無料電子ブック

学習

cython

Free unaffiliated eBook created from
Stack Overflow contributors.

#cython

1
1: cython 2
2
Cython 2
2
2
Examples 2
Cython 2
1Cython 2
2
UbuntuDebian 3
Windows 3
2C 3
UbuntuDebian 3
4
Windows 4
4
4
hello.pyx 4
test.py 4
setup.py 5
5
2: Cython 6
Examples 6
pyinstallerCython 6
Windows 6
Numpy 7
3: C 8
Examples 8
C 8
8

test_extern.pxd 8
test_extern.pyx 8
4: C ++ **9**
Examples 9
DLLCythonPythonC ++ 9
C ++ DLL complexFunLib.hcomplexFunLib.cpp 9
Cython ccomplexFunLib.pxdcomplexFunLib.pyx 10
Python setup.pyrun.py 11
..... **14**

1: cythonをいめる

Cythonとはですか

Cythonプログラミングは、Cのようなけ、Cをびすことができる、およびそののいくつかのによってPythonをします。これにより、Pythonのようなをしながら Cレベルのパフォーマンスにできます。

どのようにするのですか

Cythonコードはcython source-to-sourceコンパイラをしてコンパイルされ、CまたはC ++コードをします。これはCコンパイラをしてコンパイルできます。これにより、Pythonやファイルからインポートできるをすることができます。

なPythonとはに、Cythonができるなパフォーマンスのは、CPython APIをバイパスすることによってもたらされます。たとえば、2つのをする、Pythonはのチェックをし、つかったをたすaddをつけし、そのをびします。 Cythonでされ たCコードでは、はにかっていて、びすは1つだけです。したがって、Cythonはに、がなにきます。

コードをするためにどのようにしますか

なは、Cythonをしてプログラムをスピードアップしようとするときに、コードをプロファイルし、なをコンパイルみのCythonモジュールにすることです。これにより、のコードにしてPythonをし、もなでをすることができます。

Examples

Cythonのインストール

Cythonをするために、2つのものがまneeded.The Cythonパッケージ、ある`cython`えばnumpyのためのいくつかのCとPythonライブラリにソース-ソースコンパイラとCythonインターフェイスを。`cython`コンパイラによってされたCコードをコンパイルするには、Cコンパイラがです。

ステップ1Cythonのインストール

システムアジノス

Cythonはいくつかのシステムにしないパッケージシステムとにインストールできます。これらには、

1. pipまたはeasy_installをしてPyPI

```
$ pip install cython
$ easy_install cython
```

2. アナコンダをしてconda

```
$ conda install cython
```

3. enpkgパッケージマネージャをしたEnthought Canopy

```
$ enpkg cython
```

また、ソースコードはgithubからダウンロードして、をしてでインストールできます。

```
$ python setup.py install
```

Ubuntu、Debian

Ubuntuでは、 `cython`と`cython3`のパッケージをできます。これらは、のインストールオプションよりいバージョンをすることにしてください。

```
$ apt-get install cython cython3
```

Windows

Windowsの、pipをしてインストールできる.whlファイルはサードパーティによってされます。 .whlファイルのWindowsへのインストールにするは、 こちらをしてください 。

2Cコンパイラのインストール

CythonでされたCファイルをコンパイルするには、CおよびC ++のコンパイラがです。 gccコンパイラがされ、のようにインストールできます。

Ubuntu、Debian

`build-essential`パッケージには、なものがすべてまれています。リポジトリからインストールするには、のコマンドをします。

```
$ sudo apt-get install build-essential
```

マック

XCodeツールには、gccのようなコンパイラがまれています 。

Windows

MinGW Minimalist GNU for WindowsにはWindowsのgccがまれています。 Visual Studioのコンパイラもできます。

こんにちは

PythonからPythonファイルをするには、CythonのpyxファイルをCコード *cythonized* にしてコンパイルするがあります。なアプローチは、モジュールをしてPythonプログラムにインポートすることです。

