, the crystallographic approach continues to play a vital role because it still supplies the most precise structures, [5].

About half of the crystal structures of protein molecules are refined with the program *Phenix* [6]. This program has a user-friendly GUI that supports standard analyses [7]. Phenix runs on top of *cctbx* [8]. The Computational Crystallography Toolbox (*cctbx*) provides a transparent API, so most users of *Phenix* are barely aware that it relies on *cctbx*. However, nonstandard analyses are not available in *Phenix* and require accessing the functions in the *cctbx* library (e.g., [9]). The backend *cctbx* was written in C++ in the early 2000s for speed and to provide customized data structures for crystallography. Likewise, the GUI-driven *Olex2* small molecule refinement program uses *cctbx* for many of its crystallographic computations [10].

To ease the use of *cctbx* by general users, the C++ interfaces, classes, and functions of *cctbx* are exposed to Python via the *Boost.Python* Library [11]. Recently, dependency management in *cctbx* was reworked by leveraging the Anaconda infrastructure to ease its installation. In spite of these conveniences, the widespread adoption of Python by field practitioners over the past decade, and the presence of several on-line tutorials about *cctbx*, many structural biologists still find *cctbx* hard to master and adoption has remained low. This difficulty drove several groups to develop software libraries (e.g. *reciprocalspaceship* [12], *GEMMI* [13]) that reinvent some features of *cctbx* while utilizing the more familiar *pandas* DataFrames in place of *cctbx*'s customized data structures. In contrast to these new competitors, *cctbx* has more extensive coverage of advanced crystallographic data analysis tasks and is more thoroughly tested as the result running underneath Phenix for almost two decades. *cctbx* remains the ultimate library for building advanced crystallographic data analyses tools, so the field would benefit if *cctbx* were easier to use.

To foster adoption of *cctbx*, we present a collection of *cctbx* code snippets to be used in Jupyter notebooks [14]. Jupyter provides an excellent platform for exploring the *cctbx* library and prototyping new analysis tools. The Python API of *cctbx* simplifies running *cctbx* in Jupyter via a kernel specific for its conda environment. We formatted our snippet library for several snippet extensions for the Classic Notebook and for Jupyter Lab. To overcome the absence of tab triggers in the Jupyter ecosystem to invoke the insertion of snippets, we also made the snippets available for leading text editors. The user can use the GhostText browser plugin to edit the contents of a Jupyter cell in a full-powered external editor. GhostText enables the user to experience the joy of interactive computing in Jupyter while working from the comfort of their favorite text editor. These multiple modalities of using *cctbx* in Jupyter that we describe below may inspire workers in other domains to build similar snippet libraries for domain-specific software.

Results

We provide a survey of the snippet library that we have customized for several snippet extensions in JupyterLab and Google Colab.

* *Corresponding author: blaine-mooers@ouhsc.edu*
‡ *Department of Biochemistry and Molecular Biology, University of Oklahoma Health Sciences Center, Oklahoma City, OK 97104*
§ *Stephenson Cancer Center, University of Oklahoma Health Sciences Center, Oklahoma City, OK 97104*
¶ *Laboratory of Biomolecular Structure and Function, University of Oklahoma Health Sciences Center, Oklahoma City, OK 97104*
‖ *Biomolecular Structure Core, Oklahoma COBRE in Structural Biology, University of Oklahoma Health Sciences Center, Oklahoma City, OK 97104*

jupyterlabcctbxsnips

We developed the *jupyterlabcctbxsnips* library of code templates for the JupyterLab extension *jupyterlab-snippets* (`https://github.com/QuantStack/jupyterlab-snipp`). Access to the code templates or snippets requires the editing of the Jupyter notebook from inside of JupyterLab, a browser-based IDE for displaying, editing, and running Jupyter notebooks.

JupyterLab supports more comprehensive workflows for academic work than what is possible in the Classic Jupyter Notebook application. For example, it enables the writing or editing of a document in a pane next to the Jupyter notebook. This variant is useful for writing documentation, protocols, tutorials, blog posts, and manuscripts next to the notebook that is being described. The document can be a plain text, html, markdown, LaTeX, or even an org-mode file if one activates the text area with GhostText while running one of several advanced text editors (see the section below about GhostText). The editing of a document next to the related Jupyter notebook supports reproducible research and reduces costly context switching.

We made a variant of the library, *jupyterlabcctbxsnipsplus* (`https://github.com/MooersLab/jupyterlabcctbxsnipsp`) that has a copy of the code in a block comment (Fig. 1). In the commented code, suggested sites for editing are indicated by tab stops that are marked with dollar signs.

```
[ ]: """
     from iotbx import mtz
     mtz_obj = mtz.object(file_name="/Users/blaine/${1:3nd4}.mtz")
     mtz_obj.show_summary()
     """

     from iotbx import mtz
     mtz_obj = mtz.object(file_name="/Users/blaine/3nd4.mtz")
     mtz_obj.show_summary()

     # Description:  Read mtz file into a mtz object and print summary.
     # Source:  NA
```

Fig. 1: *A snippet from the* jupyterlabcctbxsnipsplus *library with duplicate code in a comment block. The dollar sign marks the start of a tab stop. The comment block guides the editing of the active code.*

The figure below (Fig. 2) shows part of the cascading menus for the *jupyerlabcctbxsnipsplus* library after it has been installed successfully. The submenus correspond to the names of subfolders in the *cctbx+* folder in the snippets folder, which was manually created inside of the Jupyter folder in the local library folder (i.e., `~/Library/Jupyter/multimenus_snippets/cctbx+` on macOS).

