geomdl-cli Documentation

Onur Rauf Bingol

Jul 20, 2019

Contents:

geomdl-cli is a tool for using NURBS-Python (geomdl) from the command line.

CHAPTER 1

What is geomdl-cli?

1.1 Introduction

geomdl.cli module provides a set of commands for using geomdl NURBS and B-spline evaluation library from the command line.

The geomdl-cli package automatically installs the following as its dependencies:

- geomdl
- Matplotlib
- Jinja2
- ruamel.yaml
- libconf

Although geomdl-cli is a pure Python package, it is only tested with Python v3.5.x and later.

1.2 Why geomdl-cli?

- Highly configurable via command line parameters
- Direct integration with geomdl library
- Ability to add, extend or override commands without touching the base code
- Pure Python, no external C/C++ or FORTRAN dependencies needed for installation and command execution

1.3 Author

- Onur R. Bingol (@orbingol)

1.4 License

MIT

CHAPTER 2

Installing geomdl-cli

The recommended method for installation is using pip.

```
pip install --user geomdl.cli
```

Alternatively, you can install the latest development version from the GitHub repository:

- Clone the repository: `git clone https://github.com/orbingol/geomdl-cli.git`
- Inside the directory containing the cloned repository, run: `pip install --user .`
- The setup script will install all required dependencies

2.1 Docker Containers

A collection of Docker containers is provided on Docker Hub containing NURBS-Python, Cython-compiled core and the command-line application. To get started, first install Docker and then run the following on the Docker command prompt to pull the image prepared with Python v3.5:

```
$ docker pull idealabisu/nurbs-python:py35
```

On the Docker Repository page, you can find containers tagged for Python versions and Debian (no suffix) and Alpine Linux (`-alpine` suffix) operating systems.

After pulling your preferred image, run the following command:

```
$ docker run --rm -it --name geomdl -p 8000:8000 idealabisu/nurbs-python:py35
```

In all containers, Matplotlib is set to use `webagg` backend by default. Please follow the instructions on the command line to view your figures.

CHAPTER 3

Using geomdl-cli

geomdl-cli uses the following structure for executing the commands:

```
geomdl-cli {command} {options} {parameters}
```

where

- `geomdl-cli` is the name of the command line application
- `{command}` corresponds to the command to be executed, see the list below
- `{options}` corresponds to the command input
- `{parameters}` corresponds to the command parameters, such as `--help` or `--delta`

Please see the individual command help for details on `{options}` and `{parameters}` values.

3.1 Available commands

- `help`: displays the help message
- `version`: displays the package version
- `config`: displays the configuration
- `plot`: plots single or multiple NURBS curves and surfaces using Matplotlib
- `eval`: evaluates NURBS shapes and exports the evaluated points in supported formats
- `export`: exports NURBS shapes in supported CAD exchange formats

3.2 Individual command help

Individual command help can be displayed via `--help` parameter.

```
geomdl-cli {command} --help
```

where `{command}` corresponds to the command to be executed.

3.3 Defining input file format

By default, the input file format is determined from the file extension. However, in case of a file with no or different extension, the input file format must be defined manually via `--type` parameter.

```
geomdl-cli {command} my_file --type=yaml
```

Supported input file formats: yaml, cfg, json

3.4 Examples

Please check the GitHub repository for example input files.

CHAPTER 4

Configuration

geomdl-cli allows users to override existing commands and configuration variables with and option add custom commands and configuration variables for the custom commands as well.

These changes can be directly applied by creating a special directory, `.geomdl-cli`. geomdl-cli automatically checks for the existence of this directory in the following locations:

- user home directory (e.g. `/home/user/.geomdl-cli`, `C:\Users\user\.geomdl-cli`)
- directory that you are running geomdl-cli

In the listed directories, geomdl-cli tries to load the custom configuration file `config.json` which is in JSON format with special directives discussed below. If the file doesn't exist, geomdl-cli will continue working without any problems.

The following sections discuss the details of the JSON file and the customization options.

4.1 Structure of the config file

The config file is structured as follows:

```
{
  "configuration": {
    "test_configuration": "default configuration data"
  },
  "commands": {
    "test": {
      "desc": "command description, displayed when 'geomdl-cli help' is called",
      "module": "geomdl-test.test_module",
      "func": "test_function",
      "func_args": "0"
    }
  }
}
```

There are two main sections: **configuration** and **commands**, which are used to create user-defined configuration variables and commands for geomdl-cli.

