Python Optimization and Language Binding

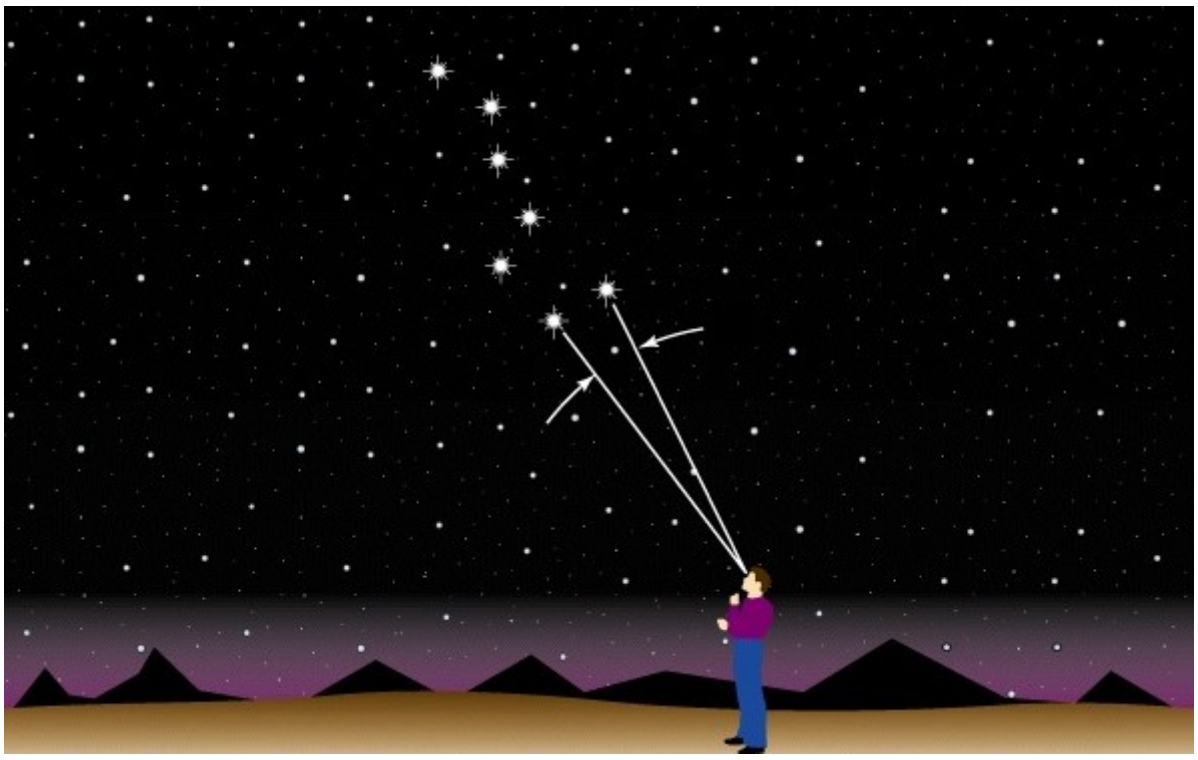
Python receives a lot of well deserved praise. Python code is clean and readable, and it's easy to learn. There's a vast library of 3rd-party packages providing support for thousands of tasks. It's well supported, and has become the language of choice for many specializations, like machine learning and analytics.

One area where Python is sometimes criticized is performance. Being an interpreted language, it's easy to understand where this concern comes from. For most tasks, though, pure Python performs quite well, and in areas where it does need a boost, there are optimization tricks that can be applied. We'll discuss a few of those.

Fully-implemented, runnable versions of all of the code mentioned in this article is available in my GitHub repo here.

A Task To Perform

The task we'll implement will be to calculate the distance between two celestial objects. Our starting code will be a very small subset of another one of my projects: an implementation of many astronomical algorithms from the "Practical Astronomy with your Calculator or Spreadsheet" book. If you're interested in learning more, you can find that project here. For the sake of this article, we'll take the existing implementation of the `angle_between_two_objects()` method from the project.



Before we start working with the code, let's talk a bit about what the angle_between_two_objects()

method does. The method takes in a set of arguments describing the right ascension/declination coordinates for two different celestial objects (like stars), and then calculates the angular distance between them in degrees, minutes, and seconds. Right ascension and declination describe the position of an object in the night sky in much the same way that latitude and longitude is used to describe the location for something on the surface of the Earth. If you want to learn more about sky coordinates, I've put together a more detailed explanation here.

