abqcy

Release 0.0.2

WANG Hailin

May 17, 2023

CONTENTS

Write Abaqus Subroutines in Cython.

- GitHub repository: https://github.com/haiiliin/abqcy
- PyPI: https://pypi.org/project/abqcy
- Documentation: https://abqcy.readthedocs.io
- Read the Docs: https://readthedocs.org/projects/abqcy
- Bug report: https://github.com/haiiliin/abqcy/issues

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Getting Started

abqcy allows you to write your Abaqus subroutines in Cython. It provides a command line tool to compile your Cython code into an object file (.obj) that can be used by Abaqus.

1.1.1 Installation

You can install abqcy with pip:

```
pip install abqcy
```

or install it from source:

```
pip install git+https://github.com/haiiliin/abqcy
```

1.1.2 Environment Setup

abqcy requires a working Abaqus installation with user subroutines enabled. Make sure the abaqus command is available in the command line, otherwise you need to create a new system environment variable ABAQUS_BAT_PATH and set it to the path of the abaqus.bat file.

abqcy uses Cython to compile your Cython code into a C source file (.c). In order to compile the C source file into an object file (.obj) that can be used by Abaqus, the abaqus make command is used (it uses the MSVC cl compiler from Visual Studio). Since the compiled .c file requires the Python headers and libraries, you need to make sure that the cl compiler can find them. This can be done by setting the INCLUDE and LIB environment variables. If you do not want to set global environment variables, you can also create a .env file in the directory where you run the abqcy command.

The following is the information of the INCLUDEenvironment variable on my computer, you need to separate the paths with ; on Windows and : on Linux:

```
C:/Users/Hailin/AppData/Local/Programs/Python/Python310/include
C:/Users/Hailin/AppData/Local/Programs/Python/Python310/Lib/site-packages/numpy/core/
↪include
C:/Program Files (x86)/Microsoft Visual Studio/2019/BuildTools/VC/Tools/MSVC/14.29.30133/
↪include
C:/Program Files (x86)/Windows Kits/10/Include/10.0.19041.0/shared
C:/Program Files (x86)/Windows Kits/10/Include/10.0.19041.0/ucrt
```

and the following is the information of the LIB environment variable on my computer:

```
C:/Users/Hailin/AppData/Local/Programs/Python/Python310/libs
C:/Users/Hailin/AppData/Local/Programs/Python/Python310/Lib/site-packages/numpy/core/lib
C:/Program Files (x86)/Windows Kits/10/Lib/10.0.19041.0/um/x64
C:/Program Files (x86)/Windows Kits/10/Lib/10.0.19041.0/ucrt/x64
```

1.1.3 Usage

You can now write your Abaqus subroutine in Cython, simple scripts can be found in the examples directory.

After you have written your subroutine, you can compile it with the abqcy command:

```
abqcy compile <path-to-your-subroutine>
```

This will compile your subroutine into a C source file (.c) and a C header file (.h), and then they will be compiled into an object file (.obj) that can be used by Abaqus. These files are in the same directory as your subroutine.

Now you can use the subroutine in Abaqus, like:

```
abaqus job=Job-1 input=model.inp user=your-subroutine.obj
```

1.2 Command Line Interface

The abqcy command line is used to compile you Cython code into an object (.obj) file that can be used by Abaqus. You can use it in the command line or in a Python script with the *abqcy.cli.abqcy* object (an *abqcy.cli.AbqcyCLI* object).

1.2.1 References

The abqcy command

```
$ abqcy
NAME
    abqcy - The ``abqcy`` command-line interface.

SYNOPSIS
    abqcy COMMAND

DESCRIPTION
    The ``abqcy`` command-line interface.

COMMANDS
    COMMAND is one of the following:

     compile
       Compile a Cython script to an Abaqus user subroutine as an object file.

     run
       Run Abaqus jobs.
```

The `abqcy compile` command

```
$ abqcy compile --help
INFO: Showing help with the command 'abqcy compile -- --help'.

NAME
    abqcy compile - Compile a Cython script to an Abaqus user subroutine as an object␣
↪file.

SYNOPSIS
    abqcy compile SCRIPT <flags>

DESCRIPTION
    Compile a Cython script to an Abaqus user subroutine as an object file.

POSITIONAL ARGUMENTS
    SCRIPT
        Type: 'str'
        The path to the Cython script to compile.

FLAGS
    --exclude=EXCLUDE
        Type: Optional['list']
        Default: None
        When passing glob patterns as ``script``, you can exclude certain module names␣
↪explicitly by passing them into the ``exclude`` option.
    -n, --nthreads=NTHREADS
        Type: 'int'
        Default: 0
        The number of concurrent builds for parallel compilation (requires the␣
↪``multiprocessing`` module).
    --aliases=ALIASES
        Type: Optional['dict']
        Default: None
        If you want to use compiler directives like ``# distutils: ...`` but can only␣
↪know at compile time (when running the ``setup.py``) which values to use, you can use␣
↪aliases and pass a dictionary mapping those aliases
    -q, --quiet=QUIET
        Type: 'bool'
        Default: False
        If True, Cython won't print error, warning, or status messages during the␣
↪compilation.
    -f, --force=FORCE
        Type: 'bool'
        Default: False
        Forces the recompilation of the Cython modules, even if the timestamps don't␣
↪indicate that a recompilation is necessary.
    -l, --language=LANGUAGE
        Type: Optional['str']
        Default: None
        To globally enable C++ mode, you can pass ``language='c++'``. Otherwise, this␣
↪will be determined at a per-file level based on compiler directives.  This affects␣
↪only modules found based on file names.  Extension instances passed
```

