Class and Home Problems (CHP) present scenarios that enhance the teaching of chemical engineering at the undergraduate or graduate level. Submissions must have clear learning objectives. CHP papers present new applications or adaptations that facilitate learning in specific ChE courses. Submit CHP papers through journals.flvc.org/cee, include CHP in the title, and specify CHP as the article type.

PROGRAMMATIC COMPILATION OF CHEMICAL DATA AND LITERATURE FROM PUBCHEM® USING MATLAB®

Vincent F. Scalfani, Serena C. Ralph, Ali Al Alshaikh, and Jason E. Bara
The University of Alabama • Tuscaloosa, AL 35487.

INTRODUCTION

Chemical engineering and chemistry are among the most information-intensive disciplines; chemical problem solving and discovery require access to numerous data classes such as chemical structures, reactions, characterization data, and property data. As a result, many extensive secondary literature and data compilations are produced that organize chemical information and facilitate discovery (e.g. CAS SciFinder®).[1, 2] The demand for data, specifically machine-readable chemical data, is rapidly increasing as new informatics and machine-learning techniques are applied to discover knowledge in drug discovery, chemical and materials synthesis, and catalysis research.[3–7]

Machine-readable data processing, real-time data acquisition, modeling, calculation, and analysis techniques are established in chemical engineering education using MATLAB®,[8–12] Mathematica®,[13] and Python®.[14, 15] However, literature searches and property dataset compilations are still largely a manual process; that is, bibliographic references and datasets in chemical engineering are typically discovered and prepared outside of any programmatic data processing workflow. This is unfortunate, as numerous databases such as PubChem®[16] and Engineering Village™[17] allow programmatic access to machine-readable chemical data and/or literature metadata. As programmatic information access continues to grow and evolve, this presents a unique opportunity for chemical engineering to advance reproducibility by incorporating literature searches and data compilations directly into existing programmatic analysis workflows.[18]

Over the past several years, the use and awareness of computational notebooks in research have grown tremendously. Computational notebooks are interactive digital laboratory notebook environments that allow users to mix code, data, results, text, and images all in one professionally formatted and easily shareable document.[19, 20] Examples include Mathematica® notebooks,[21] Jupyter® notebooks,[22] and MATLAB live scripts.[23] Even text markup languages can be configured and used like computational notebooks.[18, 24] Computational notebook methods can be ideal workflow tools for programmatic reproducible literature searching and information sharing. For example, Kitchin has developed a software library, org-ref, that can be used within a markup language computational notebook to programmatically search and cite references from databases like Web of Science™ and PubMed®.[25] Moreover, Rose and Kitchin have recently released a Python software package, pybliometrics, that allows programmatic and reproducible access to information in the Scopus® database.[26] Python code is easily incorporated into Jupyter computational notebooks and provides a convenient way to document and share reproducible programmatic literature searches and analyses.[27] Another recent example of programmatic literature searching uses Jupyter notebooks as a platform for teaching computer-aided drug design. Two of the notebooks allow programmatic retrieval

of chemical data from the ChEMBL database and the Protein Data Bank.[28] Such programmatic searching of information resources and subsequent incorporation or combination with scientific manuscripts greatly enhances the reproducibility and transparency of the research.[29]

In this paper we extend these ideas of programmatic literature and data compilations by demonstrating how to use MATLAB and MATLAB live scripts to compile chemical data and literature from PubChem. PubChem contains millions of molecules, associated literature, and bioactivity data.[16, 30] Users can compile custom PubChem datasets by saving search results from the traditional web interface or by bulk downloading data from the PubChem FTP file site.[31] PubChem also offers programmatic access to data, for example, through their Power User Gateway application programming interface (PUG-REST).[32, 33] Some examples of how to interact with PubChem programmatically are available as Perl scripts[33] and within programming packages such as PubChemPy[34] for Python and webchem for R.[35] We find the MATLAB platform ideal for brand new programmers, students, and information professionals since MATLAB has simple, compact syntax, excellent documentation, and requires minimal setup. In addition, many chemical engineering students and faculty are already familiar with MATLAB, so it is a good place to start. The methods presented herein are useful in chemistry and chemical engineering education, specifically to enhance student data/literature searching and interaction with machine-readable chemical data.

INTERACTING WITH PUBCHEM PROGRAMMATICALLY

In what follows we provide a step-by-step approach for utilizing MATLAB with PubChem, along with the relevent MATLAB code. Figure 1 orients the reader on the steps in the process.

STEP 1: Identify Chemical Search Query

Before beginning to interact with PubChem programmatically, it is useful to briefly review popular methods of encoding molecular structures and patterns as line notation text strings. Text string encoded molecular structures provide a convenient way to construct programmatic database queries in PubChem and other chemical databases. The two most widely used and supported chemical line notations are the Simplified Molecular Input Line-Entry System (SMILES)[36, 37] and the IUPAC International Chemical Identifier (InChI).[38] SMILES are machine-processable molecule structure representations with a defined, often human-friendly, character syntax for encoding features of molecules. There are relatively few rules to understand the bulk of the SMILES syntax to encode