In the example above, a command named `test` is created and this command will be executed when `geomdl-cli test` is called from the command line. A command definition can contain 4 elements:

- `desc` contains the command description text displayed when `geomdl help` is called
- `func` is the function to be called when the command is called, e.g. `geomdl-cli test`
- `func_args` is the number of arguments that the function `func` takes
- `module` points to the Python module that is required to import for calling the function `func`

Configuration variables will be available in the code via the following import statement:

```
from geomdl.cli import config
```

`config` is a dictionary containing the default and the user-defined configuration variables. In the example above, the configuration variable can be accessed using `config['test_configuration']`.

4.2 Creating commands

To create commands, the initial step is creating `.geomdl-cli` custom configuration directory and `config.json` file as instructed above. geomdl-cli tool adds the custom configuration directories to the Python path; therefore, any Python packages inside the custom configuration directory will be available to the geomdl-cli tool at the run time.

As an example, let's create a directory `geomdl-test` inside the custom configuration directory and then create an empty file `__init__.py` inside `geomdl-test` directory. A directory with an empty `__init__.py` file defines a basic Python package. Additionally, let's create another file `test.py` inside `geomdl-test` directory and put the following code in `test.py` file:

```
def test_function(**kwargs):
    print("my test function")
```

We have created a Python module with a simple function. Let's assign this function to a command. Create the file `config.json` inside the custom configuration directory and put the following in the file:

```
{
  "commands": {
    "test": {
      "desc": "test command description",
      "module": "geomdl-test.test",
      "func": "test_function"
    }
  }
}
```

Now, let's test our new command. Open your command-line and type **geomdl-cli help**. You will see your new command at the bottom of the list.

```
$ geomdl-cli help
GEOMDL-CLI - Run NURBS-Python (geomdl) from the command line

geomdl-cli is a command line tool for 'geomdl', a pure Python NURBS and B-Spline␣
↪library.
```

(continues on next page)

(continued from previous page)

```
Usage:

    geomdl-cli {command} {options}

Individual command help available via

    geomdl-cli {command} --help

Available commands:

    help                displays the help message
    version             displays the package version
    config              displays the configuration
    plot                plots single or multiple NURBS curves and surfaces using␣
↪matplotlib
    eval                evaluates NURBS shapes and exports the evaluated points in␣
↪various formats
    export              exports NURBS shapes in common CAD exchange formats
    test                test command description
```

Let's also test the output of our new command. Type **geomdl-cli test** to see the command output.

```
$ geomdl-cli test
my test function
```

Let's update our new command to take user input from the command line. Update `test.py` as follows:

```
def test_function(test_input, **kwargs):
    print("my test function prints", str(test_input))
```

and also update `config.json`

```
{
  "commands": {
    "test": {
      "desc": "test command description",
      "module": "geomdl-test.test",
      "func": "test_function",
      "func_args": 1
    }
  }
}
```

Now, our command expects 1 argument and prints it. In the following example the input argument is *hey* and *testing_input*:

```
$ geomdl-cli test hey
my test function prints hey

$ geomdl-cli test testing_input
my test function prints testing_input
```

If we omit the input, we will see a warning message:

```
$ geomdl-cli test
TEST expects 1 argument(s). Please run 'geomdl-cli test --help' for command help.
```

Let's update our command to add a help text. Update `test.py` as follows:

```
def test_function(test_input, **kwargs):
    """\
TEST: Prints input arguments.

It would be good idea to put more details here...\
    """
    print("my test function prints", str(test_input))
```

and then type **geomdl-cli test –help**.

```
$ geomdl-cli test --help
TEST: Prints input arguments.

It would be good idea to put more details here...
```

We have successfully created a very simple command for geomdl-cli tool.

