qd – Build your own LS-DYNA® Tools Quickly in Python

C. Diez
Lasso GmbH Germany

Abstract

CAE is a large field with many different use -cases. This high diversity yielded a large variety of tools, which help us engineers to achieve a high working performance every day. Unfortunately the situation arises, where engineers need to solve an additional, sometimes client or company-specific tasks, where no tool yet exists. Many of these tasks require usually only one key element: comfortable and fast data access.

The only way to access data nowadays is through another CAE tool. Engineers currently develop their own tools in other CAE frameworks (ANSA/META, Hypermesh/Hyperview, LS-PrePost®, etc.) and in consequence are bound to the tools themselves. For being part of another tool, new solutions struggle with many issues:

- *bad scripting API or old scripting languages of the tools*
- *bad performance for large computational tasks*
- *high effort to incorporate additional libraries*
- *it's hard to maintain, update and share code*
- *difficult to pair with state of the art software development tools*

The talk presents a new, next generation digitalization strategy with qd-cae-python. qd-cae-python is a free python library, which gives users comfortable access to LS-DYNA data, let it be post-processing or pre-processing. The idea is to analyze and manipulate data with the smallest amount of coding effort possible. With this library, engineers can solve their own issues themselves in a very quick manner and free themselves from the dependency of CAE tools. For providing simplistic data access, the library also encourages the development of new utilities and tools, which can again be shared more easily.

Motivation

Even though performing simulations is much cheaper than physical experiments, the departments using Computer Aided Engineering (CAE) shall also reduce their costs, while the amount of work increases steadily. The major strategy to reduce costs as a CAE department is automation. Automation can involve little things, such as automatic report generation for simulations, but also involve larger tasks, such as automatically meshing geometrically changing parts. From the abstract point of view, this strategy is valid and makes entirely sense.

The hidden, technical issue is that every technology has its limits in terms of usability. Most of the current CAE tools are being developed since the beginning of CAE, struggling to keep up the pace of modern technology. A simple example is that all CAE tools provide scripting functionality to automate "manual" processes. From an engineer's point of view this seems fine, but it has many downsides other industries have overcome for a long time:

- bad scripting API or old scripting languages
- bad performance for large computational tasks
- high effort to incorporate additional libraries
- it's hard to maintain, update and share code
- difficult to pair with state of the art software development tools

In consequence, development or customization of an own tool is limited, since it is very difficult to maintain the scripts in the environment of another tool. Also advanced automation (e.g. Simulation Data Analytics [1,2]) is computationally heavy, and simply exceeds the capabilities of data interfaces such as Postprocessors. Additionally no standard file format exists compared to other disciplines (e.g. image recognition). Postprocessors do provide means of access to data, but their environment is meant to automate "manual" tasks, and in consequence do not work very well when operating big computational tasks on an entire mesh.

qd – more than a free python library

qd is a small, international community of engineers and developers, who are working on advanced CAE topics in their spare time and who also realize their own ideas. The main driving force is the advancement of technology, as well as a playful, non-competitive environment encouraging creativity. Besides publishing code, we also write articles or upload tutorials on Youtube.

The idea of the qd python library is to give engineers free and simple data access to LS-DYNA data. If data access is made easy, then also automation is also made easy and creativity arises. The library is openly hosted and maintained on Github [3], where users can also ask questions by opening issues or join the chat on Gitter [4]. The development of the library follows two essential ideas:

"Creativity originates from freedom and thus simplicity."

**"Don't provide another software solution,
let people build their own solutions quickly"**

Python is currently the by far most frequently used scripting environment throughout all industries [5], except for web development. Since CAE data analysis has a high performance demand, the code behind the python package is written mostly in C++ and just uses python as comfortable wrapper, thus combining the best of both worlds: Performance and Usability. While the speed when iterating over a mesh is quite good, the library is not yet optimized well in terms of I/O. The library can also read files compressed with FEMZIP directly.