Let's get started!

Setup

This is written with the assumption that you'll be working in a Linux environment. Adapting it to Windows or Mac shouldn't be difficult, though.

You'll need Python 3, which should already be installed on your system.

If you want to work with Python code in a Jupyter notebook, the easiest way to do it is with Visual Studio Code, with the Python extension installed. Visual Studio Code is also the friendliest way to work with straight Python code, in my opinion.

Pure Python (no optimizations)

We'll start with a straightforward Python implementation, with no optimizations. First, we need a couple of imports:

```
import math
import random
```

Then, some supporting methods:

```
# Convert Civil Time (hours,minutes,seconds) to Decimal Hours
def hms_dh(hours,minutes,seconds):
    A = abs(seconds) / 60
    B = (abs(minutes) + A) / 60
    C = abs(hours) + B

    return -C if ((hours < 0) or (minutes < 0) or (seconds < 0)) else C

# Convert Degree-Hours to Decimal Degrees
def dh_dd(degree_hours):
    return degree_hours * 15

# Convert Degrees Minutes Seconds to Decimal Degrees
def dms_dd(degrees,minutes,seconds):
    A = abs(seconds) / 60
    B = (abs(minutes) + A) / 60
```

```
    C = abs(degrees) + B

    return -C if degrees < 0 or minutes < 0 or seconds < 0 else C

# Convert W value to Degrees
def degrees(W):
    return W * 57.29577951

# Extract degrees, minutes, and seconds from decimal degrees
def dd_deg(decimal_degrees):
    """ Return Degrees part of Decimal Degrees """
    A = abs(decimal_degrees)
    B = A * 3600
    C = round(B - 60 * math.floor(B / 60),2)
    D = 0 if C == 60 else C
    E = B = 60 if C == 60 else B

    return -math.floor(E/3600) if decimal_degrees < 0 else
math.floor(E/3600)

def dd_min(decimal_degrees):
    """ Return Minutes part of Decimal Degrees """
    A = abs(decimal_degrees)
    B = A * 3600
    C = round(B - 60 * math.floor(B / 60),2)
    D = 0 if C == 60 else C
    E = B + 60 if C == 60 else B

    return math.floor(E/60) % 60

def dd_sec(decimal_degrees):
    """ Return Seconds part of Decimal Degrees """
    A = abs(decimal_degrees)
    B = A * 3600
    C = round(B - 60 * math.floor(B / 60),2)
    D = 0 if C == 60 else C

    return D
```

Our method to calculate the angle:

```
# Calculate the angle between two celestial objects
def
angle_between_two_objects(ra_long_1_hour_deg,ra_long_1_min,ra_long_1_sec,dec
_lat_1_deg,dec_lat_1_min,dec_lat_1_sec,ra_long_2_hour_deg,ra_long_2_min,ra_l
ong_2_sec,dec_lat_2_deg,dec_lat_2_min,dec_lat_2_sec,hour_or_degree):
    ra_long_1_decimal =
hms_dh(ra_long_1_hour_deg,ra_long_1_min,ra_long_1_sec) if hour_or_degree ==
"H" else dms_dd(ra_long_1_hour_deg,ra_long_1_min,ra_long_1_sec)
    ra_long_1_deg = dh_dd(ra_long_1_decimal) if hour_or_degree == "H" else
ra_long_1_decimal
```

```
    ra_long_1_rad = math.radians(ra_long_1_deg)
    dec_lat_1_deg1 = dms_dd(dec_lat_1_deg,dec_lat_1_min,dec_lat_1_sec)
    dec_lat_1_rad = math.radians(dec_lat_1_deg1)

    ra_long_2_decimal =
hms_dh(ra_long_2_hour_deg,ra_long_2_min,ra_long_2_sec) if hour_or_degree ==
"H" else dms_dd(ra_long_2_hour_deg,ra_long_2_min,ra_long_2_sec)
    ra_long_2_deg = dh_dd(ra_long_2_decimal) if hour_or_degree == "H" else
ra_long_2_decimal
    ra_long_2_rad = math.radians(ra_long_2_deg)
    dec_lat_2_deg1 = dms_dd(dec_lat_2_deg,dec_lat_2_min,dec_lat_2_sec)
    dec_lat_2_rad = math.radians(dec_lat_2_deg1)

    cos_d = math.sin(dec_lat_1_rad) * math.sin(dec_lat_2_rad) +
math.cos(dec_lat_1_rad) * math.cos(dec_lat_2_rad) * math.cos(ra_long_1_rad -
ra_long_2_rad)
    d_rad = math.acos(cos_d)
    d_deg = degrees(d_rad)

    angle_deg = dd_deg(d_deg)
    angle_min = dd_min(d_deg)
    angle_sec = dd_sec(d_deg)

    return angle_deg,angle_min,angle_sec
```