4.3 Overriding commands

Overriding commands is a very simple task. You might need to extend or change the functionality of an existing command, then overriding would be a simple option. Let's update `config.json` as follows:

```
{
  "commands": {
    "export": {
      "desc": "test command description",
      "module": "geomdl-test.test",
      "func": "test_function",
      "func_args": 1
    }
  }
}
```

Please note that we changed the command name from **test** to **export** and we expect to see the output of **test** command when we run **geomdl-cli export**. Let's first test the change with **geomdl-cli help**:

```
$ geomdl-cli help
GEOMDL-CLI - Run NURBS-Python (geomdl) from the command line

geomdl-cli is a command line tool for 'geomdl', a pure Python NURBS and B-Spline␣
↪library.

Usage:

    geomdl-cli {command} {options}

Individual command help available via

    geomdl-cli {command} --help

Available commands:

    help                displays the help message
    version             displays the package version
```

(continues on next page)

(continued from previous page)

```
    config              displays the configuration
    plot                plots single or multiple NURBS curves and surfaces using␣
↪matplotlib
    eval                evaluates NURBS shapes and exports the evaluated points in␣
↪various formats
    export              test command description
```

Have you noticed the change in **export** command's description text? Let's try it again with one of our previous examples:

```
$ geomdl-cli export testing_input
my test function prints testing_input
```

We have successfully overridden an existing geomdl-cli command.

4.4 Creating configuration variables

A configuration variable can be used to store default values for your custom command. Custom configuration variables are also defined in `config.json` file:

```
{
  "configuration": {
    "test_configuration": "default configuration text"
  },
  "commands": {
    "test": {
      "desc": "test command description",
      "module": "geomdl-test.test",
      "func": "test_function",
    }
  }
}
```

Let's update `test.py` file as follows:

```
from geomdl.cli import config

def test_function(**kwargs):
    print("The value of the config variable is '" + config['test_configuration'] + "'
↪")
```

Have you noticed that **test_configuration** is a key defined under **configuration** in `config.json`? Then, let's test the command output:

```
$ geomdl-cli test
The value of the config variable is 'default configuration text'
```

Additionally, you can find the list of active configuration variables by typing **geomdl-cli config**.

```
$ geomdl-cli config
Configuration variables:
- user_override: True
- plot_vis: legend:off
- plot_name: None
```

(continues on next page)

(continued from previous page)

```
- eval_format: screen
- export_format: json
- test_configuration: default configuration text
```

You can check commands.py file for examples on using configuration variables.

4.5 Overriding configuration variables

Overriding configuration variables is very similar to overriding commands. For instance, **plot_vis** is a configuration variable used by **plot** command. It defines the visualization configuration, such as displaying and hiding figure elements, like legend, axes, control points grid/polygon. The default configuration can be displayed by running **geomdl-cli config** command:

```
$ geomdl-cli config
Configuration variables:
- user_override: False
- plot_vis: legend:off
- plot_name: None
- eval_format: screen
- export_format: json
```

Let's update `config.json` file as follows and override the value of **plot_vis** configuration variable:

```
{
  "configuration": {
    "plot_vis": "legend:on;ctrlpts:off"
  }
}
```

To verify the change, we can run **geomdl-cli config** command:

```
$ geomdl-cli config
Configuration variables:
- user_override: True
- plot_vis: legend:on;ctrlpts:off
- plot_name: None
- eval_format: screen
- export_format: json
```

If geomdl-cli loads the configuration variables from `config.json` file, the value of **user_override** variable changes to `True`.

CHAPTER 5

File Formats and Templating

Unless defined otherwise in the command help (`geomdl-cli {command} --help`), any command in need of a NURBS data input usse the following file formats: libconfig, YAML and JSON. Please see geomdl documentation for details on the supported file formats.

5.1 Using Jinja2 templates in input files

The following YAML file describes a 3-dimensional Bézier curve:

```
{% set degree = 3 %}
{% set kv = knot_vector(3, 4) %}

shape:
  type: curve
  data:
    degree: {{ degree }}
    knotvector: {{ kv }}
    control_points:
      points:
        - [10, 5, 10]
        - [10, 20, -30]
        - [40, 10, 25]
        - [-10, 5, 0]
```

The tags `{%` and `%}` define Jinja2 template statements, `{{` and `}}` define expressions to print to the template output. `{% set degree = 3 %}` simply creates a template variable **degree** which can be used to replace integer `3`. This variable is used to set the curve degree with `{{ degree }}`.

`knot_vector` is a wrapper for utilities.generate_knot_vector function. `{% set kv = knot_vector(3, 4) %}` sets the output of the template function to the template variable **kv**.

The following is the list of custom template functions supported by geomdl-cli:

- `knot_vector(d, np)`: generates a uniform knot vector. *d*: degree, *np*: number of control points

- `sqrt(x)`: square root of x
- `cubert(x)`: cube root of x
- `pow(x, y)`: x to the power of y