Simple from the Beginning: Installation

Python provides the standard platform PiPy [6] to retrieve or upload packages. Since also the qd library is uploaded to PiPy, installation on Windows is done in one line:

```
python -m pip install qd
```

Figure 1: Command for installing the qd library with the python package manager ‘pip’.

Python hereby fetches the most recent installation file (“python wheel” *.whl) from PiPy and installs it. One can also download the installation file on Github and simply give the path to this file. As a note, installation on Linux usually depends on the system, thus no reliable default-version can be provided for all systems. Nonetheless we also compile our packages for Anaconda Python on the latest OpenSuse Linux, which works with a lot of Linux distributions. Since the code is open-source, one can also compile it oneself, to ensure a good performance on one’s own system.

Postprocessing: Accessing D3plot Data

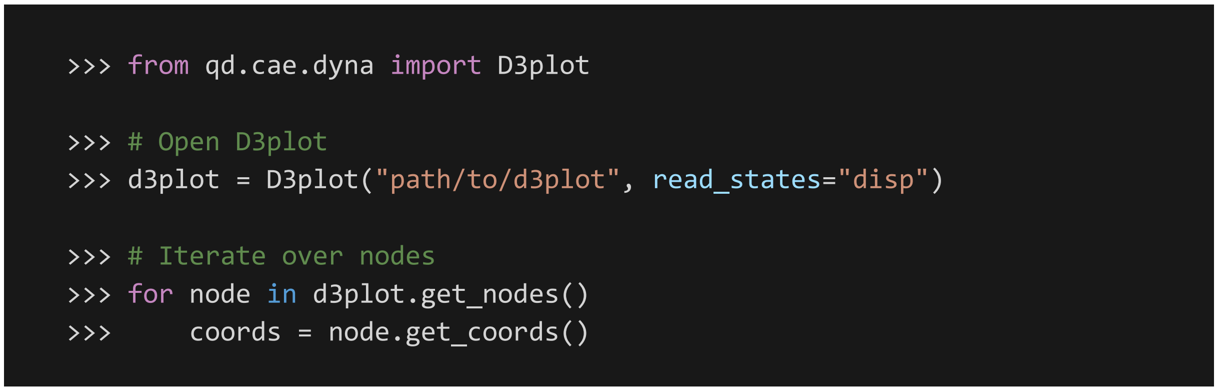
The oldest and also most mature feature of the library is giving access to the data of a D3plot. The library therefore has two classes:

D3plot	**RawD3plot**

• Works object-oriented
  o Node
  o Element
  o Part
• Loads only what you specify
• Many, but not all result fields supported
	• Works array-based (access to raw data)
	• Loads all data
	• Access to all result fields
	• Supports HDF5 export
	• Recommended only for specific purposes

The main class is D3plot, which loads specified result fields and provides access to them. The class is object-oriented, thus provides Node, Element and Part objects as containers for the respective data. The RawD3plot is working array-based and as the name suggests gives access to the raw data of the result file. Accessing the raw data arrays is only recommended for specific purposes. Therefore, this article will in the following focus on the D3plot class.

As an example the D3plot class makes it very simple to iterate over all nodes of a mesh and get their time series of coordinates:

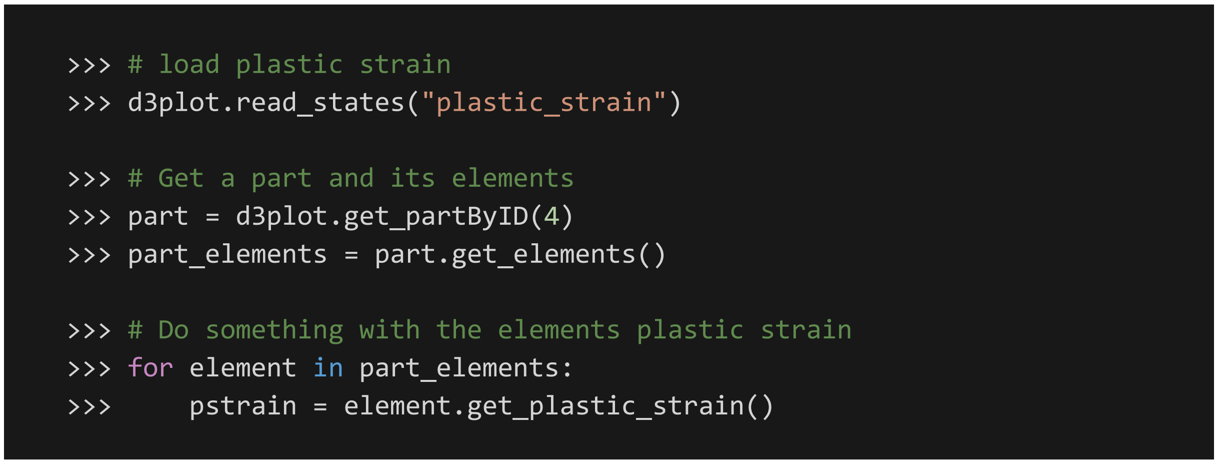


```
>>> from qd.cae.dyna import D3plot

>>> # Open D3plot
>>> d3plot = D3plot("path/to/d3plot", read_states="disp")

>>> # Iterate over nodes
>>> for node in d3plot.get_nodes()
>>>     coords = node.get_coords()
```

Figure 2: The code example shows the simplicity of retrieving nodes and their coordinates from a D3plot.

The coordinates array has the shape (nTimesteps x 3), where 3 is the number of dimension. For nodes LS-DYNA outputs only the displacement, velocity and acceleration field.

In a similar way, also the data of parts and elements are accessible:



```
>>> # load plastic strain
>>> d3plot.read_states("plastic_strain")

>>> # Get a part and its elements
>>> part = d3plot.get_partByID(4)
>>> part_elements = part.get_elements()

>>> # Do something with the elements plastic strain
>>> for element in part_elements:
>>>     pstrain = element.get_plastic_strain()
```

Figure 3: In this code example, the plastic strain of a parts' elements is retrieved.

In the previous code example, the D3plot first loads the plastic strain field, since it isn't loaded yet. Thereafter a Part object is retrieved from its part id in the input deck. One can easily get all the elements of a part by using the 'part.get_elements' method. It's then possible to iterate over all elements and work with their results, such as the time series of plastic strain here.

Preprocessing: Handling Keyfiles

A Keyfile is an LS-DYNA input file, which contains all the definitions for the simulation. Building up a new file is much easier compared to changing it, since text parsing and filtering is a tedious task. As an engineer one should not have to write a file parser just to edit such textual files.

The Keyfile class has been developed just recently with the following major purpose:

- Create or read an arbitrary Keyfile
- Provide a simple API for manipulation
- Export an identical file again (including comments), except for the changes performed

This can be guaranteed, since the file is kept entirely as text in memory (if the mesh is not parsed), and provides means to manipulate the text of each keyword smartly. The supported features are:

- Adding, removing or editing arbitrary keywords, cards and fields
- (Re)formatting of comments and fields (text alignment, etc.)
- Include handling
- Creating and parsing meshes

The library does not deal with the following topic though:

- Removal of mesh entities (Node, Part, Element), if the mesh is parsed.