Some code to make a single call to the method, to make sure everything's working correctly:

```
# First object is at right ascension 5 hours 13 minutes 31.7 seconds,
declination -8 degrees 13 minutes 30 seconds

# Second object is at right ascension 6 hours 44 minutes 13.4 seconds,
declination -16 degrees 41 minutes 11 seconds

angle_deg,angle_min,angle_sec = angle_between_two_objects(5, 13, 31.7, -8,
13, 30, 6, 44, 13.4, -16, 41, 11, "H")

# Result (should be 23 degrees, 40 minutes, 25.86 seconds)
print(f"Result is {angle_deg} degrees, {angle_min} minutes, {angle_sec}
seconds.")
```

And finally, multiple calls to the method, so that we can get a better idea of how it's performing. (This code uses %time, available in Jupyter, to time the call.)

```
# Multiple Test Runs (timed)
def exec_tests():
    for test_iter in range(1,1000):
        right_ascension_hour = random.randrange(1,12)

        angle_deg,angle_min,angle_sec =
```

```
angle_between_two_objects(right_ascension_hour, 13, 31.7, -8, 13, 30, 6, 44,
13.4, -16, 41, 11, "H")

%time exec_tests()
```

Timing results:

```
CPU times: user 6.41 ms, sys: 235 µs, total: 6.64 ms
Wall time: 6.62 ms
```

Now we have working code calculating angular distance between objects, and we've established a baseline for how long it takes to run using pure Python. With that, we'll move on to optimization.

Numba

Numba is a JIT compiler that translates Python into machine code. It doesn't support the entire language, but it's quite handy for optimizing subsets of code. It's also (usually) very easy to implement.

Numba is an external package, so you'll need to install it:

```
pip3 install numba
```

Add an import for Numba:

```
import math
import numba  # new!
import random
```

Then, you mark individual methods for compilation with a simple decorator, like this:

```
@numba.jit
def hms_dh(hours,minutes,seconds):
    A = abs(seconds) / 60
    B = (abs(minutes) + A) / 60
    C = abs(hours) + B

    return -C if ((hours < 0) or (minutes < 0) or (seconds < 0)) else C
```

The decorator supports additional arguments for things like parallelization, but we'll keep it simple.

I added the numba decorator to each of the support methods, and also to the angle_between_two_objects() method. Running the same timed test gave me the following results:

```
CPU times: user 3.78 ms, sys: 0 ns, total: 3.78 ms
Wall time: 3.74 ms
```

With no optimization beyond adding a simple decorator, our processing time is cut almost in half. Not bad at all!

But, what if you have highly optimized code in a language like C, with performance that you just can't match in Python? Or, optimization aside, what if that C code is complex, mature, and well-tested, and you'd prefer to leverage it directly, instead of dealing with the effort of porting it?

There are several options available for this as well. We'll explore a couple: **ctypes** and **CFFI**.

Language Bindings

As we prepare to implement language bindings in Python, our first step is to create something we can bind *to*. We'll use C, and we'll start with a header file:

abo_lib.h

```
#ifndef abo_lib
#define abo_lib
#define M_PI 3.14159265358979323846264338327

struct angle {
  double angleDegrees, angleMinutes, angleSeconds;
};

typedef struct angle TAngle;

TAngle AngleBetweenTwoObjects(double raLong1HourDeg, double raLong1Min,
                              double raLong1Sec, double decLat1Deg,
                              double decLat1Min, double decLat1Sec,
                              double raLong2HourDeg, double raLong2Min,
                              double raLong2Sec, double decLat2Deg,
                              double decLat2Min, double decLat2Sec,
                              char hourOrDegree);
#endif
```

