

Cython

Naoya Kawakami

October 5, 2022

Contents

Chapter 1	Accelerate python code with Cython	1
1.1	Introduction	1
1.2	Python code	1
1.3	Using Cython	3
1.4	Type defenition	4
1.5	Cython with cimport	6
1.6	Comparison of each code	7
Chapter 2	Codes	8
2.1	Python	8
2.2	Just use cythone	12
2.3	Type defined cython code	13
2.4	C import cython code	17

Chapter 1

Accelerate python code with Cython

1.1 Introduction

Python is one of the most widely used computing languages because of its simple coding and various useful libraries. However, the execution speed of Python is slow compared with other languages. The slow execution speed can be problematic when performing a long-time simulation with many cycles of iterations. One of the solutions to improve the speed of Python is to use Cython. In Cython, the Python code is automatically translated into C (or C++), resulting in a faster speed, while Cython still shares similar grammar as Python. In this brief review, I will introduce the process of converting Python code into Cython with a simple example. All the codes are shown in Chapter 2..

1.2 Python code

Firstly, I prepared codes for simulating "random walk." In this program, a particle randomly moves around in a square lattice under the periodic boundary condition. The position of the particle is recorded at each step, and the number of occupations at each lattice site is output as a color map. I coded the program a little complicated way, while the simulation itself is very simple, to sufficiently explain how to convert Python to Cython. Here, I just briefly introduce each code. If you need a detailed explanation of the coding of Python, refer to the other textbook.

1.2.1 Python_program.py

Python_program.py (2.1) is used to start the calculation. This code will not be changed anymore (even after converting to Cython).

1.2.2 main_functions.py

The components for conducting the simulation are defined in the class "functions" in main_function.py (2.2). The class "function" is inherited in class "random_walk" in Python_program.py.

1.2.3 sub_function.py

In sub_function.py (2.3), several additional functions are defined. The functions are imported and used in main_functions.py.

1.2.4 recording.py

A function for recording the simulation result is defined in recording.py (2.4). The defined function (record_data) is imported and used in main_functions.py.

1.2.5 InputParameter.py

A class for reading the simulation parameter is defined in InputParameter.py (2.5).

1.2.6 input.yml

This is not a Python file but a Yaml (.yml) file. Yaml form is often used to describe serialized data. Using the Yaml format, even a large number of parameters can be easily incorporated into the program as written in InputParameter.py.

"cell_size_xy" defines the number of lattice sites in one direction. "probabilities" defines the probability of hopping in each direction. "total_steps" determines the number of iterations. The progress of simulation is output at each "print_per" step. This file will not be changed anymore.

1.2.7 Calculation speed

It takes about 2 min 34 sec to run this program. Next, we will see how this program becomes fast by using Cython.

1.3 Using Cython

The first step in converting Python codes to Cython is to change the name of the file. Change the extension of the files from ".py" to ".pyx", except for the `python_program.py` and `input.yml`. It is not necessary to rewrite the code inside. Then, prepare a file "setup.py", as shown in 2.7. The setup.py file is used to convert the .pyx file to a C code (.c) and then make an execution file from the C code. Run the setup file by using a command

```
python setup.py build_ext -inplace
```

Several files will be formed. When running "python_program.py", the execution files are automatically used.

1.3.1 Calculation speed

Try running "python_program.py". The calculation speed is about 2 min 22 sec. So even without changing the code, the calculation speed becomes ten % faster than the normal Python code.

1.4 Type defenition

For further speed up, we should declare the type of all the variables in the Cython code. The codes are shown in Chapter 3.3. The revised and additional parts from the original Python code are highlighted in red. The following revisions are made for all the .pyx codes.

- For the class object, change "def" to "cdef".
- For the function, change "def" to "cdef" or "cpdef". The function defined by "cdef" is the fastest, but it cannot be called from a Python file. "cpdef" can be called from both Python and Cython codes. If you cannot judge how the functions are called, using "cpdef" is the safest choice.
- Describe all the types of variables used in each function. For class objects, the self variables should be declared just after defining the class. The local variables should be declared in the function. The declaration style is as follows:

```
1 cdef [type] [variable name]
```

For example, if variables "a" and "b" is used for storing integers (such as a = 1, b= 2),

```
1 cdef int a, b
```

should be written. Note that the "float" objects in Python are declared by "double" in Cython. Do not forget to declare the type of iteration variables (typically i in for loop).