Each final menu item is linked to a Python snippet file. The selection of a snippet file by clicking on it with the left-mouse button inserts its content into a new cell below the current cell.

In contrast, the *mtzOjbectSummary.py* snippet was selected from the *cctbx* submenu and lacks the comment block. This code was inserted in the current notebook cell (Fig. 3). The code in this cell was be executed by entering **Shift-Enter**.

The *mtzObjectSummary.py* snippet prints a summary of an mtz file. A mtz file is a binary file that contains diffraction data in a highly customized data structure. The data in this mtz file has columns of I(+) and I(-). These are the Bijvoet pairs of diffraction intensities. These pairs are related by symmetry and should have equal intensity values within experimental error. The differences in intensities are a measure of the presence of

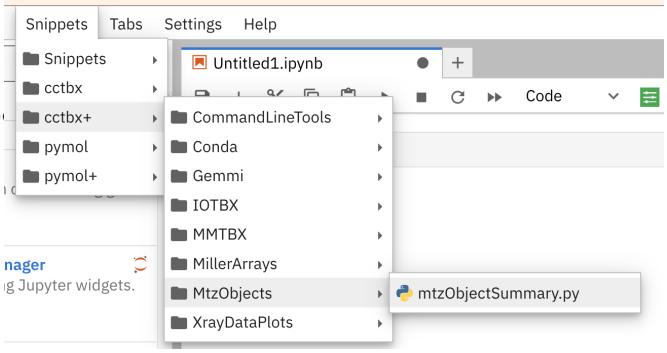

Fig. 2: *The cascading menus for the* jupyterlabcctbxsnipsplus *library for the jupyterlab-snippets version 0.4.1 extension in JupyterLab version 3.5.2.*

```
[11]: from iotbx import mtz
      mtz_obj = mtz.object(file_name="/Users/blaine/3hz7.mtz")
      mtz_obj.show_summary()

Title: phenix.cif_as_mtz
Space group symbol from file: P43212
Space group number from file: 96
Space group from matrices: P 43 21 2 (No. 96)
Point group symbol from file: 422
Number of crystals: 2
Number of Miller indices: 6482
Resolution range: 25.5941 1.99683
History:
Crystal 1:
  Name: HKL_base
  Project: HKL_base
  Id: 0
  Unit cell: (46.046, 46.046, 82.811, 90, 90, 90)
  Number of datasets: 1
  Dataset 1:
    Name: HKL_base
    Id: 0
    Wavelength: 0
    Number of columns: 0
Crystal 2:
  Name: crystal_0
  Project: project
  Id: 2
  Unit cell: (46.046, 46.046, 82.811, 90, 90, 90)
  Number of datasets: 1
  Dataset 1:
    Name: dataset
    Id: 1
    Wavelength: 0.97976
    Number of columns: 11
    label   #valid  %valid    min       max type
    H         6482 100.00%   0.00     22.00 H: index h,k,l
    K         6482 100.00%   0.00     16.00 H: index h,k,l
    L         6482 100.00%   0.00     41.00 H: index h,k,l
    F(+)      6448  99.48%   0.00    476.80 G: F(+) or F(-)
    SIGF(+)   6448  99.48%   0.00     10.19 L: standard deviation
    F(-)      5005  77.21%   0.00    393.03 G: F(+) or F(-)
    SIGF(-)   5005  77.21%   0.00      8.88 L: standard deviation
    I(+)      6448  99.48% -76.20 227334.00 K: I(+) or I(-)
    SIGI(+)   6448  99.48%   4.00   4014.00 M: standard deviation
    I(-)      5005  77.21% -98.90 154474.00 K: I(+) or I(-)
    SIGI(-)   5005  77.21%  10.20   2081.00 M: standard deviation
```

Fig. 3: *The code and output from the* mtzObjectSummary.py *snippet in JupyterLab.*

anomalous scattering. Anomalous scattering can be measured for elements like sulfur and phosphorus that are part of the native protein and nucleic acid structures and heavier elements like metals that are naturally occurring as part of metalloproteins or that were purposefully introduced by soaking crystals or that were incorporated covalently into the protein (e.g., selenomethionine) or nucleic acid (e.g., 5-bromouracil) during its synthesis.

The anomalous differences can be used to determine the positions of the anomalous scattering atoms. Once the positions of the anomalous scatterers are known, it is possible to work out the positions of the lighter atoms in the protein. We use these data to make the I(+) vs I(-) scatter plot below (Fig. 4). The mtz file contains data for SirA-like protein (DSY4693) from *Desultobacterium hafniense*, Northeast Structural Genomics Consortium Target DhR2A. The diffraction data were retrieved

from the Protein Data Bank, a very early open science project that recently celebrated its 50th anniversary [15].