When opening a Keyfile, are three options:

```
>>> kf = KeyFile("path/to/keyfile",
                 read_keywords=True,
                 parse_mesh=True,
                 load_includes=True)
```

If all includes are loaded with the ‘load_includes’ option, then one can also access them, otherwise the include keywords are read, but not loaded. The ‘read_ keywords’ option loads all keywords and if a keyword is not recognized as *PART, *NODE or *ELEMENT, it is treated as a generic keyword. Note that if the mesh is not parsed with the ‘parse_mesh’ option, also the mesh-related keywords are treated as generic keywords. If the mesh is parsed, the reader also checks the mesh for consistency and thereafter one can also access the mesh as if it was a D3plot object. In following figure is an example for the manipulation of a KeyFile.

```
>>> # check which keywords are in the file
>>> kf.keys()
['*BOUNDARY_SPC_SET_ID', '*CONSTRAINED_INTERPOLATION_SPOTWELD',
 '*CONTACT_AUTOMATIC_SINGLE_SURFACE_ID',
'*DATABASE_CROSS_SECTION_PLANE_ID', '*ELEMENT_SHELL', '*END',
'*HOURGLASS_TITLE', '*INCLUDE', '*INITIAL_VELOCITY', '*KEYWORD', '*NODE',
'*PART', '*PART_CONTACT', '*PART_INERTIA_CONTACT', '*SECTION_SHELL_TITLE',
'*SET_NODE_LIST_TITLE', '*SET_PART_LIST_TITLE']

>>> # find and display all part keywords
>>> kf["*PART"]
[<Keyword: *PART>]

>>> # take the first/only keyword
>>> kw = kf["*PART"][0]
>>> print(kw)
'''
$--------------------------------------------------
$ Parts, Sections, and Materials
$--------------------------------------------------
*PART
$# title
engine part number one
$#     pid     secid       mid     eosid      hgid
2000001   2000001   2000017
'''

>>> # searches pid in the comments above and sets the value
>>> kw["pid"] = 100

>>> # get "mid" field from indexing (card 1, field 2)
>>> kw[1,2]
2000017
```

Figure 5: This example first extracts a part keyword from the Keyfile in figure 4 and then modifies its pid, as well as its mid.

Tools: 3D HTML Plotting

By simply giving access to the raw simulation data, many utilities and tools could be built based on the library. The first utility was exporting an entire model or part into an interactive 3D HTML. With this technology, one can view the model in the browser without requiring any plugins, programs or anything else. This functionality can be triggered by using the plot function of a Part, KeyFile or D3plot.

```
>>> # export the 3D results of a specific part
>>> part = d3plot.get_partByID(4)
>>> part.plot(iTimestep=22, export_filepath="model.html")

>>> # directly triggers the browser (see figure below)
>>> d3plot.plot(iTimestep=33
>>>             element_result="plastic_strain",
>>>             fringe_bounds=[0, 0.025]
```