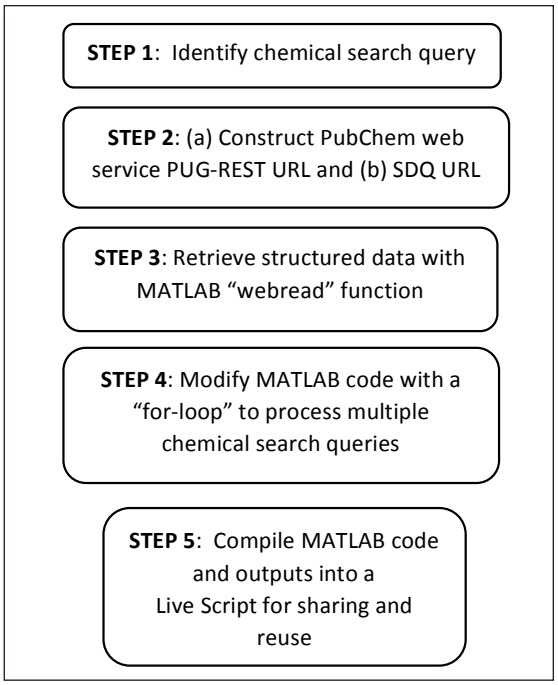

Figure 1. *Steps for interacting with PubChem programmatically.*

atoms, bonds, branching, and rings. Atoms are represented by their atomic symbols, and hydrogen atoms are typically omitted. Bonds are represented with symbols -, =, #, and : for single, double, triple, and aromatic, respectively (single and aromatic bond symbols often omitted). Branching is represented with parentheses, and ring closures use numeric digits to specify connection points. Aromatic molecules can be represented with aromatic atoms denoted as lowercase letters or as Kekule form with upper case letters containing alternating double bonds. Additional syntax exists to represent disconnections (e.g. salts), charges, stereochemistry, and isotopes (Figure 2).[36]

InChI is a machine-friendly chemical structure identifier. The InChI software algorithm normalizes chemical structures to produce a unique line notation with layers. In a standard InChI (standardized options), these layers include characters separated by a slash symbol that describe the molecular formula, connectivity, charge, stereochemistry, and isotope layer (Figure 3).[38] There is also a hashed shorter version of the InChI called an InChIKey, which is useful for web searching. InChIs are incredibly powerful for chemical data linking and organization, as they can uniquely describe molecules.[40]

Finally, it is often useful to be able to describe a molecular pattern within a compact line notation string for programmatic substructure searching (i.e. find an input pattern as part of a molecule). The most common line notation used for this purpose is the SMILES arbitrary target specification

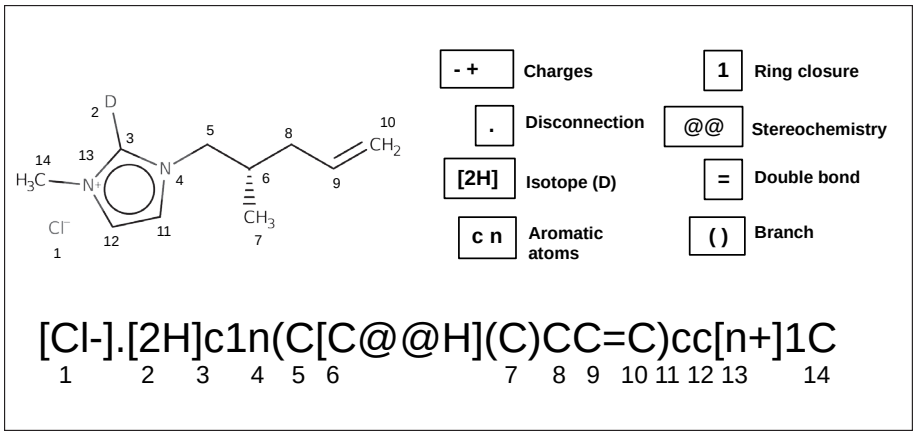

Figure 2. *Example SMILES notation for a 1,3-functionalized imidazolium chloride (the D label on the molecule is for deuterium). Numbers represent the molecule atom mapping. Some selected key features of the SMILES syntax are noted in the legend. The chemical structure depiction and SMILES were generated with ChemAxon MarvinSketch v19.27.*[39]

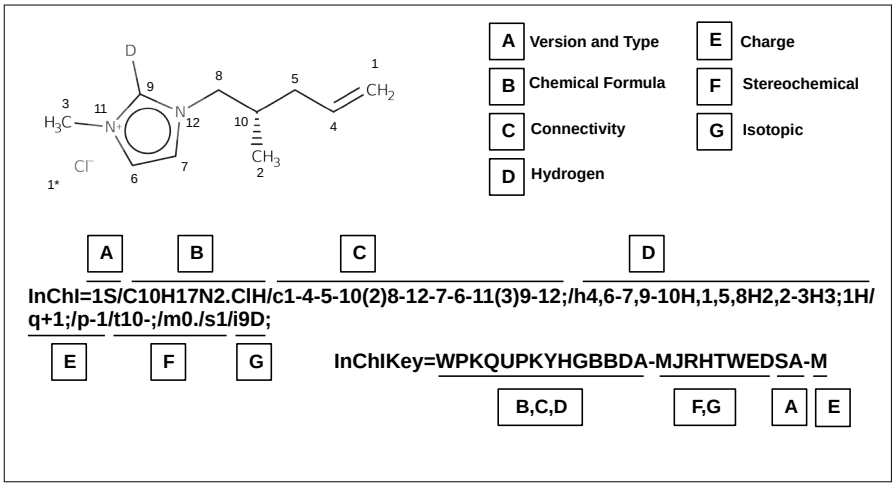

Figure 3. *Example InChI notation for a 1,3-functionalized imidazolium chloride (the D label on the molecule is for deuterium). Numbers represent the InChI molecule atom mapping. Individual layers of the InChI and InChIKey syntax are noted in the legend. The chemical structure depiction was created with ChemAxon MarvinSketch v19.27*[39] *and the InChI calclated directly from a molfile with the InChI 1.05 software.* [38]

(SMARTS) notation, which is a direct extension of the core SMILES syntax.[36] SMARTS allows users to incorporate ambiguity and logical operators into molecular structure searches. Any atom can be defined with the wildcard asterisk symbol and Boolean queries can be constructed with symbols such as exclamations, ampersands/semicolons, and commas to denote NOT, AND, OR, respectively. For example, the query [C,N;H1]-Cl would match molecules containing aliphatic carbon OR nitrogen AND with one hydrogen connected to chlorine with a single bond.[36]