The I(+) vs I(-) plot was made after reading the X-ray data into a *cctbx* Miller array, a data structure designed for handling X-ray data in *cctbx*. The I(+) and I(-) were eventually read into separate lists. We plot the two lists against each other in a scatter plot using *matplotlib* [16]. There is no scatter from the $x = y$ line in this plot if there is no anomalous signal. The larger the anomalous signal, the greater the scatter. The departure from this line is expected to be greater for intensities of large magnitude.

```
[12]: %matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.ticker as ticker
from matplotlib.ticker import MultipleLocator #, FormatStrFormatter
from matplotlib.ticker import FuncFormatter
from iotbx.reflection_file_reader import any_reflection_file

# >>> change the mtz file name
hkl_file = any_reflection_file('3hz7.mtz')
miller_arrays = hkl_file.as_miller_arrays(merge_equivalents=False)
Iobs = miller_arrays[1]
i_plus, i_minus = Iobs.hemispheres_acentrics()
ipd = i_plus.data()
ip=list(ipd)
imd = i_minus.data()
im = list(imd)
len(im)

comma_fmt = FuncFormatter(lambda x, p: format(int(x), ','))

mpl.rcParams['savefig.dpi'] = 600
mpl.rcParams['figure.dpi'] = 600

# Set to width of a one column on a two-column page.
# May want to adjust settings for a slide.
fig, ax = plt.subplots(figsize=[3.25, 3.25])
ax.scatter(ip,im,c='k',alpha=0.3,s=5.5)
ax.set_xlabel(r'I(+)',fontsize=12)
ax.set_ylabel(r'I(-)',fontsize=12)
ax.xaxis.set_major_locator(MultipleLocator(50000.))
ax.yaxis.set_major_locator(MultipleLocator(50000.))
ax.get_xaxis().set_major_formatter(comma_fmt)
ax.get_yaxis().set_major_formatter(comma_fmt)

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
ax.grid(False)

# >>> change name of the figure file
plt.savefig('3hz7IpIm.pdf',bbox_inches='tight')
```

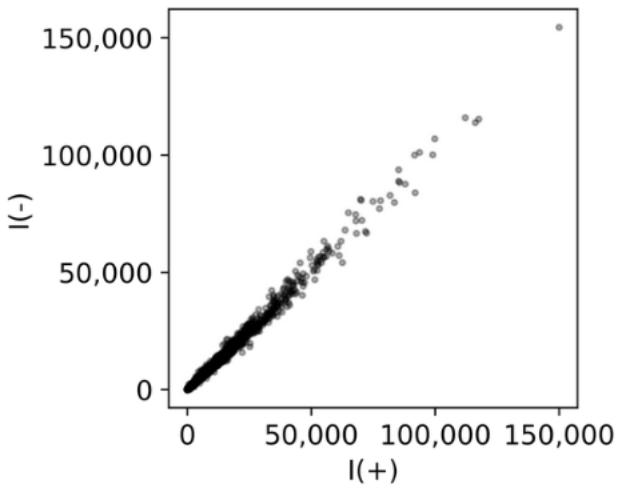

Fig. 4: *The code snippet to generate a Ip versus Im plot and the corresponding plot generated by the code.*

Plots of this nature are useful for detecting very weak anomalous signals from native anomalous scatters like sulfur and phosphorus. The collection of the anomalous signal from native scatters enables structure determination without having to spend the extra time and money to introduce heavier atoms that are not native to the protein. The measurement of the very weak signal from native anomalous scatterers is still at the edge of what is technically possible. It has rarely been achieved with in-house instruments. Success generally requires the faster multi-million dollar detectors at beamlines, tunable wavelengths of synchrotron radiation available at one of 30+ laboratories around the world, and cryogenic temperatures (-173 C) maintained by a cryostream of nitrogen gas that slows radiation damage long enough to collect complete datasets.

However, recently, several groups have completed successful native phasing experiments at room temperature by collecting data from large numbers of crystals and merging the data [17], [18]. The advantages of room temperature data collection include avoidance of conformational changes in the protein induced by supercooling the crystal. The room temperature data were collected from each crystal briefly before radiation damage degraded the diffraction too much. This is a remarkable achievement because the merging of diffraction data from many crystals in various orientations enhances the experimental error; this error can mask the weak anomalous signal that is being sought.

The plot (Fig. 4) was adapted from an example in the *reciprocalspaceship* project from the Hekstra Lab [12]. This new project takes a more Pythonic approach than *cctbx* by utilizing many of the packages in the SciPy stack that did not exist when *cctbx* was initiated. For example, it uses the *pandas* package to manage diffraction data whereas *cctbx* uses a custom C++ data structure for diffraction data that predates *pandas* by almost a decade. The utilization of *pandas* enables easier integration with the other components of the SciPy software stack including machine learning packages.

The *cctbx* is most easily installed into its own environment by using Anaconda with the command `conda create -n my_env -c conda-forge cctbx-base python=3.11`.

The atomic coordinates of the biomolecular structures are the other major type of data that are intimately associated with diffraction data. The fixed file format of Protein Data Bank coordinate files with the file extension of *pdb* originated in the 1970s with the birth of the Protein Data Bank, but very large biological macromolecules have been determined over the past two decades that exceeded the limits on the number of atoms permitted in one file. To address this and other shortcomings of the PDB file format, the PDBx/mmCIF (Protein Data Bank Exchange macromolecular Crystallographic Information Framework) file format recently became the new data standard [19]. The *cctbx* has been adapted to read mmCIF files.

taggedcctbxsnips

The Elyra-snippets extension for Jupyter Lab supports the use of tagged snippets (`https://elyra.readthedocs.io/en/latest/user_guide/co`
Each snippet is in a separate JavaScript file with the *json* file extension 5.