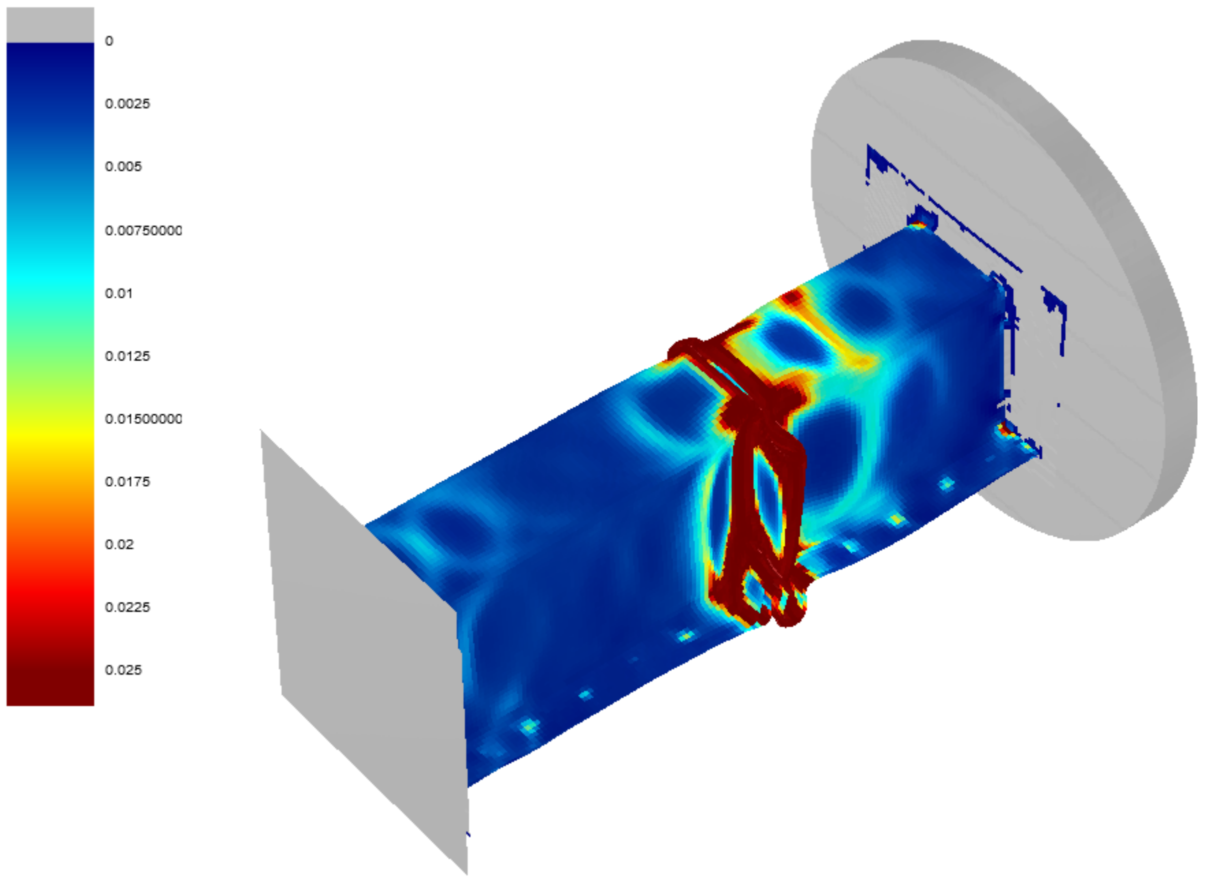
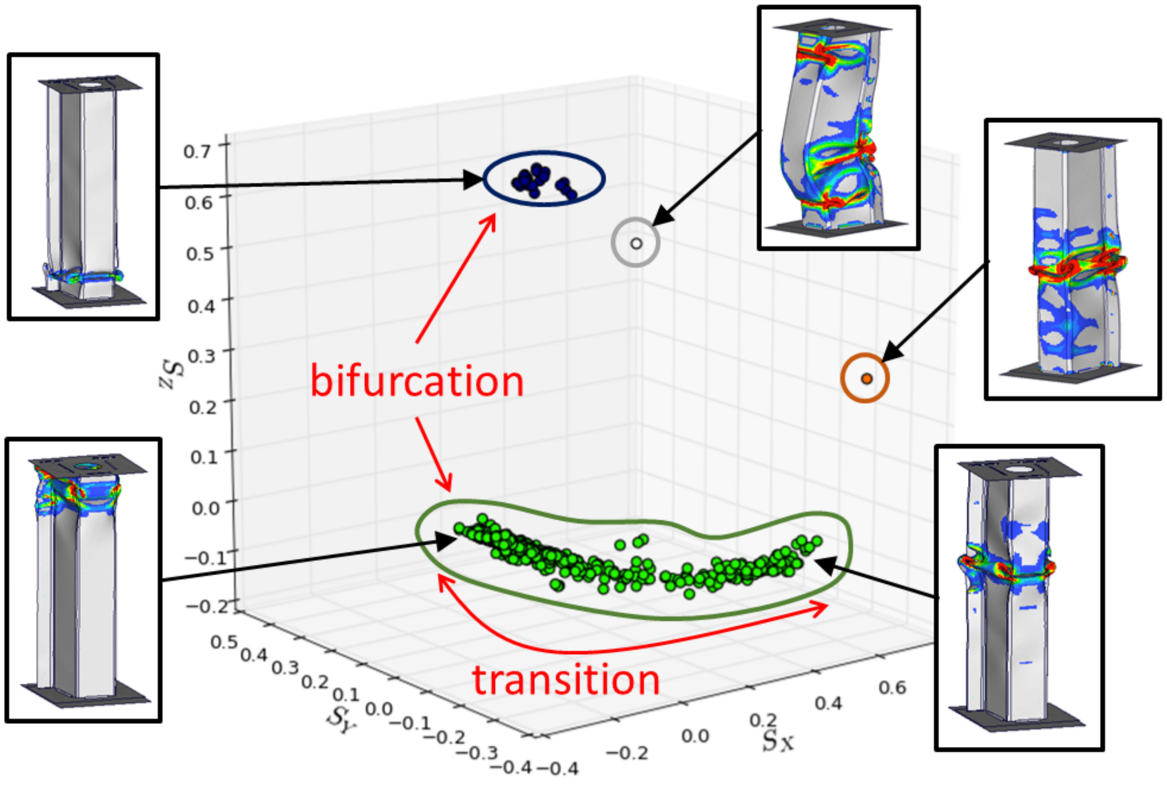