Next, we'll implement our methods:

abo_lib.c

```
#include "abo_lib.h"
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

/**
 * Round input to specified number of decimal places.
 */
double Round(double input, int places) {
```

```
  bool isNegative = (input < 0) ? true : false;

  long double multiplier = pow(10, places);

  if (isNegative) {
    input = fabs(input);
  };

  long double a = input * multiplier;
  a = (a >= 0) ? a + 0.5 : a - 0.5;

  double returnValue =
      floor(a) / multiplier; // floor() gives good results for more
places (7+)
                             // than the original (int) cast.

  return (isNegative) ? -(returnValue) : returnValue;
}

/**
 * Convert Civil Time (hours,minutes,seconds) to Decimal Hours
 */
double HmsToDh(double hours, double minutes, double seconds) {
  double fHours = hours;
  double fMinutes = minutes;
  double fSeconds = seconds;

  double a = fabs(fSeconds) / 60;
  double b = (fabs(fMinutes) + a) / 60;
  double c = fabs(fHours) + b;

  return (fHours < 0 || fMinutes < 0 || fSeconds < 0) ? -c : c;
}

/**
 * Convert Degrees Minutes Seconds to Decimal Degrees
 */
double DegreesMinutesSecondsToDecimalDegrees(double degrees, double
minutes,
                                             double seconds) {
  double a = fabs(seconds) / 60;
  double b = (fabs(minutes) + a) / 60;
  double c = fabs(degrees) + b;

  return (degrees < 0 || minutes < 0 || seconds < 0) ? -c : c;
}

/**
 * Convert Degree-Hours to Decimal Degrees
 */
double DegreeHoursToDecimalDegrees(double degreeHours) {
```

```
  return degreeHours * 15;
}

/**
 * Convert Degrees to Radians
 */
double DegreesToRadians(double degrees) { return (degrees * M_PI) /
180; }

/**
 * Convert W value to Degrees
 */
double WToDegrees(double w) { return w * 57.29577951; }

/**
 * Extract Degrees part of Decimal Degrees
 */
double DecimalDegreesDegrees(double decimalDegrees) {
  double a = fabs(decimalDegrees);
  double b = a * 3600;
  double c = Round(b - 60 * floor(b / 60), 2);
  double e = (c == 60) ? 60 : b;

  return (decimalDegrees < 0) ? -(floor(e / 3600)) : floor(e / 3600);
}

/**
 * Extract Minutes part of Decimal Degrees
 */
double DecimalDegreesMinutes(double decimalDegrees) {
  double a = fabs(decimalDegrees);
  double b = a * 3600;
  double c = Round(b - 60 * floor(b / 60), 2);
  double e = (c == 60) ? b + 60 : b;

  return (int)floor(e / 60) % 60;
}

/**
 * Extract Seconds part of Decimal Degrees
 */
double DecimalDegreesSeconds(double decimalDegrees) {
  double a = fabs(decimalDegrees);
  double b = a * 3600;
  double c = Round(b - 60 * floor(b / 60), 2);
  double d = (c == 60) ? 0 : c;

  return d;
}
```

```
/**
 * Calculate the angle between two celestial objects, in
 * degrees,minutes,seconds.
 */
TAngle AngleBetweenTwoObjects(double raLong1HourDeg, double raLong1Min,
                              double raLong1Sec, double decLat1Deg,
                              double decLat1Min, double decLat1Sec,
                              double raLong2HourDeg, double raLong2Min,
                              double raLong2Sec, double decLat2Deg,
                              double decLat2Min, double decLat2Sec,
                              char hourOrDegree) {

  TAngle returnValue;

  double raLong1Decimal = (hourOrDegree == 'H')
                              ? HmsToDh(raLong1HourDeg, raLong1Min,
raLong1Sec)
                              : DegreesMinutesSecondsToDecimalDegrees(
                                    raLong1HourDeg, raLong1Min,
raLong1Sec);

  double raLong1Deg = (hourOrDegree == 'H')
                          ? DegreeHoursToDecimalDegrees(raLong1Decimal)
                          : raLong1Decimal;

  double raLong1Rad = DegreesToRadians(raLong1Deg);
  double decLat1Deg1 =
      DegreesMinutesSecondsToDecimalDegrees(decLat1Deg, decLat1Min,
decLat1Sec);
  double decLat1Rad = DegreesToRadians(decLat1Deg1);

  double raLong2Decimal = (hourOrDegree == 'H')
                              ? HmsToDh(raLong2HourDeg, raLong2Min,
raLong2Sec)
                              : DegreesMinutesSecondsToDecimalDegrees(
                                    raLong2HourDeg, raLong2Min,
raLong2Sec);
  double raLong2Deg = (hourOrDegree == 'H')
                          ? DegreeHoursToDecimalDegrees(raLong2Decimal)
                          : raLong2Decimal;
  double raLong2Rad = DegreesToRadians(raLong2Deg);
  double decLat2Deg1 =
      DegreesMinutesSecondsToDecimalDegrees(decLat2Deg, decLat2Min,
decLat2Sec);
  double decLat2Rad = DegreesToRadians(decLat2Deg1);

  double cosD =
      sin(decLat1Rad) * sin(decLat2Rad) +
      cos(decLat1Rad) * cos(decLat2Rad) * cos(raLong1Rad - raLong2Rad);
  double dRad = acos(cosD);
```

```
  double dDeg = WToDegrees(dRad);

  double angleDeg = DecimalDegreesDegrees(dDeg);
  double angleMin = DecimalDegreesMinutes(dDeg);
  double angleSec = DecimalDegreesSeconds(dDeg);

  returnValue.angleDegrees = angleDeg;
  returnValue.angleMinutes = angleMin;
  returnValue.angleSeconds = angleSec;

  return returnValue;
}
```