Each snippet file has a set of metadata. These data include a list of tags. The tags are used to find the snippet while editing a Jupyter notebook in JupyterLab. We made a version of the cctbxsnips library for the Elyra-snippets extension (`https://github.com/MooersLab/taggedcctbxsnips`).

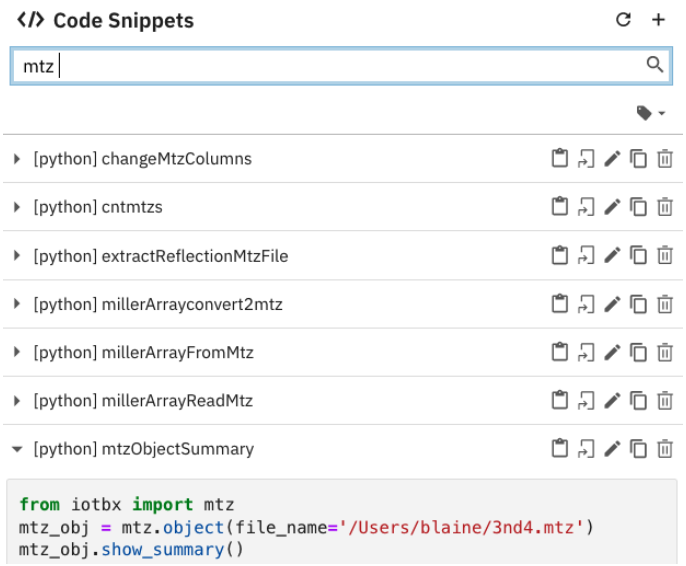

Fig. 5: *Snapshot of a list of snippets in JupyterLab supported by the Elyra-snippet extension. The 80* cctbx *snippets have been narrowed to seven snippets by entering the* `mtz` *tag. Additional tags can be entered to further narrow the list of candidate snippets.*

To add a new snippet, click on the + in the upper right of the Code Snippets icon (Fig. 6). This will open new GUI (see below) for creating a snippet. The value of *Name* should be one word or a compound word. The value of *Description* describes in one or more sentences what the snippet does. The values of the *Tags* field are used to narrow the listing of snippets in the menu. The value of the *Source* is the programming language; the value is Python in this example. The *Code* can be entered by selecting code in a notebook cell or copying and pasting from a script file.

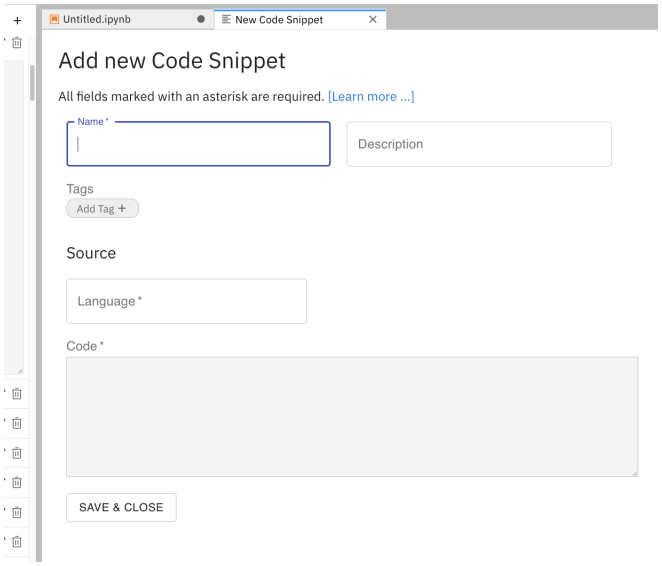

Fig. 6: *The GUI to create a new snippet via the Elyra-snippet extension for JupyterLab.*

colabcctbxsnips

The Google Colab notebook enables the running of software on Google's servers in a computational notebook that resembles the Jupyter notebook. Colab notebooks are useful for workshop settings where there is no time for installing software on a heterogeneous mix of operating systems when the attendees are following the presentation by using their own computers.

Colab notebooks do no support external extensions, but they have built-in support for snippets. A particular snippet library is stored in a dedicated Google Colab notebook rather than in individual files. The notebook of snippets is stored on the user's Google Drive account. While the software installed in a Colab session is lost upon logging out, the snippets remain available on the next login.

After the snippet notebook is installed, the user opens a new notebook to use the snippets. From that new notebook, the list of snippets will be exposed by clicking on the **<>** icon in the left margin of the notebook. Click on the `Insert` button in the upper righthand corner of the snippet to copy the snippet to the current code cell in the notebook.

We developed the *colabcctbxsnips* library and stored it in a Colab Notebook (`https://github.com/MooersLab/colabcctbxsnips`). Two snippets have the code for installing *mamba* and then *cctbx* (Fig. 7). These code snippets have to be run before *cctbx* can be accessed. The two code fragments require less than two minutes to install the software.

Fig. 7: *Snippets from the* cctbx *library for installing* mamba *and then* cctbx *on Google Colab.*

The Colab snippet system also lacks support for tab triggers and tab stops. We address this problem by supplying a copy of the snippet with the sites of the tab stops marked up like a yasnippet snippet. Unlike the case of the *jupyterlabcctbxsnipsplus* library, the marked up copy of the code snippet is displayed only in the preview of the snippet and is not inserted into the code cell along with the active code (Fig. 8).