Figure 6: Plotting a D3plot or Part exports a 3D HTML and opens it with the browser, if no filepath was specified.

While the 3D HTML plotting comes in handy for parts and components, the files tend to become quite big for entire cars (up to 200MB and more) and start to stutter in the browser. As a side note, the development of this prototype took only 1 day and this is in a very early stage. With appropriate compression techniques, these 3D HTML reports could become very useful for reporting, especially in combination with cloud technology. For that reason, we develop the topic of web visualization further and want to make it usable for everyone.

Tools: Machine Learning Postprocessing

In [2,3] the python library was used in order to analyze design study of more than a thousand simulations with Machine Learning. By having access to the raw data, a procedure was developed to compute the similarity between simulation results. This similarity between simulations can be plotted in a 3-dimensional embedding. This embedding simply shows one point for each simulation. Close points in the plot represent simulations with very similar results. In the example below, clouds of simulations form, which have similar deformation behavior.



By using this procedure one can identify two main clusters. The green cluster represents samples buckling mostly in the top area, whereas the dark, blue cluster represents samples buckling in the bottom area. Also two outliers can be identified to have a very rare buckling behavior. Loading and analyzing these 1200 simulations took around 35 minutes on a desktop machine (16 cores, 64GB RAM) using the qd library.

Summary

The qd library is a free python library providing data access and tools related to LS-DYNA. The library makes it possible to develop tools of any size quite easily since the library doesn't have the constraints of a specific CAE tool or environment. The first example was a web-visualization, which exports a simulation model into a 3D HTML. The second example was using the library to analyze thousands of Crash Simulations with Machine Learning, which would exceed the capabilities of nowadays postprocessors. This not only pushes democratization of CAE, but also suits our modern CAE world better, where a rigid environment is very hurtful for automation of processes. The library is free of charge but can be supported financially on Patreon [7].

References

[1] C. Diez, P. Kunze, D. Toewe, C. Wieser, L. Harzheim, A. Schumacher, Big.Data based rule-finding for analysis of crash simulations, WCSMO 12, Germany, Braunschweig, 2017,
[2] C. Diez, Machine Learning Process to analyze Big-Data from Crash Simulations, BETA CAE International Conference, 2017, Greece, Thessaloniki.
[3] Github, qd repository, https://github.com/qd-cae
[4] Gitter chat qd, https://gitter.im/qd-cae-python/Lobby
[5] Stackoverflow Trends, https://stackoverflow.blog/2017/05/09/introducing-stack-overflow-trends/, 2017
[6] PiPy Python Package Index, https://pypi.python.org/pypi
[7] Patreon support website, https://www.patreon.com/qd_cae