Many software tools and web services are available to help users generate SMILES, InChI, and SMARTS notations. Chemical structure drawing tools such as ChemAxon MarvinSketch®[39] and the free PubChem Sketcher[41] all have robust support for reading and writing SMILES, InChI, and SMARTS. There are also free web conversion tools available for generating SMILES and InChI, such as the NIH CACTVS Chemical Identifier Resolver.[42] In our experience with constructing SMARTS queries, we have found the online SMARTSviewer and desktop SMARTSeditor software particularly helpful.[43, 44] While it is possible to construct SMILES and SMARTS strings manually, you must use software to compute InChIs, as the InChI software is required to apply the appropriate normalization and canonicalization algorithms.[38]

STEP 2a: Construct PubChem Web Service PUG-REST URL

The PubChem PUG-REST service is a web interface that allows users to submit queries as HTTP URLs and retrieve structured machine-readable data, instead of a formatted web page like in a traditional database search.[32, 33] The PUG-REST service and syntax for constructing the requests are well documented in the official PubChem Docs.[45] We recommend testing the PUG-REST service in your web browser before attempting any request with a programming language. Every PUG-REST URL request starts with the same base prefix:

https://pubchem.ncbi.nlm.nih.gov/rest/pug

For clarity, we will abbreviate this prefix as [api]:

[api] = https://pubchem.ncbi.nlm.nih.gov/rest/pug

To this prefix, three more pieces of information are added: an input, operation choice, and an output format:

[api]/**input/operation/output**

The input specifies the record or query to use for the search. For example, a PubChem Compound Identifier (CID) number, SMILES, or InChIKey:

[api]/**compound/cid/795/**

[api]/**compound/smiles/C1=CN=CN1/**

[api]/**compound/inchikey/RAXXELZNTBOGNW-UHFFFAOYSA-N/**

Next, the operation must be specified. Operations tell PubChem what data to retrieve. Examples include compound properties, synonyms, or a description of the record:

[api]/compound/cid/795/**property/MolecularWeight/**

[api]/compound/cid/795/**synonyms/**

[api]/compound/cid/795/**description/**

Finally, the output format is defined such as plain text or a structured machine-readable format like Extensible Markup Language (XML)[46] or Javascript Object Notation (JSON)[47]:

[api]/compound/cid/795/property/MolecularWeight/**TXT**

Here, PubChem returns a value of 68.08 indicating the molecular weight of CID 795 (imidazole).

```
68.080000
```

[api]/compound/cid/795/property/MolecularWeight/**XML**

```
<PropertyTable
xs:schemaLocation="http://pubchem.ncbi.nlm.nih.gov/pug_rest
   https://pubchem.ncbi.nlm.nih.gov/pug_rest/pug_rest.xsd">
  <Properties>
    <CID>795</CID>
    <MolecularWeight>68.08</MolecularWeight>
  </Properties>
</PropertyTable>
```

[api]/compound/cid/795/property/MolecularWeight/**JSON**

```
{
  "PropertyTable": {
    "Properties": [
      {
        "CID": 795,
        "MolecularWeight": 68.08
      }
    ]
  }
}
```

We have found the plain TXT format and the JSON format are the easiest to parse and work with in MATLAB (*vide infra*). When using the PUG-REST service, you will need to be aware that some characters may require URL encoding before sending the request to PubChem[45]; for example, the following URL with the SMILES for cyanoacetic acid containing a triple bond # character will return a bad request error:

[api]/compound/smiles/**C(C#N)C(=O)O/**property/MolecularFormula/JSON

The SMILES need to be URL encoded first. There are many free online services that can compute the URL encoding, or alternatively, text can be URL encoded in MATLAB as follows:

```
>> urlencode('C(C#N)C(=O)O')

ans =

    'C%28C%23N%29C%28%3DO%29O'
```

There is also a fourth parameter, operation options, that can be added at the end of the URL (?option). Operation options are useful for changing default parameters. For example, the Tanimoto coefficient is one of many different measures used to describe the similarity in structure of two different compounds.[3,49] If we wanted to run a fingerprint Tanimoto-based 2D similarity search in PubChem[48,50] on the cyanoacetic acid with a similarity threshold of 95 instead of the default 90, the syntax is as follows::

[api]/compoundfastsimilarity_2d/smiles/C%28C%23N%29C%28%3DO%29O/cids/**JSON?Threshold=95**

```
{
  "IdentifierList": {
    "CID": [
      9740,
      59463708,
      58931961,
      …
    ]
  }
}
```


The aforementioned examples are a small sampling of the available PUG-REST features. The complete and authoritative specification is available from PubChem.[45]

STEP 2b: Construct PubChem Web Service SDQ URL

The PubChem Structured Data Query (SDQ) agent is used internally by PubChem web pages.[51] The syntax is not officially documented; however, the SDQ agent query URLs are visible on the PubChem compound web pages (e.g. see literature download links). So with a bit of caution (see programming etiquette note on pause in Step 4) and experimentation, the basic syntax can be deconstructed. In addition, in our experience, the PubChem staff are highly responsive to questions about programmatic access.

The PubChem SDQ agent is similar to the PUG-REST function where a unique HTTP URL is sent to PubChem, and then structured machine-readable data is returned, such as XML or JSON. The base SDQ URL prefix is as follows:

SDQ = https://pubchem.ncbi.nlm.nih.gov/sdq/sdqagent.cgi?

To the base SDQ URL, an output format, input, and query is added.