Snippets for OnDemand Notebooks at HPCs

We have also worked out how to deploy this snippet library in OnDemand notebooks at High-Performance Computing centers. These notebooks resemble Colab notebooks in that JupyterLab extensions cannot be installed. However, they do not have any alternate support for accessing snippets from menus in the GUI. Instead, we had to create IPython magics for each snippet that load

millerArrays mtz2array Insert

- Read a mtz file into a miller array.
- The $ below marks a site for editing.

```
from iotbx.reflection_file_reader import any_reflection_file
hkl_file = any_reflection_file("${1:3hz7.mtz}")
miller_arrays = hkl_file.as_miller_arrays(merge_equivalents=False)
```

```
from iotbx.reflection_file_reader import any_reflection_file
hkl_file = any_reflection_file("3hz7.mtz")
miller_arrays = hkl_file.as_miller_arrays(merge_equivalents=False)
```

View source notebook

Fig. 8: *Preview of a Colab code snippet. The preview contains two copies of the code. The bottom copy of the code will be inserted into the current code cell. The top copy of the code serves as a guide to sites to be edited. The dollar sign marks the start of a tab stop where the enclosed placeholder value may need to be changed.*

the snippet's code into the code cell. This system would also work on Colab and may be preferred by expert users because the snippet names used to invoke the Ipython magic are under autocompletion. We offer a variant library that inserts a commented out copy of the code that has been annotated with the sites that are to be edited by the user.

cctbxsnips *for leading text editors*

To support the use of the *cctbx* code snippets in text editors, we made versions of the library for Emacs, Vim, Neovim, Visual Studio Code, Atom, and Sublime Text3. We selected these text editors because they are the most advanced and most popular with software developers and because they are supported by the GhostText project described below .

For Emacs, we developed a library for use with the yasnippets package (`https://github.com/MooersLab/cctbxsnips-Emacs`). Emacs supports repel-driven software development, which resembles the interactive software development experience in Jupyter notebooks. Emacs also supports the use of literate programming in several kinds of documents, including the very popular org-mode document [20]. Code blocks in org documents can be given a **jupyter** option with a Jupyter kernel name that enables running a specific Jupyter kernel including one mapped to a conda environment that has the *cctbx* package installed. A similar example using the molecular graphics package PyMOL is demonstrated in this short video (`https://www.youtube.com/watch?v=ZTocGPS-Uqk&t=2s`).

Using GhostText to edit Jupyter cells from a favorite text editor

By adding the GhostText extension (`https://ghosttext.fregante.com/`) to the web browser and a server to one of several leading text editors, it is possible to send the text from the browser through a WebSocket to a server in the text editor. Thus, it is possible to edit the contents of a computational notebook cell from inside a text editor. Changes made in the text editor instantly appear in the notebook and vice versa. By applying the power of a text editor to computational notebooks, experienced developers can continue to use familiar editing commands and tools in their preferred text editor.

GhostText is a Javascript program developed by Federico Brigante, a prolific JavaScript developer. Versions of the extension are available for the Google Chrome, Firefox, Edge, Opera, and Safari. The extension for the Google Chrome browser works in the Brave browser, and the extension for Firefox works in the Waterfox and Icecat browsers. GhostText was developed initially for Sublime Text 3, so Sublime Text 3 can serve as a positive control even if another editor in the list is the favored editor. (Sublime Text 3 is available for most platforms for a free trial period of infinite length.)

The snippet extensions for the Classic Jupyter Notebook and JupyterLab lack support for tab triggers to insert snippets as you type and tab stops inside the snippet to advance to sites in the snippet that may need to be edited. These two features are standard in the software that supports the use of snippet libraries in most text editors.

As a quick reminder, tab triggers in text editors insert chunks of computer code after the user enters the tab trigger name and hits the TAB key (Fig. 9). The tab trigger name can be as short as several letters. Many text editors and IDEs have pop-up menus that aid the selection of the correct tab trigger. Tab stops are sites within the code snippet where the cursor advances to after entering TAB again. These sites often have placeholder values that can be either edited or accepted by entering TAB again. Sites with identical placeholder values can be mirrored so that a change in value at one site is propagated to the other tab stops with the same placeholder value. The absence of tab stops can increase the number of bugs introduced by the developer by overlooking parameter values in the code snippet that need to be changed to adapt the snippet to the current program.

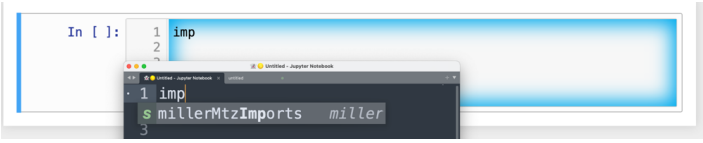

Fig. 9: *Example of a tab trigger being entered in Sublime Text 3 editor and appearing in a Jupyter Notebook cell. A pop-up menu lists the available snippets. The list was narrowed to one snippet by the entry of three letters.*

The text editor also needs to be extended with a server that enables two-way communication with the web page via a WebSocket. Edits made on the browser side of the WebSocket are immediately sent to an open page in the Text Editor and vice versa; however, the text editor's snippets and other editing tools only work in the text editor. The connection can be closed from either side of the WebSocket. It is closed on the web browser side via an option in GhostTest's pulldown menu, and it closed on the text editor side by closing the active buffer.