コード

このでは、3つのファイルをします。

- `hello.pyx`にはCythonコードがまれています。
- `test.py`はhelloをするPythonスクリプトです。
- `setup.py`はCythonコードをコンパイルするためにわれます。

hello.pyx

```
from libc.math cimport pow

cdef double square_and_add (double x):
    """Compute x^2 + x as double.

    This is a cdef function that can be called from within
    a Cython program, but not from Python.
    """
    return pow(x, 2.0) + x

cpdef print_result (double x):
    """This is a cpdef function that can be called from Python."""
    print("({} ^ 2) + {} = {}".format(x, x, square_and_add(x)))
```

test.py

```
# Import the extension module hello.
import hello

# Call the print_result method
hello.print_result(23.0)
```

setup.py

```
from distutils.core import Extension, setup
from Cython.Build import cythonize

# define an extension that will be cythonized and compiled
ext = Extension(name="hello", sources=["hello.pyx"])
setup(ext_modules=cythonize(ext))
```

コンパイル

これは、 `cython hello.pyx`をしてコードをCにし、 `gcc`をしてコンパイルすることによってできます。よりなは、distutilsにこれをさせることです

```
$ ls
hello.pyx  setup.py  test.py
$ python setup.py build_ext --inplace
$ ls
build  hello.c  hello.cpython-34m.so  hello.pyx  setup.py  test.py
```

オブジェクト.soファイルをPythonからインポートしてすることができるので、 `test.py`は`test.py`をすることができます

```
$ python test.py
(23.0 ^ 2) + 23.0 = 552.0
```

オンラインでcythonをいめるをむ https://riptutorial.com/ja/cython/topic/2925/cythonをいめる

2: Cythonバンドリング

Examples

pyinstallerをったCythonプログラムのバンドル

エントリポイントをつCythonプログラムからする

```
def do_stuff():
    cdef int a,b,c
    a = 1
    b = 2
    c = 3
    print("Hello World!")
    print([a,b,c])
    input("Press Enter to continue.")
```

`setup.py`ファイルをじフォルダにします

```
from distutils.core import setup
from Cython.Build import cythonize
setup(
    name = "Hello World",
    ext_modules = cythonize('program.pyx'),
)
```

`python setup.py build_ext --inplace`をすると、 `.pyd`ライブラリがサブフォルダにされます。

その、ライブラリえば、 `main.py` をってバニラのPythonスクリプトをし、 `.pyd`ファイルをそのに`.pyd`ます

```
import program
program.do_stuff()
```

pyinstallerをしてpyinstallerをバンドルし`pyinstaller --onefile "main.py"`。 これにより、ライブラリとPythonランタイムをむ4 MB +サイズのファイルをむサブフォルダがされます。

ビルドのWindows

Windowsでのをするには、のの`.bat`をします。

```
del "main.exe"
python setup.py build_ext --inplace
del "*.c"
rmdir /s /q ".\build"
pyinstaller --onefile "main.py"
copy /y ".\dist\main.exe" ".\main.exe"
rmdir /s /q ".\dist"
rmdir /s /q ".\build"
```

```
del "*.spec"
del "*.pyd"
```

バンドルにNumpyをする

バンドルにNumpyをするには、 `setup.py`を`include_dirs`キーワードでし、にじてnumpyをラッパーPythonスクリプトにインポートして、Pyinstallerにします。

`program.pyx`

```
import numpy as np
cimport numpy as np


def do_stuff():
    print("Hello World!")
    cdef int n
    n = 2
    r = np.random.randint(1,5)
    print("A random number: "+str(r))
    print("A random number multiplied by 2 (made by cdef):"+str(r*n))
    input("Press Enter to continue.")
```

`setup.py`

```
from distutils.core import setup, Extension
from Cython.Build import cythonize
import numpy

setup(
    ext_modules=cythonize("hello.pyx"),
    include_dirs=[numpy.get_include()]
)
```

`main.py`

```
import program
import numpy
program.do_stuff()
```

オンラインでCythonバンドリングをむ https://riptutorial.com/ja/cython/topic/6386/cythonバンドリング

3: Cコードをりす

Examples

カスタムCライブラリのの

カスタムからをする`my_random`というCライブラリがあります。 `set_seed(long seed)`と`rand()` そしてもっとくのはありませんの2つのをいたいといます。 Cythonでそれらをするには、

1. .pxdファイルにインターフェイスをし、
2. .pyxファイルでをびします。

コード

test_extern.pxd

```
# extern blocks define interfaces for Cython to C code
cdef extern from "my_random.h":
    double rand()
    void c_set_seed "set_seed" (long seed) # rename C version of set_seed to c_set_seed to
avoid naming conflict
```

test_extern.pyx

```
def set_seed (long seed):
    """Pass the seed on to the c version of set_seed in my_random."""
    c_set_seed(seed)

cpdef get_successes (int x, double threshold):
    """Create a list with x results of rand <= threshold

    Use the custom rand function from my_random.
    """
    cdef:
        list successes = []
        int i
    for i in range(x):
        if rand() <= threshold:
            successes.append(True)
        else:
            successes.append(False)
    return successes
```

