abqcy

Release 0.0.4

WANG Hailin

May 18, 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

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). Since the compiled `.c` file requires the Python headers and libraries, abqcy will try to find them automatically and update the `INCLUDE` and `LIB` environment variables. If it fails to find them, you need to update the `INCLUDE` and `LIB` environment variables manually.

1.1.3 Usage

You can now write your Abaqus subroutine in Cython, simple scripts can be found in *Examples*.

Note: In order to not mess up with the Cython declarations, you can add a companion `.pxd` file with the same name as your Cython `.py` or `.pyx` file, and put the Cython declarations in it. If you are not comfortable with keeping two files, you can just use the `.pyx` file with the Cython declarations.

See *Examples* for detailed examples.

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 Examples

Below are some examples of how to use the library. To compile the examples into an object file (`.obj`) that can be used by Abaqus, you can run the following command:

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

1.2.1 Example: Elastic `umat` subroutine

This example shows how to write an Abaqus elastic `umat` subroutine in Cython.

```cython
1   cdef extern from "<aba_for_c.h>":
2       pass
3
4
5   cdef extern void umat(
6       double *stress, double *statev, double *ddsdde, double *sse, double *spd,
7       double *scd, double *rpl, double *ddsddt, double *drplde, double *drpldt,
8       double *stran, double *dstran, double *time, double *dtime, double *temp,
9       double *dtemp, double *predef, double *dpred, char *cmname, int *ndi,
10      int *nshr, int *ntens, int *nstatv, double *props, int *nprops, double *coords,
11      double *drot, double *pnewdt, double *celent, double *dfgrd0, double *dfgrd1,
12      int *noel, int *npt, int *layer, int *kspt, int *jstep, int *kinc,
13  ):
14      cdef double E, nu, lam, G
15      E, nu = props[0], props[1]
16      lam = E * nu / ((1.0 + nu) * (1.0 - 2.0 * nu))
17      G = E / (2.0 * (1.0 + nu))
18
19      cdef int i, j
20      for i in range(3):
21          for j in range(3):
22              ddsdde[6 * i + j] = lam
23          ddsdde[6 * i + i] += 2.0 * G
24          ddsdde[6 * (i + 3) + (i + 3)] = G
25      for i in range(6):
26          for j in range(6):
27              stress[i] += ddsdde[6 * i + j] * dstran[j]
```

```cython
1   import cython
2
```

(continued from previous page)

```python
3
4  def umat(
5      stress, statev, ddsdde, sse, spd, scd, rpl, ddsddt, drplde, drpldt, stran, dstran,
6      time, dtime, temp, dtemp, predef, dpred, cmname, ndi, nshr, ntens, nstatv, props,
7      nprops, coords, drot, pnewdt, celent, dfgrd0, dfgrd1, noel, npt, layer, kspt,
8      jstep, kinc,
9  ):  # fmt: skip
10     cython.declare(E=cython.double, nu=cython.double, lam=cython.double, G=cython.double)
11     E, nu = props[0], props[1]
12     lam = E * nu / ((1.0 + nu) * (1.0 - 2.0 * nu))
13     G = E / (2.0 * (1.0 + nu))
14
15     cython.declare(i=cython.int, j=cython.int)
16     for i in range(3):
17         for j in range(3):
18             ddsdde[6 * i + j] = lam
19         ddsdde[6 * i + i] += 2.0 * G
20         ddsdde[6 * (i + 3) + (i + 3)] = G
21     for i in range(6):
22         for j in range(6):
23             stress[i] += ddsdde[6 * i + j] * dstran[j]
```

Note: You will need to add the Cython header file (`.pxd`) along with the Python file (`.py`) in order to use the Cython declarations.

```python
1  cdef extern from "<aba_for_c.h>":
2      pass
3
4
5  cdef extern void umat(
6      double *stress, double *statev, double *ddsdde, double *sse, double *spd,
7      double *scd, double *rpl, double *ddsddt, double *drplde, double *drpldt,
8      double *stran, double *dstran, double *time, double *dtime, double *temp,
9      double *dtemp, double *predef, double *dpred, char *cmname, int *ndi,
10     int *nshr, int *ntens, int *nstatv, double *props, int *nprops, double *coords,
11     double *drot, double *pnewdt, double *celent, double *dfgrd0, double *dfgrd1,
12     int *noel, int *npt, int *layer, int *kspt, int *jstep, int *kinc,
13 )
```

1.3 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).

1.3.1 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
    --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 MODEL USER <flags>

DESCRIPTION
    Run Abaqus jobs.

POSITIONAL ARGUMENTS
    MODEL
        Type: 'str'
        The path to the input file or a Python script to create 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. If a companion ``.pxd`` file is found, it
→will be copied to the output directory along with the Cython/Pure Python script.

FLAGS
    -j, --job=JOB
        Type: Optional['str']
        Default: None
        The name of the job, by default the model name without the extension.
    -o, --output=OUTPUT
        Type: Optional['str']
        Default: None
        The path to the output directory, by default the current directory.
    -p, --post=POST
        Type: Optional['str']
        Default: None
        The Python script to run after finishing the job to post-process the results. In
→the output script, a placeholder ``{odb}`` will be replaced with the path to the
→output database file.
    -v, --visualization=VISUALIZATION
        Type: Optional['str']
        Default: None
        The Python script to run after finishing the job to visualize the results.
→Typically, this script will plot a figure based on the data saved by the post-
→processing script.
    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
```

1.4 API Reference

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

1.4.1 abqcy

Submodules

`abqcy.cli`

Module Contents

Classes

AbqcyCLI	The abqcy command-line interface.

Attributes

abqcy

class AbqcyCLI

    The abqcy command-line interface.

    _update_include_lib()

        Update the INCLUDE and LIB environment variables.

    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.

            • **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

---

[1] Created with sphinx-autoapi

> # 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(*model: str, user: str, *, job: str = None, output: str = None, post: str = None, visualization: str = None, **kwargs*)

Run Abaqus jobs.

> **Parameters**
>
> - **model** (str) – The path to the input file or a Python script to create 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. If a companion `.pxd` file is found, it will be copied to the output directory along with the Cython/Pure Python script.
>
> - **job** (str, *optional*) – The name of the job, by default the model name without the extension.
>
> - **output** (str, *optional*) – The path to the output directory, by default the current directory.
>
> - **post** (str, *optional*) – The Python script to run after finishing the job to post-process the results. In the output script, a placeholder {odb} will be replaced with the path to the output database file.
>
> - **visualization** (str, *optional*) – The Python script to run after finishing the job to visualize the results. Typically, this script will plot a figure based on the data saved by the post-processing script.
>
> - **kwargs** – Additional keyword arguments to pass to the `abaqus` command to make the object file.

abqcy

abqcy.subs

Module Contents

```
STANDARD = ['CREEP', 'DFLOW', 'DFLUX', 'DISP', 'DLOAD', 'FILM', 'FLOW', 'FRIC',
'FRIC_COEF', 'GAPCON',...
```

```
EXPLICIT = ['VDFLUX', 'VDISP', 'VDLOAD', 'VEXTERNALDB', 'VFABRIC', 'VFRIC', 'VFRIC_COEF',
'VFRICTION',...
```

subs

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__

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

Symbols

A

C

E

M

R

S