Here, we describe the setup for Emacs as an example of configuring a text editor to use GhostText. The server for *Emacs* is provided by the *atomic-chrome* package that is available in both the Milkypostman's Emacs Lisp Package Archive (MELPA) and on GitHub (`https://github.com/alpha22jp/atomic-chrome`). The configuration for *atomic-chrome* in the Emacs initialization file (e.g., init.el) is listed below (Fig. 10). The third line in Code listing 1 sets the default Emacs mode (equivalent to a programming language scope): We set it to Python for Jupyter code cells. Atomic-chrome uses text-mode by default. You can change the default mode to other programming languages that you may use in Jupyter, like Julia or R. The last three lines specify the Emacs mode to be used when text is imported from the text

areas on github.com, Overleaf.com, and 750words.com. Similar configuration options are available in the other text editors, or you manually change the language scope for the window with the text imported from Jupyter.

```
(use-package atomic-chrome)
(atomic-chrome-start-server)
(setq atomic-chrome-default-major-mode 'python-mode)
(setq atomic-chrome-extension-type-list '(ghost-text))
(setq atomic-chrome-server-ghost-text-port 4001)
(setq atomic-chrome-url-major-mode-alist
      '(("github\\.com" . gfm-mode)
        ("overleaf.com" . latex-mode)
        ("750words.com" . latex-mode)))
```

Fig. 10: *Emacs lisp code to configure the atomic-chrome package for Emacs. This configuration opens Jupyter notebooks in the Python major mode and the 750words.com webpage in the LaTeX major mode.*

GhostText provides keyboard shortcuts to improve productivity. These shortcuts keep the developer's hands on the keyboard and avoid breaks in context by moving the hand to the mouse. The shortcut by operating system is as follows: macOS, command-shift-K; Linux, control-shift-H; and Windows, control-shift-K.

To support the use of *GhostText* to edit electronic notebooks containing code from the *cctbx* library, we have made variants of a collection of *cctbx* snippets for *Visual Studio Code*, *Atom*, *Sublime Text 3*, *Vim*, *NeoVim*, and *Emacs*. For *Vim* and *NeoVim*, the snippets are available for the *UltiSnips*, *Snipmate*, and *neosnippets* plugins. The snippets are available for download on GitHub (`https://github.com/MooersLab`). From our experience, Sublime Text 3 has the easiest setup while Emacs provides the highest degree of customization. The *cctbx* snippet library was previously only available for use in Jupyter notebooks via extensions for the Classic Jupyter Notebook application or Jupyter Lab.

Note that the snippet library cannot be used with the program *nteract* (`https://nteract.io/`). The *nteract* is an easy-to-install and use desktop application for editing and running Jupyter notebooks offline. The ease of installation makes the *nteract* application popular with new users of Jupyter notebooks. Obviously *nteract* is not browser-based, so it cannot work with *GhostText*. *nteract* has yet to be extended to support the use of code snippet libraries, but *nteract* allows the switching of jupyter kernels between code cells.

While the focus of this report is on Jupyter and Colab notebooks, the *cctbxsnips* snippet library can be used to aid the development of Python scripts in plain text files, which have the advantage of easier version control. The snippets can also be used in other kinds of literate programming documents that operate off-line like org-mode files in Emacs and the *Quarto* (`http://quarto.org`) markdown representation of Jupyter notebooks. *Quarto* is available for several leading text editors. In the later case, you may have to extend the scope of the editing session in the editor to include Python source code.

Discussion

What is new

We report a set of code template libraries for doing biomolecular crystallographic computing in Jupyter. These template libraries only need to be installed once because they persist between logins.

We also include support for Colab notebooks where the snippets also persist between logins but other installed software is lost upon logging out of a session. The templates include the code for installing the software required for crystallographic computing. The installation templates automate as many as seven installation steps. Once the user runs the installation code at the top of a given Colab notebook, the user only needs to rerun these blocks of code upon logging into Colab to be able to reinstall the software during later sessions. The user can also modify the installation templates to install the crystallographic software on their local machine and then run the notebook in Jupyter Classic and JupyterLab. The template libraries presented here lower an important barrier to the use of Colab by those interested in crystallographic computing on the cloud.

We also report the use of GhostText to edit notebook code cells in Jupyter notebooks and text documents in JupyterLab. This capability enables a user to use an external text editor to edit code. The user can thereby take advantage of the support for tab triggers and tab stops in the external editor. This support can ensure faster and more accurate writing and editing of new code.

Relation to other work with snippet libraries

This snippet library is among the first that is domain specific. Most snippet libraries are for programming languages or for hypertext languages like HTML, markdown, and LaTeX. The average snippet in these libraries also tends to be quite short, and the sizes of the libraries tend to be small. The audience for these general purpose libraries are the millions of professional programmers and web page developers. We reasoned that domain-specific snippet libraries with long code fragments are a great coding tool that should be brought to the aid of the tens of thousands of workers in biological crystallography.

The other area where domain-specific snippets have been provided is in molecular graphics. A pioneering scripting wizard provided templates for use in the early molecular graphics program RasMol [21]. In addition, the conscript program provided a converter from *RasMol* to *PyMOL* [22]. We also provided snippets for *PyMOL*, which has about 100,000 users, for use in text editors [23] and Jupyter notebooks [24]. The former supports tab triggers and tab stops; the latter does not.