- If the function returns an object, declare the type of the return just after "cdef (or cpdef)". Therefore, the function with return should be written in the format as follows:

```
1 cdef [type of return object] [function name]():
2 ...
```

If a function with the name of "test_function" returns an integer "int_a", it will be

```
1 cdef int test_function():
2 cdef int int_a
3     int_a = 1
4 return int_a
```

- If the function takes arguments, declare the type of the arguments before the parameters.

```
1 cdef [return type] [function name]([type] [parameter name]):
```

If the previous "test_function" takes two integer arguments (arg_a and arg_b), the function should be written as follows:

```
1 cdef int test_function(int arg_a, int arg_b):
2     cdef int int_a
3     int_a = arg_a + arg_b
4 return int_a
```

In addition,

```
1 # cython: language_level=3, boundscheck=False, wraparound=False
```

are added at the beginning of all the pyx files. The "language.lebel" declare that the code is written based on Python 3. The "boundscheck=False, wraparound=False" disables some functions for lists in Python. In Python code, the final object in a list can be read by list[-1], but this format will not be used if declared. Instead, this declaration contributes about ten % speed up.

1.4.1 Calculation speed

After the above revisions, do not forget to run "setup.py" again. The calculation speed is 36 sec, four times faster than the original Python code.

1.5 Cython with cimport

We will change the import method of the functions. By using "cimport" instead of "import", the program can know the type of arguments and returns of the imported functions in advance, which contribute to the faster calculation time. For using "cimport", we should prepare ".pxd" files for all the ".pxy" files. In the .pxd files, all the types of self variables (for class) and functions are declared. The functions declared in .pxd can be imported by the "cimport" method. See Code 2.12 - 2.17 for the example of the .pxd files.

For including the pxd file in translation, the setup.py file should be revised. See Code 2.18.

1.5.1 Calculation speed

After the revision by running setup.py, we can try the calculation. The calculation time is 29 sec, which is five times faster than the original Python code.

1.6 Comparison of each code

Table 1.1 Calculation time

Method	time	/Python
Python	2:34	1
Cython	2:22	0.92
Cython (Type defined)	0:36	0.23
Cython (cimport)	0:29	0.19

Finally, Cython code achieved five times faster speed compared with normal Python. Further improvement might be possible by replacing Python functions with corresponding C (or C++) functions, but it requires significant modifications to the code because some of the useful functions in Python will not be available. Considering the balance between the benefit of speed up and time for revision, the `c import` Cython would be the efficient choice.

Chapter 2

Codes

The change of the code are highlighted by red. The additional comments (not necessary in code) are highlighted by blue.

2.1 Python

Code 2.1 Python_ program.py

```

1 from main_function import functions
2
3 class random_walk(functions):
4     def __init__(self):
5         functions.__init__(self)
6
7     def end(self):
8         print("Recording")
9         self.end_of_loop()
10
11    def start(self):
12        print("Calculation start")
13        self.start_setting()
14        print("Loop start")
15        self.loop()
16        self.end()
17
18 if __name__ == "__main__":
19     rw_class = random_walk()
20     rw_class.start()

```

Code 2.2 main_function.py

```

1 from InputParameter import Params
2 import time
3 from sub_functions import pos_rec_lattice, choice_event
4 from recording import record_data
5 import os
6 import math
7
8 class functions:
9     def __init__(self):
10        self.init_value = Params("input.yml")
11        self.cell_size = self.init_value.cell_size_xy
12        self.count = 0
13        self.output_count = self.init_value.print_per