[SDQ]**output&query{...{input}}}**

The output format is defined similarly to the PUG-REST API, though it is more explicit. For example, to specify JSON format:

[SDQ]outfmt=json

Queries search the available "collections" within the PubChem data. There are 33 collections that are linked to a specific PubChem Compound Identifier (CID) and include data such as literature collection reference metadata (e.g. PubMed, Springer Nature, Wiley), associated substances (depositor submitted), and assay data. To view the collection data associated with imidazole (CID: 795), the SDQ search is as follows:

[SDQ]outfmt=json&**query={"hide":"*","collection":"*","where":{"ands":{"cid":"795"}}}**


```
{
  "SDQOutputSet": [
    {
      "status": {
       "code": 0
      },
      "totalCount": 1,
      "collection": "compound",
      "type": "flattable",
      "rows": [
      ]
    },
    {
      "status": {
        "code": 0
      },
      "totalCount": 1241,
      "collection": "substance",
      "type": "flattable",
      "rows": [
      ]
    },
{
      "status": {
        "code": 0
      },
      "totalCount": 406,
      "collection": "assay",
      "type": "flattable",
      "rows": [
      ]
    },
     …
     …  (+ 30 more)
     …
```


In the above query, the "hide" parameter specifies to only retrieve the total count, and not the actual data. The "*" is a wildcard for all collections. To retrieve the actual data within the collections, we can change the "hide" to "select" and then specify how many results we want to retrieve with a limiter:

[SDQ]outfmt=json&query={"**select**":"*","collection":"*","where":{"ands":{"cid":"795"}},"start":1,"limit":1}

This request displays one full data record from each of the 33 collections (not shown). We can also specify a collection by replacing the wildcard (*) with the name of a specific collection such as springernature:

[SDQ]outfmt=json&query={"select":"*","collection":"**springernature**","where":{"ands":{"cid":"795"}},"start":1,"limit":1}

```
{
  "SDQOutputSet": [
    {
      "status": {
        "code": 0
      },
      "totalCount": 27824,
      "collection": "springernature",
      "type": "flattable",
      "rows": [
        {
          "cid": 795,
          "sid": 341139407,
          "oid": 7097783,
          "openaccess": 0,
          "scorefloat": 0.1052,
          "articlepubdate": "2002",
          "articletitle": "Cardiovascular Actions of Nitric Oxide",
          "articlejourname": "Nitric Oxide and Infection",
          "subject": "Medicine",
          "doctype": "book chapter",
          "language": "En",
          "doi": "10.1007/0-306-46816-6_7",
          "url": "https://doi.org/10.1007/0-306-46816-6_7",
          "imageurl":        "https://pubchem.ncbi.nlm.nih.gov/image/
imgsrv.fcgi?doi=10.1007/0-306-46816-6_7",
          "extid": "5041005-10699077"
        }
      ]
    }
  ]
}
```


Lastly, it is possible to add a search within a specific collection, limiting to one or more available fields. We can refine the springernature query to only retrieve article metadata with "synthesis" in the title (articletitle) and published in 2019 (articlepubdate):

> [SDQ]outfmt=json&query={"select":"*","collection":"springernature","where":{"ands":{"cid":"795","**articletitle":"synthesis", "articlepubdate":"2019**"}},"start":1,"limit":10}


```
{
 "SDQOutputSet": [
    {
      "status": {
        "code": 0
      },
      "totalCount": 152,
      "collection": "springernature",
      "type": "flattable",
      "rows": [
        {
          "cid": 795,
          "sid": 341139407,
          "oid": 29259880,
          "openaccess": 0,
          "scorefloat": 0.4852,
          "articlepubdate": "2019",
          "articletitle": "Development of a New Method for Synthesis of
Tandem Hairpin Pyrrole–Imidazole Polyamide Probes Targeting Human
Telomeres",
         "articlejourname": "Synthesis and Biological Evaluation of
Pyrrole–Imidazole Polyamide Probes for Visualization of Telomeres",
          "subject": "Chemistry and Material Science",
          "doctype": "book chapter",
          "language": "En",
          "doi": "10.1007/978-981-13-6912-4_2",
          "url": "https://doi.org/10.1007/978-981-13-6912-4_2",
          "extid": "5041005-10699077"
        },
      …
      …  (+ 9 more)
      …
```

There are likely many more features available through the SDQ agent. Some of the SDQ URLs we have seen suggest it is possible to filter and order results; however, we have not experimented with those features yet. The PubChem SDQ agent is still being rapidly developed by PubChem. In our experience the SDQ agent has been stable over the past year with only minor changes observed in the data structure JSON output. We have used the SDQ agent frequently to obtain bibliographic literature data in PubChem. Literature bibliographic data in PubChem are also programmatically accessible through alternative web services including PubChem PUG-VIEW[52] and the NCBI E-Utilities.[53,54] NCBI also maintains a free software package, EDirect, which uses E-Utilities for programmatic access to NCBI data from a Unix Shell[54]. We have recently started to explore EDirect's capabilities and plan to discuss EDirect in a future report.

STEP 3: Retrieve Structured Data with MATLAB "webread" Function

Interacting with the PubChem PUG-REST and SDQ agent through a web browser is perfect for learning and testing, but moving to a programming language like MATLAB offers many advantages. Some of these advantages include the ability to execute a series of searches in a script, capture error messages, and the ability to compile and analyze the data from within MATLAB. The MATLAB function webread allows reading content from web services.[55] The webread function, therefore, can be used instead of an internet browser to interact with PubChem. URLs are constructed as shown in the previous sections and used as the webread input. Before using the webread function, it is a good idea to adjust a few of the webread options including increasing the server timeout (in case PubChem is slow to respond) and specifying the returned data format.

```
% set weboptions
options = weboptions('Timeout', 30,'ContentType','json');
% retrieve synonyms for imidazole (CID 795)
api = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug';
myURL = [api '/compound/cid/795/synonyms/JSON'];
myData = webread(myURL,options)

myData =

  struct with fields:

    InformationList: [1×1 struct]
```

MATLAB imports the requested JSON imidazole synonyms list as a structure array. If we visualize the structure array as a directory tree, it looks like this:

```
myData
  InformationList
      Information
         CID
           795
         Synonym
           imidazole
           1H-Imidazole
           288-32-4
           Glyoxaline
           Imidazol
           ...
```

The synonyms can then be accessed by using dot indexing,[56] returning a cell array.

```
>> myData.InformationList.Information.Synonym

ans =

  395×1 cell array

    {'imidazole'}
    {'1H-Imidazole'}
    {'288-32-4'}
    {'Glyoxaline'}
    {'Imidazol'}
    ...
```

This technique is analogous for using webread and dot indexing for interacting with the PubChem SDQ agent data.