Opportunities for interoperability

The code template libraries can encourage synergistic interoperability between software packages. That is, the development of notebooks that use two or more software packages and even two or more programming languages. More general and well-known examples of interoperability include the Cython module in Python that enables the running of C++ code inside Python [25], the *reticulate* library that enables the running of Python code in R [26], and the *PyCall* package in Julia that enables the running of the Python packages in Julia (`https://github.com/JuliaPy/PyCall.jl`). The latter package is widely used to run *matplotlib* in Julia. Interoperability already occurs in computational crystallography between *CCP4* [27], *clipper* [28], *GEMMI* [13], *reciprocalspaceship* [12],

Careless [29], and *cctbx* and to a limited extent between *cctbx* and *PyMOL*. The snippet libraries reported here can promote taking advantage of this interoperability in Jupyter and Colab notebooks. We hope that our effort will help raise awareness of interoperability issues among the community.

Snippets in the age of AI-assisted autocompletion

Snippet libraries of domain specific software may not be as redundant as they first appear in the age of chatbots. The code fragments of domain-specific libraries have a limited presence on GitHub, so they may be underrepresented in large language models. In addition, chatbots are designed to return text rather than code. However, *copilot* and *tabnine* were designed for code completion and are good at autosuggesting code fragments. Via GhostText, it is possible to run *copilot* or *tabnine* in a text editor while editing Jupyter notebook cells.

Conclusions

Our explorations suggest that code snippets for domain-specific software libraries have several roles to play in supporting the use of such libraries. First, the snippets illustrate possible uses of the library, thereby, playing educational and inspirational roles. Second, the snippets can speed up the assembly of scripts while reducing the time spent on debugging, thereby, playing a productivity enhancement role. We hope that the *cctbxnsips* library will inspire the creation of similar libraries in other domains.

Acknowledgments

This work was supported in part by the following grants: Oklahoma Center for the Advancement of Science and Technology HR20-002, National Institutes of Health grants R01 CA242845, P30 CA225520, and P30 AG050911-07S1. In addition, we thank the Biomolecular Structure Core of the NIH supported Oklahoma COBRE in Structural Biology (PI: Ann West, P20 GM103640 and P30 GM145423).

References

[1] W. Kühlbrandt, "The resolution revolution," *Science*, vol. 343, no. 6178, pp. 1443–1444, 2014, https://doi.org/10.1126/science.1251652.

[2] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021, https://doi.org/10.1038/s41586-021-03819-2.

[3] M. Mirdita, K. Schütze, Y. Moriwaki, L. Heo, S. Ovchinnikov, and M. Steinegger, "Colabfold: making protein folding accessible to all," *Nature Methods*, vol. 19, pp. 1–4, 2022, https://doi.org/10.1038/s41592-022-01488-1.

[4] R. Chowdhury, N. Bouatta, S. Biswas, C. Floristean, A. Kharkar, K. Roy, C. Rochereau, G. Ahdritz, J. Zhang, G. M. Church *et al.*, "Single-sequence protein structure prediction using a language model and deep learning," *Nature Biotechnology*, vol. 40, no. 11, pp. 1617–1623, 2022.

[5] A. Foerster and C. Schulze-Briese, "A shared vision for macromolecular crystallography over the next five years," *Structural Dynamics*, vol. 6, no. 6, p. 064302, 2019, https://doi.org/10.1063/1.5131017.

[6] D. Liebschner, P. V. Afonine, M. L. Baker, G. Bunkóczi, V. B. Chen, T. I. Croll, B. Hintze, L.-W. Hung, S. Jain, A. J. McCoy *et al.*, "Macromolecular structure determination using x-rays, neutrons and electrons: recent developments in phenix," *Acta Crystallographica Section D: Structural Biology*, vol. 75, no. 10, pp. 861–877, 2019, https://doi.org/10.1107/S2059798319011471.

[7] N. Echols, R. W. Grosse-Kunstleve, P. V. Afonine, G. Bunkóczi, V. B. Chen, J. J. Headd, A. J. McCoy, N. W. Moriarty, R. J. Read, D. C. Richardson *et al.*, "Graphical tools for macromolecular crystallography in phenix," *Journal of Applied Crystallography*, vol. 45, no. 3, pp. 581–586, 2012, https://doi.org/10.1107/S0021889812017293.

[8] R. W. Grosse-Kunstleve, N. K. Sauter, N. W. Moriatry, and P. D. Adams, "The computational crystallography toolbox: crystallographic algorithms in a reusable software framework," *Journal Application Crystallography*, vol. 35, no. 1, pp. 126–136, 2002, https://doi.org/10.1107/S0021889801017824.

[9] E. De Zitter, N. Coquelle, P. Oeser, T. R. Barends, and J.-P. Colletier, "Xtrapol8 enables automatic elucidation of low-occupancy intermediate-states in crystallographic studies," *Communications Biology*, vol. 5, no. 1, p. 640, 2022, https://doi.org/10.1038/s42003-022-03575-7.

[10] L. J. Bourhis, O. V. Dolomanov, R. J. Gildea, J. A. Howard, and H. Puschmann, "The anatomy of a comprehensive constrained, restrained refinement program for the modern computing environment–olex2 dissected," *Acta Crystallographica Section A: Foundations and Advances*, vol. 71, no. 1, pp. 59–75, 2015, https://doi.org/10.1107/S2053273314022207.