```

```

14         self.dir_rec = [0, 0, 0, 0, 0]
15
16     def start_setting(self):
17         self.direction = [0, 0]
18         self.current_position = [int(self.cell_size / 2), int(self.
19             cell_size / 2)]
19         if os.path.exists("Record") is False:
20             os.mkdir("Record")
21         self.pos_rec = pos_rec_lattice(self.cell_size)
22         self.probability_set()
23         self.start_time = time.time()
24
25     def probability_set(self):
26         self.up = self.init_value.proBABILITIES["up"]
27         self.right = self.init_value.proBABILITIES["right"]
28         self.down = self.init_value.proBABILITIES["down"]
29         self.left = self.init_value.proBABILITIES["left"]
30         total = self.up + self.down + self.left + self.right
31         self.up = self.up / total
32         self.left = self.left / total + self.up
33         self.down = self.down / total + self.left
34         self.right = self.right / total + self.down
35
36     def loop(self):
37         for _ in range(self.init_value.total_steps):
38             self.function_loop()
39
40     def function_loop(self):
41         self.direction = choice_event(self.up, self.left, self.down)
42         self.direction_record()
43         self.update_events()
44         self.record_position()
45         self.count += 1
46         if self.count == self.output_count:
47             percentage = self.count / self.init_value.total_steps * 100
48             print(f"Progress: {percentage:.2f} %")
49             self.output_count += self.init_value.print_per
50
51     def direction_record(self):
52         if self.direction == [0, -1]:
53             self.dir_rec[0] += 1
54         elif self.direction == [-1, 0]:
55             self.dir_rec[1] += 1
56         elif self.direction == [0, 1]:
57             self.dir_rec[2] += 1
58         elif self.direction == [1, 0]:
59             self.dir_rec[3] += 1
60         else:
61             self.dir_rec[4] += 1
62
63     def update_events(self):
64         self.current_position[0] = self.boundary_check(
65             self.current_position[0] + self.direction[0]
66         )
67         self.current_position[1] = self.boundary_check(
68             self.current_position[1] + self.direction[1]

```

```

69         )
70
71     def boundary_check(self, val):
72         if val < 0:
73             return self.cell_size - 1
74         elif val >= self.cell_size:
75             return 0
76         else:
77             return val
78
79     def record_position(self):
80         self.pos_rec[self.current_position[0]][self.current_position[1]]
            += 1
81
82     def end_of_loop(self):
83         self.elapsed_time = time.time() - self.start_time
84         self.minute = math.floor(self.elapsed_time / 60)
85         self.second = int(self.elapsed_time % 60)
86         record_data(self.pos_rec, self.minute, self.second)
87         print(self.dir_rec)
88         print("Finished: " + str(self.minute) + " min " + str(self.
            second) + " sec")

```

Code 2.3 sub.function.py

```

1  import random
2
3  def pos_rec_lattice(cell_size):
4      pos_rec = [[0 for _ in range(cell_size)] for _ in range(cell_size)]
5      return pos_rec
6
7  def choice_event(up, left, down):
8      random_val = random.random()
9      if random_val <= up:
10         return [0, -1]
11     elif random_val <= left:
12         return [-1, 0]
13     elif random_val <= down:
14         return [0, 1]
15     else:
16         return [1, 0]

```

Code 2.4 recording.py

```

1  import seaborn as sns
2  import matplotlib.pyplot as plt
3
4  def record_data(pos_rec, minute, second):
5      plt.figure()
6      sns.heatmap(pos_rec)
7      plt.savefig("Record/heat_map.png")
8      #
9      file_data = open("Record/execution_time.txt", "a")
10     file_data.write(str(minute) + " min " + str(second) + " sec" "\n")
11     file_data.close()

```

Code 2.5 InputParameter.py

```
1 import yaml
2
3 class Params:
4     def __init__(self, filename):
5         if filename:
6             with open(filename) as f:
7                 input_yaml = yaml.safe_load(f)
8                 for k, v in input_yaml.items():
9                     setattr(self, k, v)
```

Code 2.6 input.yml

```
1 cell_size_xy: 100
2 probabilities:
3     up: 1.0
4     left: 1.0
5     down: 1.0
6     right: 1.0
7 total_steps: 100000000
8 print_per: 10000000
```

2.2 Just use cythone

Code 2.7 setup.py

```
1 from setuptools import setup
2 from Cython.Build import cythonize
3 import Cython.Compiler.Options
4
5 Cython.Compiler.Options.annotate = True
6
7 setup(ext_modules=cythonize("sub_functions.pyx"), annotate=True)
8 setup(ext_modules=cythonize("recording.pyx"), annotate=True)
9 setup(ext_modules=cythonize("InputParameter.pyx"), annotate=True)
10 setup(ext_modules=cythonize("main_function.pyx"), annotate=True)
```