STEP 4: Modifying MATLAB Code with a "For-Loop" to Process Multiple Chemical Search Queries

In the previous section we reviewed how to perform one programmatic PubChem request with webread in MATLAB. In order to make multiple requests, we can still use webread , but we need to add a few programming techniques to process multiple requests. Consider the following programmatic search using the PUG-REST API to perform a similarity search with 1-butyl-3-methylimidazolium:

```
% set weboptions
>> options = weboptions('Timeout', 30,'ContentType','json');

% PubChem API
>> api = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug';

% 1-Butyl-3-methyl-imidazolium; CID = 2734162
>> CID_SS_query = '2734162';

% Search for chemical structures by Similarity Search (SS),
% 2D Tanimoto threshold 97% to 1-Butyl-3-methyl-imidazolium;
% CID = 2734162
>> SS_url = [api '/compound/fastsimilarity_2d/cid/' CID_SS_query...
    '/cids/JSON?Threshold=97'];
>> SS_CIDs = webread(SS_url,options);

% index into SS_CIDs to extract out list of CIDs
>> SS_CIDs = num2cell(SS_CIDs.IdentifierList.CID)

SS_CIDs =

  127×1 cell array

    {[   304622]}
    {[    61347]}
    {[ 11448496]}
    {[ 11171745]}
    {[  2734161]}
    ...
```

The similarity search returned a list of 127 PubChem CIDs, or compounds that match our search. Next, we want to retrieve information associated with each of the CIDs. Manually constructing hundreds of URLs would be inefficient, so we can write a script to assist. In the example below each CID is sent back to PubChem. The Isomeric SMILES are requested through the PUG-REST web service, and the total count of PubMed literature associated with the CID is requested with the SDQ agent (see code on next page).

When the sim_infoSearch.m script is executed, it produces the following output:

```
>> sim_infoSearch

SS_CIDs =

 127×3 cell array

   {[  304622]}    {'CCCCN1C=CN=C1C↵'                }    {[  7]}
   {[   61347]}    {'CCCCN1C=CN=C1↵'                 }    {[ 21]}
   {[11448496]}    {'CCCCN1C=C[N+](=C1)C.[I-]↵'      }    {[  0]}
   {[11171745]}    {'CCCCN1C=C[N+](=C1)C.C(=[N-])=…'}    {[  0]}
   {[ 2734161]}    {'CCCCN1C=C[N+](=C1)C.[Cl-]↵'     }    {[323]}
   {[  118785]}    {'CCCN1C=CN=C1↵'                  }    {[  3]}
   {[ 2734236]}    {'CCCCN1C=C[N+](=C1)C.[Br-]↵'     }    {[323]}
   {[ 2734162]}    {'CCCCN1C=C[N+](=C1)C↵'           }    {[668]}
   {[  529334]}    {'CCCCCN1C=CN=C1↵'                }    {[  1]}
   ...
```

```
sim_infoSearch.m

% PubChem API and weboptions
api = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug';
options_api = weboptions('Timeout', 30);

% PubChem SDQ agent and weboptions
sdq = 'https://pubchem.ncbi.nlm.nih.gov/sdq/sdqagent.cgi?';
options_sdq = weboptions('Timeout', 60,'ContentType','json');

% setup a for loop that processes each CID one-by-one.
for j = 1:length(SS_CIDs)
   CID = SS_CIDs{j};

   % define url for isomeric SMILES property data request
   CID_IsoSMILES_url = [api '/compound/cid/' num2str(CID)...
       '/property/IsomericSMILES/TXT'];
   % retrieve isomeric SMILES
   try
       CID_IsoSMILES = webread(CID_IsoSMILES_url,options_api);
   catch ME
       CID_IsoSMILES = 'not found';
   end
   % be polite to PubChem servers and pause for 1s between requests
   n = 1;
   pause(n)

   % add isomeric SMILES data to 2nd column of SS_CIDs cell array.
   SS_CIDs{j,2} = CID_IsoSMILES;

   % define sdq url to retrieve collection count data
   litQ_url = [sdq 'outfmt=json&query={"hide":"*","collection":"*","where":
   {"ands":{"cid":"'...
       num2str(CID) '"}}}'];
   % retrieve collection count data
   try
       litCountQ = webread(litQ_url, options_sdq);
   catch ME
       litCountQ = 'not found';
   end
   n = 1;
   pause(n)

   % index into the litCountQ structure array
   % the pubmed count data is in the 7th row
   % add selected collection count data to 3rd column of SS_CIDs data array
   SS_CIDs{j,3} = litCountQ.SDQOutputSet{7,1}.totalCount;

end
```

There are four key concepts within the sim_infoSearch.m script:

1. for-loop[57]: The for statement repeats the code so we only need to provide the list of CIDs. The URLs are then generated, differing only by the CID. These unique URLs are then input into the webread function to retrieve the data. for-loops can be used for thousands of requests.
2. try,catch[58]: The try,catch statement can "catch" errors if the "try" fails. This is a basic error handling method in MATLAB and is particularly useful when processing multiple web data requests. In the event the webread fails for a particular request, it sets the data to "not found".
3. IMPORTANT: pause[59]: The pause function allows us to practice good programming etiquette when interacting with PubChem web services. According to the PubChem documentation, the limit is five requests per second.[45] We typically add a pause of 0.5 or 1.0 second between requests, which is well below the limit. If you go over five requests per second, PubChem will likely block your requests.

4. Adding variables to a cell array[60]: We need somewhere to store the data as the for-loop is iterating over the CID list and executing the code. A convenient method is to store the data in a cell array variable. In the sim_InfoSearch script, the line 'SS_CIDs{j,2} = CID_IsoSMILES;' adds the isomeric SMILES data to the second column of SS_CIDs cell array. Since j increases by 1 on each iteration, the first isomeric SMILES gets added to {1,2} (row 1, column 2) , the second to {2,2}, and the third to {3,2}.