(continues on next page)

(continued from previous page)

```
    --exclude_failures=EXCLUDE_FAILURES
        Type: 'bool'
        Default: False
        For a broad 'try to compile' mode that ignores compilation failures and simply␣
↪excludes the failed extensions, pass ``exclude_failures=True``. Note that this only␣
↪really makes sense for compiling ``.py`` files which can also be used without␣
↪compilation.
    --annotate=ANNOTATE
        Type: 'bool'
        Default: True
        Whether to generate an HTML file with annotations, by default True.
    Additional flags are accepted.
        Additional keyword arguments to pass to the ``cythonize`` function.

NOTES
    You can also use flags syntax for POSITIONAL ARGUMENTS
```

The `abqcy run` command

```
$ abqcy run --help
INFO: Showing help with the command 'abqcy run -- --help'.

NAME
    abqcy run - Run Abaqus jobs.

SYNOPSIS
    abqcy run INPUT USER <flags>

DESCRIPTION
    Run Abaqus jobs.

POSITIONAL ARGUMENTS
    INPUT
        Type: 'str'
        The path to the input file.
    USER
        Type: 'str'
        The name of the user subroutine, if it is a Cython/Pure Python script, it will␣
↪be compiled to an object file automatically.

FLAGS
    -j, --job=JOB
        Type: Optional['str']
        Default: None
        The name of the job, by default the current directory name.
    -o, --output=OUTPUT
        Type: Optional['str']
        Default: None
        The path to the output directory, by default the current directory.
    -s, --script=SCRIPT
```

(continues on next page)

(continued from previous page)

```
        Type: Optional['str']
        Default: None
        The Python script to run after finishing the job to post-process the results.
    Additional flags are accepted.
        Additional keyword arguments to pass to the ``abaqus`` command to make the␣
↪object file.

NOTES
    You can also use flags syntax for POSITIONAL ARGUMENTS
```

1.3 API Reference

This page contains auto-generated API reference documentation[1].

1.3.1 abqcy

Submodules

abqcy.cli

Module Contents

Classes

AbqcyCLI	The abqcy command-line interface.

Attributes

abqcy

class AbqcyCLI

The abqcy command-line interface.

compile(*script: str*, *, *exclude: list = None*, *nthreads: int = 0*, *aliases: dict = None*, *quiet: bool = False*, *force: bool = False*, *language: str = None*, *exclude_failures: bool = False*, *annotate: bool = True*, ***kwargs*)

Compile a Cython script to an Abaqus user subroutine as an object file.

Parameters

- **script** (str) – The path to the Cython script to compile.
- **exclude** (list, *optional*) – When passing glob patterns as script, you can exclude certain module names explicitly by passing them into the exclude option.

[1] Created with sphinx-autoapi

- **nthreads** (int, *optional*) – The number of concurrent builds for parallel compilation (requires the `multiprocessing` module).
- **aliases** (dict, *optional*) – If you want to use compiler directives like `# distutils: ...` but can only know at compile time (when running the `setup.py`) which values to use, you can use aliases and pass a dictionary mapping those aliases to Python strings when calling `cythonize()`. As an example, say you want to use the compiler directive `# distutils: include_dirs = ../static_libs/include/` but this path isn't always fixed and you want to find it when running the `setup.py`. You can then do `# distutils: include_dirs = MY_HEADERS`, find the value of `MY_HEADERS` in the `setup.py`, put it in a python variable called `foo` as a string, and then call `cythonize(.., aliases={'MY_HEADERS': foo})`.
- **quiet** (bool, *optional*) – If True, Cython won't print error, warning, or status messages during the compilation.
- **force** (bool, *optional*) – Forces the recompilation of the Cython modules, even if the timestamps don't indicate that a recompilation is necessary.
- **language** (str, *optional*) – To globally enable C++ mode, you can pass `language='c++'`. Otherwise, this will be determined at a per-file level based on compiler directives. This affects only modules found based on file names. Extension instances passed into `cythonize()` will not be changed. It is recommended to rather use the compiler directive `# distutils: language = c++` than this option.
- **exclude_failures** (bool, *optional*) – For a broad 'try to compile' mode that ignores compilation failures and simply excludes the failed extensions, pass `exclude_failures=True`. Note that this only really makes sense for compiling `.py` files which can also be used without compilation.
- **annotate** (bool, *optional*) – Whether to generate an HTML file with annotations, by default True.
- **kwargs** – Additional keyword arguments to pass to the `cythonize` function.

run(*input: str*, *user: str*, ***, *job: str = None*, *output: str = None*, *script: str = None*, ***kwargs*)

Run Abaqus jobs.

Parameters

- **input** (str) – The path to the input file.
- **user** (str) – The name of the user subroutine, if it is a Cython/Pure Python script, it will be compiled to an object file automatically.
- **job** (str, *optional*) – The name of the job, by default the current directory name.
- **output** (str, *optional*) – The path to the output directory, by default the current directory.
- **script** (str, *optional*) – The Python script to run after finishing the job to post-process the results.
- **kwargs** – Additional keyword arguments to pass to the `abaqus` command to make the object file.

abqcy

abqcy.version

Module Contents

Functions

_get_version()	Return the version string used for __version__.

Attributes

_default_version
__version__

_default_version = '0.0.0'

_get_version()

Return the version string used for __version__.

__version__

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

C

M

R