2.3 Type defined cython code

Code 2.8 main_function.pyx

```

1  # cython: language_level=3, boundscheck=False, wraparound=False
2
3  from InputParameter import Params
4  import time
5  from sub_functions import pos_rec_lattice, choice_event
6  from recording import record_data
7  import os
8  import math
9
10
11  cdef class functions:
12      cdef int cell_size, count, output_count, minute, second
13      cdef double start_time, up, left, down, right, elapsed_time
14      cdef list dir_rec
15      cdef list direction
16      cdef list current_position
17      cdef list pos_rec
18
19
20  def __init__(self):
21      self.init_value = Params("input.yml")
22      self.cell_size = self.init_value.cell_size_xy
23      self.count = 0
24      self.output_count = self.init_value.print_per
25      self.dir_rec = [0, 0, 0, 0, 0]
26
27  cpdef start_setting(self):
28      self.direction = [0, 0]
29      self.current_position = [int(self.cell_size / 2), int(self.
30          cell_size / 2)]
31      if os.path.exists("Record") is False:
32          os.mkdir("Record")
33      self.pos_rec = pos_rec_lattice(self.cell_size)
34      self.probability_set()
35      self.start_time = time.time()
36
37  cdef probability_set(self):
38      cdef double total
39      self.up = self.init_value.probabilities["up"]
40      self.right = self.init_value.probabilities["right"]
41      self.down = self.init_value.probabilities["down"]
42      self.left = self.init_value.probabilities["left"]
43      total = self.up + self.down + self.left + self.right
44      self.up = self.up / total
45      self.left = self.left / total + self.up
46      self.down = self.down / total + self.left
47      self.right = self.right / total + self.down
48
49  cpdef loop(self):
50      cdef int _
51      for _ in range(self.init_value.total_steps):

```

```

51         self.function_loop()
52
53     cdef function_loop(self):
54         cdef double percentage
55         self.direction = choice_event(self.up, self.left, self.down)
56         self.direction_record()
57         self.update_events()
58         self.record_position()
59         self.count += 1
60         if self.count == self.output_count:
61             percentage = self.count / self.init_value.total_steps * 100
62             print(f"Progress: {percentage:.2f} %")
63             self.output_count += self.init_value.print_per
64
65     cdef direction_record(self):
66         if self.direction == [0, -1]:
67             self.dir_rec[0] += 1
68         elif self.direction == [-1, 0]:
69             self.dir_rec[1] += 1
70         elif self.direction == [0, 1]:
71             self.dir_rec[2] += 1
72         elif self.direction == [1, 0]:
73             self.dir_rec[3] += 1
74         else:
75             self.dir_rec[4] += 1
76
77     cdef update_events(self):
78         self.current_position[0] = self.boundary_check(
79             self.current_position[0] + self.direction[0]
80         )
81         self.current_position[1] = self.boundary_check(
82             self.current_position[1] + self.direction[1]
83         )
84
85     cdef int boundary_check(self, int val):
86         if val < 0:
87             return self.cell_size - 1
88         elif val >= self.cell_size:
89             return 0
90         else:
91             return val
92
93     cdef record_position(self):
94         self.pos_rec[self.current_position[0]][self.current_position[1]]
95         += 1
96
97     cpdef end_of_loop(self):
98         self.elapsed_time = time.time() - self.start_time
99         self.minute = math.floor(self.elapsed_time / 60)
100        self.second = int(self.elapsed_time % 60)
101        record_data(self.pos_rec, self.minute, self.second)
102        print(self.dir_rec)
103        print("Finished: " + str(self.minute) + " min " + str(self.
104            second) + " sec")

```


Code 2.9 sub_function.pyx

```

1  # cython: language_level=3, boundscheck=False, wraparound=False
2
3  import random
4
5  cdef list pos_rec_lattice(int cell_size):
6      cdef list pos_rec
7      pos_rec = [[0 for _ in range(cell_size)] for _ in range(cell_size)]
8      return pos_rec
9
10
11 cdef list choice_event(double up, double left, double down):
12     cdef double random_val
13     random_val = random.random()
14     if random_val <= up:
15         return [0, -1]
16     elif random_val <= left:
17         return [-1, 0]
18     elif random_val <= down:
19         return [0, 1]
20     else:
21         return [1, 0]

```

Code 2.10 recording.pyx

```

1  # cython: language_level=3, boundscheck=False, wraparound=False
2
3  import seaborn as sns
4  import matplotlib.pyplot as plt
5
6
7  cdef record_data(list pos_rec, int minute, int second):
8      plt.figure()
9      sns.heatmap(pos_rec)
10     plt.savefig("Record/heat_map.png")
11     #
12     file_data = open("Record/execution_time.txt", "a")
13     file_data.write(str(minute) + " min " + str(second) + " sec" "\n")
14     file_data.close()