STEP 5: Compile MATLAB Code and Outputs into a Live Script for Sharing and Reuse

MATLAB live scripts allow for the combination of both the input code and output in one formatted interactive document. Using the programming techniques described in the previous sections, we created four in-depth live scripts with MATLAB R2020a that further explore how to interact with PubChem programmatically (Supporting Information). The included live scripts allow for programmatic searching of chemical substances by similarity or SMARTS substructure queries. A literature search live script is also included for retrieving references associated with a compound and sharing reproducible literature searches.[20, 29] Finally, a live script is included that is useful for programmatically compiling bibliometric compound data. Compound-based bibliometrics are useful for identifying gaps in the literature.[61-63]

EDUCATIONAL CONSIDERATIONS

Learning Outcomes

Interacting with literature and data from PubChem programmatically presents several learning outcomes for students, including an understanding of:

1. Machine-readable data (e.g. JSON) and chemical file representation (SMILES), identifier (InChI) syntax, and pattern matching (SMARTS).
2. Basic programming techniques including interacting with programmatic web services, structured data queries, for-loop syntax, error handling, and data indexing.
3. How to formulate workflows for reproducible literature searching, compound searching, and data compilations.

We co-developed the PubChem programmatic literature and data MATLAB live scripts with an undergraduate student and an MS chemical engineering student. The code presented in this paper and supporting information are a result of combining and harmonizing code that we have written for several internal research and informational education projects over the past couple years. We envision these MATLAB live scripts will be ideal for chemistry and chemical engineering capstone courses where students can dedicate several weeks to work on one project, focused on reproducible programmatic compilation of data and literature. Since many research methods courses require a written paper, these techniques could be used to produce the literature review. Furthermore, the methods documented herein can be used as a modern case study in an "computer methods" course for chemical engineering seniors and graduate students such as the one taught annually by Professor Bara at the University of Alabama. In order to produce graduates that are prepared to tackle 21st century chemical engineering challenges, students must learn that many modern programming applications will not just rely on code run on the local machine and need to learn methods by which large data sets are retrieved from remote sources and further processed/analyzed.

For instructors already familiar with MATLAB, the techniques presented in this paper for interacting with PubChem programmatically will likely be straightforward and only require a couple days of practice and preparation in order to teach. It will also be helpful to review chemical line notations (SMILES and InChI) and pattern matching syntax (SMARTS) since they are core components of many programmatic chemical information queries. For new users of MATLAB, we recommend first starting with the introductory tutorials from MathWorks,[64] then working through the examples in this paper.

In the classroom we recommend working with students in a similar workflow to that presented in this paper; that is, start with reviewing chemical line notations and experimenting with the PubChem web services in a standard web browser. Then the students can work up to MATLAB in stages, first with one programmatic query, then multiple in a for loop.

There is a learning curve involved with these methods; however, we believe one of the pedagogical benefits is that searching for data and literature programmatically forces students (and instructors) to thoughtfully construct their literature and data search queries. The process requires users to think about key information concepts such as the data source, data search fields, type of search, and information access. As a result, these searches also make student work more transparent for instructors as they can immediately see what kind of literature search the student is performing and how, which can be particularly helpful when offering assistance with keyword ideas. We also note that searching for data and literature programmatically is engaging and fun!

While the live scripts available in the supporting information can be used directly with students to reproduce the datasets and literature searches presented herein, greater pedagogical value will be achieved by using the live scripts as a starting point for related programmatic data compilation and literature search projects. What follows are a few examples of how we envision instructors may incorporate

these live scripts into chemical engineering research methods course assignments:

1. Complete a compound bibliometrics study using the PubChem_SDQ_Bibliometrics live script with a different compound input. Students can then extend the script to plot and analyze the compiled bibliometrics data within MATLAB. From the analyzed data, students can be tasked to identify gaps in the literature and present a systematic overview of a subset of chemical compounds.
2. Adapt the PubChem_SDQ_LitSearch to complete a new programmatic literature search using different compound and metadata field keyword inputs based on an assigned topic area. A more challenging adaption would be to incorporate additional literature search web service functionality into the script such as the Engineering Village programmatic web service[17] or the NCBI Entrez Programming Utilities service.[53,54]
3. The PubChem_Similarity and PubChem_SMARTS live scripts can be adapted with different CID input queries or SMARTS substructure queries, respectively. Both live scripts are also straightforward to extend with additional PubChem compound property requests such as retrieving the IUPAC name or stereochemistry atom counts.

CONCLUSIONS

Core programmatic MATLAB techniques for interacting with PubChem web services are discussed in a guided tutorial style. The guided tutorial includes a review of chemical line notations, the PubChem web service HTTP URL syntax, the MATLAB webread function, and processing multiple PubChem web service requests in MATLAB with a for-loop. Several programmatic searches are demonstrated and discussed, including chemical structure similarity searching, substructure searching, literature searching, and compound-based bibliometric data compiling. These techniques are expanded into MATLAB live scripts included in the supporting information, which are available openly for instructors to use and adapt as desired. The live scripts are useful for chemical education, particularly for teaching students how to interact with information programmatically and reproducibly. We plan to continue promoting programmatic literature searching with MATLAB and are also exploring the Python programming language combined with Jupyter notebooks.

Supporting Information

GitHub Repository (Software license: BSD 2-Clause License): https://github.com/vfscalfani/MATLAB-cheminformatics

ACKNOWLEDGMENTS

JEB acknowledges NSF CBET 1605411 for support of this work. We thank the NIH/NLM/NCBI PubChem staff for their timely helpful responses to our programmatic access questions. VFS thanks ChemAxon for the MarvinSketch academic research license.