オンラインでCコードをりすをむ https://riptutorial.com/ja/cython/topic/3626/cコードをりす

4: ラッピングC ++

Examples

DLLのラッピングCythonからPythonへのC ++

これは、C ++ dllをCythonでラップするをしています。それはのなステップをカバーします

- Visual StudioをしてC ++でサンプルDLLをします。
- CythonでDLLをラップすると、Pythonでびすことができます。

Cythonがインストールされていて、それを**Python**でにインポートできるとしています。

DLLのについては、Visual StudioでDLLをすることにしていることもとしています。

なには、のファイルのがまれます。

1. `complexFunLib.h` C ++ DLLソースのヘッダーファイル
2. `complexFunLib.cpp` C ++ DLLソースのCPPファイル
3. `ccomplexFunLib.pxd` Cythonの "ヘッダー"ファイル
4. `complexFunLib.pyx` Cythonの "ラッパー"ファイル
5. `setup.py complexFunLib.pyd`をするためのPythonセットアップファイル
6. `run.py` コンパイルみのCythonラップDLLをインポートするPythonファイルの

C ++ DLLソース `complexFunLib.h`および`complexFunLib.cpp`

すでに**DLL**とヘッダソースファイルがあるは、これをスキップしてください。に、Visual StudioをしてDLLをコンパイルするC ++ソースをします。この、なをしてをいたいとえています。の2つのは、が`res`される`k`と`eek k*exp(ee)`のをします。とのにする2つのがあります。これらのサンプルはOpenMPをしているので、OpenMPがプロジェクトのVisual Studioオプションでになっていることをしてください。

Hファイル

```
// Avoids C++ name mangling with extern "C"
#define EXTERN_DLL_EXPORT extern "C" __declspec(dllexport)
#include <complex>
#include <stdlib.h>

// Handles 64 bit complex numbers, i.e. two 32 bit (4 byte) floating point numbers
EXTERN_DLL_EXPORT void mp_mlt_exp_c4(std::complex<float>* k,
                                     std::complex<float>* ee,
                                     int sz,
                                     std::complex<float>* res,
                                     int threads);
```

```
// Handles 128 bit complex numbers, i.e. two 64 bit (8 byte) floating point numbers
EXTERN_DLL_EXPORT void mp_mlt_exp_c8(std::complex<double>* k,
std::complex<double>* ee,
                                     int sz,
                                     std::complex<double>* res,
                                     int threads);
```

CPPファイル

```
#include "stdafx.h"
#include <stdio.h>
#include <omp.h>
#include "complexFunLib.h"

void mp_mlt_exp_c4(std::complex<float>* k,
                   std::complex<float>* ee,
                   int sz,
                   std::complex<float>* res,
                   int threads)
{
    // Use Open MP parallel directive for multiprocessing
    #pragma omp parallel num_threads(threads)
    {
        #pragma omp for
        for (int i = 0; i < sz; i++) res[i] = k[i] * exp(ee[i]);
    }
}

void mp_mlt_exp_c8(std::complex<double>* k,
                   std::complex<double>* ee,
                   int sz, std::complex<double>* res,
                   int threads)
{
    // Use Open MP parallel directive for multiprocessing
    #pragma omp parallel num_threads(threads)
    {
        #pragma omp for
        for (int i = 0; i < sz; i++) res[i] = k[i] * exp(ee[i]);
    }
}
```

Cythonソース **ccomplexFunLib.pxd**および**complexFunLib.pyx**

に、C ++ DLLをラップするのになCythonソースファイルをします。このステップでは、のをします。

- あなたはCythonをインストールしました
- でしたような、のDLLをしている

なは、Pythonモジュールとしてインポートして C ++でかれたをする`.pyd`ファイルをコンパイルするために、これらのCythonソースファイルをのDLL `.pyd`することです。