```

Code 2.11 recording.pyx

```

1  # cython: language_level=3, boundscheck=False, wraparound=False
2
3  import yaml
4
5  cdef class Params:
6      cdef public int cell_size_xy
7      cdef public dict probabilities
8      cdef public int total_steps
9      cdef public int print_per
10     def __init__(self, filename):
11         if filename:
12             with open(filename) as f:
13                 input_yaml = yaml.safe_load(f)

```

```
14         for k, v in input_yaml.items():
15             setattr(self, k, v)
```

2.4 C import cython code

Code 2.12 main functions.pyx

```

1  # cython: language_level=3, boundscheck=False, wraparound=False
2
3  from InputParameter cimport Params
4  import time
5  from sub_functions cimport pos_rec_lattice, choice_event
6  from recording cimport record_data
7  import os
8  import math
9
10
11 cdef class functions:
12     # Type declareion of self variables are moved to pxd file
13     def __init__(self):
14         .....

```

Code 2.13 main functions.pxd

```

1  # cython: language_level=3, boundscheck=False, wraparound=False
2
3  from InputParameter cimport Params
4
5  cdef class functions:
6      # Declaretion of self variables, which are moved from pyx file.
7      cdef Params init_value
8      # In pxd file, user defined class can be used to type assignment.
9      # Here, type "Params" is defined as the type of "init_value".
10     cdef int cell_size, count, output_count, minute, second
11     cdef double start_time, up, left, down, right, elapsed_time
12     cdef list dir_rec
13     cdef list direction
14     cdef list current_position
15     cdef list pos_rec
16     #
17     # Declaretion of functions. These should be consistent with those in
18     # pyx file.
19     cpdef start_setting(self)
20     cdef probability_set(self)
21     cpdef loop(self)
22     cdef function_loop(self)
23     cdef direction_record(self)
24     cdef update_events(self)
25     cdef int boundary_check(self, int val)
26     cdef record_position(self)
27     cpdef end_of_loop(self)

```

Code 2.14 sub functions.pxd

```

1  # cython: language_level=3, boundscheck=False, wraparound=False
2
3  cdef list pos_rec_lattice(int cell_size)

```

```
4 cdef list choice_event(double up, double left, double down)
```

Code 2.15 recording.pxd

```
1
2 # cython: language_level=3, boundscheck=False, wraparound=False
3
4 cdef record_data(list pos_rec, int minute, int second)
```

Code 2.16 InputParameter.pyx

```
1 # cython: language_level=3, boundscheck=False, wraparound=False
2
3 import yaml
4
5 cdef class Params:
6     def __init__(self, filename):
7         if filename:
8             with open(filename) as f:
9                 input_yaml = yaml.safe_load(f)
10                for k, v in input_yaml.items():
11                    setattr(self, k, v)
```

Code 2.17 InputParameter.pxd

```
1 # cython: language_level=3, boundscheck=False, wraparound=False
2
3 cdef class Params:
4     cdef public int cell_size_xy
5     cdef public dict probabilities
6     cdef public int total_steps
7     cdef public int print_per
```

Code 2.18 InputParameter.pxd

```
1 from setuptools import setup
2 from Cython.Build import cythonize
3 import Cython.Compiler.Options
4 from distutils.extension import Extension
5
6 Cython.Compiler.Options.annotate = True
7
8 ext_modules = [
9     Extension("InputParameter", ["InputParameter.pyx"]),
10 ]
11 setup(ext_modules=cythonize(ext_modules, annotate=True))
12 #
13 ext_modules = [
14     Extension("recording", ["recording.pyx"]),
15 ]
16 setup(ext_modules=cythonize(ext_modules, annotate=True))
17 #
18 ext_modules = [
19     Extension("sub_functions", ["sub_functions.pyx"]),
20 ]
```

```
21 setup(ext_modules=cythonize(ext_modules, annotate=True))
22 #
23 ext_modules = [
24     Extension("main_function", ["main_function.pyx"]),
25 ]
26 setup(ext_modules=cythonize(ext_modules, annotate=True))
27 #
```