REFERENCES

1. Currano J and Roth, D (2014) *Chemical Information for Chemists: A Primer*. Royal Society of Chemistry. Cambridge, UK.
2. Krallinger M, Rabal O, Lourenço A, Oyarzabal J, and Valencia A (2017) Information Retrieval and Text Mining Technologies for Chemistry. *Chem. Rev*. 117(12): 7673-7761. https://doi.org/10.1021/acs.chemrev.6b00851
3. Engel TD and Gasteiger J (2018) *Chemoinformatics : basic concepts and methods*. Wiley-VCH, Weinheim, Germany.
4. Engel TD and Gasteiger J (2018) *Applied chemoinformatics : achievements and future opportunities*. Wiley-VCH, Weinheim, Germany.
5. Senderowitz H and Tropsha A (2018) Materials Informatics. *J. Chem. Inf. Model*. 58(7): 1313–1314. https://doi.org/10.1021/acs.jcim.8b00016
6. Audus DJ and de Pablo JJ (2017) Polymer Informatics: Opportunities and Challenges. *ACS Macro Letters*. 6(10): 1078–1082. https://doi.org/10.1021/acsmacrolett.7b00228
7. Medford AJ, Kunz MR, Ewing SM, Borders T, and Fushimi R (2018) Extracting Knowledge from Data through Catalysis Informatics. *ACS Catalysis*. 8(8): 7403–7429. https://doi.org/10.1021/acscatal.8b01708
8. Shacham M, Brauner N, and Cutlip MB (2003) An exercise for practicing programming in the ChE curriculum: Calculation of thermodynamic properties using the Redlich-Kwong Equation of State. *Chem. Eng. Educ*. 37: 148–153.
9. Li X and Huang Z (Jacky) (2017) An inverted classroom approach to educate MATLAB in chemical process control. *Educ. Chem. Eng*. 19: 1–12. https://doi.org/10.1016/j.ece.2016.08.001
10. Joss L and Müller EA (2019) Machine Learning for Fluid Property Correlations: Classroom Examples with MATLAB. *J. Chem. Educ*. 96(4): 697–703. https://doi.org/10.1021/acs.jchemed.8b00692
11. Lee K, Comolli NK, Kelly WJ, and Huang Z (2015) MATLAB-based teaching modules in biochemical engineering. *Chem. Eng. Educ*. 49(2): 95–100.
12. Ricker NL (2001) Using MATLAB/Simulink for data acquisition and control. *Chem. Eng. Educ*. 35(4): 286–289.
13. Housam B (2008) Equilibrium-stage separations using Matlab and Mathematica. *Chem. Eng. Educ*. 42:69–73.
14. Golman B (2019) A set of Jupyter notebooks for the analysis of transport phenomena and reaction in porous catalyst pellet. *Comput. Appl. Eng. Educ*. 27: 531–542. https://doi.org/10.1002/cae.22095.
15. Weiss CJ (2017) Introduction to Stochastic Simulations for Chemical and Physical Processes: Principles and Applications. *J. Chem. Educ*. 94(12): 1904–1910. https://doi.org/10.1021/acs.jchemed.7b00395
16. Kim S et al. (2016) PubChem Substance and Compound databases. *Nucleic Acids Res*. 44(D1): D1202–D1213. https://doi.org/10.1093/nar/gkv951
17. Elsevier Engineering Village API. https://dev.elsevier.com Accessed April 3, 2020.
18. Kitchin JR (2015) Examples of Effective Data Sharing in Scientific Publishing. *ACS Catal*. 5(6): 3894–3899. https://doi.org/10.1021/acscatal.5b00538
19. Perkel JM (2018) Why Jupyter is data scientists' computational notebook of choice. *Nature*. 563(7729): 145–146. https://doi.org/10.1038/d41586-018-07196-1.
20. Rule A et al. (2019) Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks. *PLOS Computational*