PXDファイル

このファイルは、C ++ヘッダーファイルにしています。ほとんどの、さなCythonのでこのファイルにヘッダーをコピー・ペーストすることができます。この、のCythonがされました。`ccomplexFunLib.pxd`のに`c`をすることにしてください。これはではありませんが、このようながのにつことがわかりました。

```
cdef extern from "complexFunLib.h":
    void mp_mlt_exp_c4(float complex* k, float complex* ee, int sz,
                       float complex* res, int threads);
    void mp_mlt_exp_c8(double complex* k, double complex* ee, int sz,
                       double complex* res, int threads);
```

PYXファイル

このファイルは、C ++の`cpp`ソースファイルにしています。このでは、Numpy `ndarray`オブジェクトへのポインタをインポートDLLに`ndarray`ます。みみのCython `memoryview`オブジェクトをにすることもできますが、そのパフォーマンスは`ndarray`オブジェクトほどくないかもしれませんただし、はかなりクリーンです。

```
cimport ccomplexFunLib  # Import the pxd "header"
# Note for Numpy imports, the C import most come AFTER the Python import
import numpy as np  # Import the Python Numpy
cimport numpy as np  # Import the C Numpy

# Import some functionality from Python and the C stdlib
from cpython.pycapsule cimport *

# Python wrapper functions.
# Note that types can be delcared in the signature

def mp_exp_c4(np.ndarray[np.complex64_t, ndim=1] k,
              np.ndarray[np.complex64_t, ndim=1] ee,
              int sz,
              np.ndarray[np.complex64_t, ndim=1] res,
              int threads):
    '''
    TODO: Python docstring
    '''
    # Call the imported DLL functions on the parameters.
    # Notice that we are passing a pointer to the first element in each array
    ccomplexFunLib.mp_mlt_exp_c4(&k[0], &ee[0], sz, &res[0], threads)

def mp_exp_c8(np.ndarray[np.complex128_t, ndim=1] k,
              np.ndarray[np.complex128_t, ndim=1] ee,
              int sz,
              np.ndarray[np.complex128_t, ndim=1] res,
              int threads):
    '''
    TODO: Python docstring
    '''
    ccomplexFunLib.mp_mlt_exp_c8(&k[0], &ee[0], sz, &res[0], threads)
```

Python ソース `setup.py` と `run.py`

setup.py

このファイルは、Cythonのコンパイルをするpythonファイルです。そのは、コンパイルされた `.pyd`ファイルをして、Pythonモジューールによってインポートすることです。このでは、 `setup.py` とじディレクトリになすべてのファイル `complexFunLib.h` 、 `complexFunLib.dll` 、 `ccomplexFunLib.pxd` 、および`complexFunLib.pyx` を`complexFunLib.pyx`しています。

このファイルがされたら、コマンドラインからパラメーータきでするがあります `build_ext --inplace`

このファイルがされると、エラーをさせることなく`.pyd`ファイルがされます。によっては、いがあると、 `.pyd`がされるがありますがであることにしてください。された`.pyd`をするに、 `setup.py` のにエラーがスローされていないことをしてください。

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
import numpy as np

ext_modules = [
    Extension('complexFunLib',
              ['complexFunLib.pyx'],
              # Note here that the C++ language was specified
              # The default language is C
              language="c++",
              libraries=['complexFunLib'],
              library_dirs=['.'])
    ]

setup(
    name = 'complexFunLib',
    cmdclass = {'build_ext': build_ext},
    ext_modules = ext_modules,
    include_dirs=[np.get_include()]  # This gets all the required Numpy core files
)
```

run.py

`complexFunLib` Pythonモジューールにインポートし、ラップされたDLLをびすことができます。

```
import complexFunLib
import numpy as np

# Create arrays of non-trivial complex numbers to be exponentiated,
# i.e. res = k*exp(ee)
k = np.ones(int(2.5e5), dtype='complex64')*1.1234 + np.complex64(1.1234j)
ee = np.ones(int(2.5e5), dtype='complex64')*1.1234 + np.complex64(1.1234j)
sz = k.size  # Get size integer
res = np.zeros(int(2.5e5), dtype='complex64')  # Create array for results

# Call function
complexFunLib.mp_exp_c4(k, ee, sz, res, 8)

# Print results
print(res)
```

オンラインでラッピングC ++をむ https://riptutorial.com/ja/cython/topic/3525/ラッピングc-

plusplus

クレジット

S. No		Contributors
1	cythonをいめる	Community, J.J. Hakala, Keith L, m00am
2	Cythonバンドリング	Andrii Magalich
3	Cコードをりす	m00am
4	ラッピングC ++	J.J. Hakala, Keith L, Kevin Pasquarella