[11] D. Abrahams and R. W. Grosse-Kunstleve, "Building hybrid systems with boost.python," *C/C++ Users Journal*, vol. 21, no. 7, 2003. [Online]. Available: https://www.osti.gov/biblio/815409

[12] J. B. Greisman, K. M. Dalton, and D. R. Hekstra, "Reciprocalspaceship: A python library for crystallographic data analysis," *Journal of Applied Crystallography*, vol. 54, no. 5, 2021, https://doi.org/10.1101/2021.02.03.429617.

[13] M. Wojdyr, "Gemmi: A library for structural biology," *Journal of Open Source Software*, vol. 7, no. 73, p. 4200, 2022, https://doi.org/10.21105/joss.04200.

[14] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. development team, "Jupyter notebooks - a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Scmidt, Eds. IOS Press, pp. 87–90, https://doi.org/10.3233/978-1-61499-649-1-87.

[15] wwPDB consortium, "Protein Data Bank: the single global archive for 3D macromolecular structure data," *Nucleic Acids Research*, vol. 47, no. D1, pp. D520–D528, 10 2018, https://doi.org/10.1093/nar/gky949.

[16] J. D. Hunter, "Matplotlib: A 2d graphics environment," vol. 9, no. 3, pp. 90–95, https://doi.org/10.1109/MCSE.2007.55.

[17] F. Yabukarski, T. Doukov, D. A. Mokhtari, S. Du, and D. Herschlag, "Evaluating the impact of x-ray damage on conformational heterogeneity in room-temperature (277 k) and cryo-cooled protein crystals," *Acta Crystallographica Section D: Structural Biology*, vol. 78, no. 8, 2022, https://doi.org/10.1107/S2059798322005939.

[18] J. B. Greisman, K. M. Dalton, C. J. Sheehan, M. A. Klureza, I. Kurinov, and D. R. Hekstra, "Native sad phasing at room temperature," *Acta Crystallographica Section D: Structural Biology*, vol. 78, no. 8, pp. 986–996, 2022, https://doi.org/10.1107/s2059798322006799.

[19] J. D. Westbrook, J. Y. Young, C. Shao, Z. Feng, V. Guranovic, C. L. Lawson, B. Vallat, P. D. Adams, J. M. Berrisford, G. Bricogne *et al.*, "Pdbx/mmcif ecosystem: foundational semantic tools for structural biology," *Journal of Molecular Biology*, vol. 434, no. 11, p. 167599, 2022, https://doi.org/10.1016/j.jmb.2022.167599.

[20] E. Schulte, D. Davison, T. Dye, C. Dominik *et al.*, "A multi-language computing environment for literate programming and reproducible research," *Journal of Statistical Software*, vol. 46, no. 3, pp. 1–24, 1 2012, https://doi.org/10.18637/jss.v046.i03. [Online]. Available: http://www.jstatsoft.org/v46/i03

[21] R. M. Horton, "Scripting wizards for chime and rasmol," *Biotechniques*, vol. 26, no. 5, pp. 874–6, 1999, https://doi.org/10.2144/99265ir01.

[22] S. E. Mottarella, M. Rosa, A. Bangura, H. J. Bernstein, and P. A. Craig, "Conscript: Rasmol to pymol script converter," *Biochem Mol Biol Educ*, vol. 38, no. 6, pp. 419–22, 2010, https://doi.org/10.1002/bmb.20450.

[23] B. H. Mooers and M. E. Brown, "Templates for writing pymol scripts," *Protein Science*, vol. 30, no. 1, pp. 262–269, 2021, https://doi.org/10.1002/pro.3997.

[24] B. H. Mooers, "A pymol snippet library for jupyter to boost researcher productivity," *Computing in Science and Engineering*, vol. 23, no. 2, pp. 47–53, 2021, https://doi.org/10.1109/MCSE.2021.3059536.

[25] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2011, https://doi.org/10.1109/mcse.2010.118.

[26] K. Ushey, J. Allaire, and Y. Tang, *reticulate: Interface to 'Python'*, 2023, r package version 1.28. [Online]. Available: https://CRAN.R-project.org/package=reticulate

[27] J. Agirre, M. Atanasova, H. Bagdonas, C. Ballard, A. Baslé, J. Beilsten-Edmands, R. Borges, D. Brown, J. Burgos-Mármol, J. Berrisford *et al.*, "The ccp4 suite: integrative software for macromolecular crystallogra-

phy," *Acta Crystallographica Section D: Structural Biology*, vol. 79, no. 6, pp. 449–461, 2023, https://doi.org/10.1107/S2059798323003595.

[28] S. McNicholas, T. Croll, T. Burnley, C. M. Palmer, S. W. Hoh, H. T. Jenkins, E. Dodson, K. Cowtan, and J. Agirre, "Automating tasks in protein structure determination with the clipper python module," *Protein Science*, vol. 27, no. 1, pp. 207–216, 2018, https://doi.org/10.1002/pro.3299.

[29] K. M. Dalton, J. B. Greisman, and D. R. Hekstra, "A unifying bayesian framework for merging x-ray diffraction data," *Nature Communications*, vol. 13, no. 1, p. 7764, 2022, https://doi.org/10.1038/s41467-022-35280-8.