Biology, 15:(7), e1007007.https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007007
21. Wolfram Notebooks. https://www.wolfram.com/notebooks/. Accessed April 3, 2020.
22. Project Jupyter. https://jupyter.org/. Accessed April 3, 2020.
23. MathWorks Documentation: Live Scripts and Functions. https://www.mathworks.com/help/matlab/live-scripts-and-functions.html Accessed April 3, 2020.
24. Kitchin JR (2016) Data sharing in Surface Science. Surface Science, 647: 103–107. https://doi.org/10.1016/j.susc.2015.05.007
25. Kitchin JR. org-ref: org-mode modules for citations, cross-references, bibliographies in org-mode and useful bibtex tools to go with it. https://github.com/jkitchin/org-ref. Accessed April 3, 2020.
26. Rose ME and Kitchin JR (2019) pybliometrics: Scriptable bibliometrics using a Python interface to Scopus. *SoftwareX* 10:100263. https://doi.org/10.1016/j.softx.2019.100263.
27. Heldens S, Sclocco A, and Dreuning H (Zenodo, 2019) NLeSC/*automated-literature-analysis* https://doi.org/10.5281/zenodo.3386072
28. Sydow D, Morger A, Driller M, and Volkamer A (2019) TeachOpenCADD: a teaching platform for computer-aided drug design using open source packages and data. *J. Cheminform*. 11:29 (2019). https://doi.org/10.1186/s13321-019-0351-x
29. Vaganay A. To save the research literature, let's make literature reviews reproducible. https://blogs.lse.ac.uk/impactofsocialsciences/2018/06/19/to-save-the-research-literature-lets-make-literature-reviews-reproducible/. Accessed April 3, 2020.
30. Kim S et al. (2019) PubChem 2019 update: Improved access to chemical data. *Nucleic Acids Research*, 47(D1): D1102–D1109. https://doi.org/10.1093/nar/gky1033
31. PubChem Docs: Downloading PubChem Data. https://pubchemdocs.ncbi.nlm.nih.gov/downloads. Accessed April 3, 2020.
32. Kim S, Thiessen PA, Bolton EE, and Bryant SH (2015) PUG-SOAP and PUG-REST: Web services for programmatic access to chemical information in PubChem. Nucleic Acids Research, 43(W1):W605–W611. https://doi.org/10.1093/nar/gkv396
33. Kim S, Thiessen PA, Cheng T, Yu B, and Bolton EE (2018) An update on PUG-REST: RESTful interface for programmatic access to PubChem. Nucleic Acids Res. 46(W1), W563–W570. https://doi.org/10.1093/nar/gky294
34. Swain M. PubChemPy: Python wrapper for the PubChem PUG REST API. https://github.com/mcs07/PubChemPy. Accessed April 3, 2020.
35. Szöcs E. webchem: Chemical Information from the Web. https://github.com/ropensci/webchem. Accessed April 3, 2020.
36. Daylight Chemical Information Systems. Daylight Theory Manual v4.9. https://www.daylight.com/dayhtml/doc/theory/. Accessed April 3, 2020.
37. Weininger D (1988) SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci*. 28: 31–36 https://doi.org/10.1021/ci00057a005.
38. Heller SR, McNaught A, Pletnev I, Stein S, and Tchekhovskoi D (2015) InChI, the IUPAC International Chemical Identifier. *J. Cheminform*. 7:23. https://doi.org/10.1186/s13321-015-0068-4.
39. ChemAxon MarvinSketch. https://chemaxon.com/products/marvin Accessed April 3, 2020.
40. Chambers J, Davies M, Gaulton A, Hersey A, Velankar S, Petryszak R, Hastings J, Bellis L, McGlinchey S, and Overington, JP (2013) UniChem: A unified chemical structure cross-referencing and identifier tracking system. J. Cheminform. 5(1): 3. https://doi.org/10.1186/1758-2946-5-3.
41. Ihlenfeldt WD, Bolton EE, and Bryant SH (2009) The PubChem chemical structure sketcher. *J. Cheminform*. 1:20. https://doi.org/10.1186/1758-2946-1-20.
42. NIH Chemical Identifier Resolver. https://cactus.nci.nih.gov/chemical/structure. Accessed April 3, 2020.
43. Schomburg K, Ehrlich H-C, Stierand K, and Rarey M (2010) From Structure Diagrams to Visual Chemical Patterns. *J. Chem. Inf. Model*. 50(9): 1529–1535. https://doi.org/10.1021/ci100209a.
44. Schomburg KT, Wetzer L, and Rarey M (2013) Interactive design of generic chemical patterns. Drug Discov. *Today* 18(13-14): 651–658. https://doi.org/10.1016/j.drudis.2013.02.001.
45. PubChem Docs: PUG-REST. https://pubchemdocs.ncbi.nlm.nih.gov/pug-rest. Accessed April 3, 2020.
46. Extensible Markup Language (XML) 1.0 (Fifth Edition). https://www.w3.org/TR/xml/. Accessed April 3, 2020.
47. Introducing JSON. https://www.json.org/json-en.html. Accessed April 3, 2020.
48. Kim S (2016) Getting the most out of PubChem for virtual screening. Expert Opin. *Drug Discov*. 11(9): 843–855. https://doi.org/10.1080/17460441.2016.1216967.
49. Holliday JD, Hu C-Y, and Willett P (2002) Grouping of Coefficients for the Calculation of Inter-Molecular Similarity and Dissimilarity using 2D Fragment Bit-Strings. *Comb. Chem. High Throughput Screen*. 5(2): 155-166. https://doi.org/10.2174/1386207024607338.
50. PubChem Substructure Fingerprint, v1.3. ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.pdf. Accessed April 3, 2020.
51. Warr WA (2016), Herman Skolnik Award Symposium 2016 Honoring Stephen Bryant and Evan Bolton - Open chemical information: where now and how? *ACS Chemical Information Bulletin*. 68 (4): 37–39. https://digital.library.unt.edu/ark:/67531/metadc967401/
52. Kim S, Thiessen PA, Cheng T, Zhang J, Gindulyte A, and Bolton EE (2019) PUG-View: Programmatic access to chemical annotations integrated in PubChem. *J. Cheminform*. 11(1): 56. https://doi.org/10.1186/s13321-019-0375-2
53. S. Kim, et al. Literature information in PubChem: associations between PubChem records and scientific articles. *J. Cheminform*. 8, (2016). https://doi.org/10.1186/s13321-016-0142-6
54. NCBI Entrez Programming Utilities Help. https://www.ncbi.nlm.nih.gov/books/NBK25501/. Accessed May 22, 2020.
55. MathWorks Documentation: webread. https://www.mathworks.com/help/matlab/ref/webread.html. Accessed May 26, 2020.
56. MathWorks Documentation: Access Data in a Structure Array. https://www.mathworks.com/help/matlab/matlab_prog/access-data-in-a-structure-array.html. Accessed May 26, 2020.
57. MathWorks Documentation: for. https://www.mathworks.com/help/matlab/ref/for.html. Accessed May 26, 2020.
58. MathWorks Documentation: try,catch. https://www.mathworks.com/help/matlab/ref/try.html. Accessed May 26, 2020.
59. MathWorks Documentation: pause. https://www.mathworks.com/help/matlab/ref/pause.html. Accessed May 26, 2020.
60. MathWorks Documentation: Cell Arrays. *https://www.mathworks.com/help/matlab/cell-arrays.html*. Accessed May 26, 2020.
61. Tomaszewski R (2019) Substance-Based Bibliometrics: Identifying Research Gaps by Counting and Analyzing Substances. *ACS Omega*. 4(1): 86–94. https://doi.org/10.1021/acsomega.8b02201.
62. Barth A and Marx W (2012) Stimulation of Ideas through Compound-Based Bibliometrics: Counting and Mapping Chemical Compounds for Analyzing Research Topics in Chemistry, Physics, and Materials Science. *ChemistryOpen* 1: 276–283. https://doi.org/10.1002/open.201200029.
63. Scalfani VF, Alshaikh AA, and Bara JE (2018) Analysis of the Frequency and Diversity of 1,3-Dialkylimidazolium Ionic Liquids Appearing in the Literature. Ind. *Eng. Chem. Res*. 57(47): 15971–15981. https://doi.org/10.1021/acs.iecr.8b02573.
64. MathWorks MATLAB and Simulink Tutorials. https://www.mathworks.com/support/learn-with-matlab-tutorials.html. Accessed May 26, 2020. ❒