Then create a main() module, so we can test it as a pure C implementation first:

abo_client.c

```
/**
 * Test client for the abo_lib library.
 */

#include "abo_lib.h"
#include <stdio.h>

int main() {
  TAngle angle = AngleBetweenTwoObjects(5, 13, 31.7, -8, 13, 30, 6, 44,
13.4,
                                        -16, 41, 11, 'H');

  printf("The result is %f degrees %f minutes %f seconds.\n",
         angle.angleDegrees, angle.angleMinutes, angle.angleSeconds);

  return (0);
}
```

Build it:

```
gcc -c abo_client.c

gcc -c abo_lib.c

gcc -o abo_client abo_client.o abo_lib.o -lm
```

Run it:

```
./abo_client
```

Result:

```
The result is 23.000000 degrees 40.000000 minutes 25.860000 seconds.
```

Now that we know our C implementation is working, we can move on to accessing it from Python.

ctypes

The ctypes library provides access to foreign functions from Python. It has the advantage of maturity and availability. (It's a part of the standard Python library. No need to install anything extra.)

Before we start to write our Python code, we need to build a shared library from our C implementation. Open a terminal, and run the following:

```
gcc -I. abo_lib.c -shared -o abo_lib.so
```

Then, you can write your Python code. First, import ctypes:

```
import ctypes as ct
```

Create a structure to hold our return value, and describe it for ctypes:

```
class TAngle(ct.Structure):
    _fields_ = [
        ("angleDegrees", ct.c_double),
        ("angleMinutes", ct.c_double),
        ("angleSeconds", ct.c_double)
    ]
```

Load the shared library:

```
libc = ct.cdll.LoadLibrary("./abo_lib.so")
```

Then, call it:

```
libc.AngleBetweenTwoObjects.argtypes = [ct.c_double, ct.c_double,
ct.c_double, ct.c_double, ct.c_double, ct.c_double, ct.c_double,
ct.c_double, ct.c_double, ct.c_double, ct.c_double, ct.c_double, ct.c_char]
libc.AngleBetweenTwoObjects.restype = TAngle

angle_between_objects = libc.AngleBetweenTwoObjects(5, 13, 31.7, -8, 13, 30,
6, 44, 13.4, -16, 41, 11, b'H')

print(f"Angle between the two objects is
{angle_between_objects.angleDegrees} degrees
{angle_between_objects.angleMinutes} minutes
{angle_between_objects.angleSeconds} seconds")
```

Result:

```
Angle between the two objects is 23.0 degrees 40.0 minutes 25.86 seconds
```

CFFI

CFFI is another foreign function library for Python. It's newer than ctypes, and it offers a couple of advantages: Python code written to access it is considered more "Pythonic" (that is, more true to Python coding standards), and it's faster.

CFFI is not part of the standard library, so you'll need to install it:

```
pip3 install cffi
```

Then, write your Python code as follows. First, import CFFI:

```
from cffi import FFI
```

Then, initialize it, and describe the C code it's accessing:

```
ffi = FFI()

ffi.cdef("""
typedef struct TAngle TAngle;
struct TAngle
{
  double angleDegrees;
  double angleMinutes;
  double angleSeconds;
};
TAngle AngleBetweenTwoObjects(double raLong1HourDeg, double raLong1Min,
double raLong1Sec, double decLat1Deg, double decLat1Min, double decLat1Sec,
double raLong2HourDeg, double raLong2Min, double raLong2Sec, double
decLat2Deg, double decLat2Min, double decLat2Sec, char hourOrDegree);
"""
)
```

Load the shared library:

```
lib = ffi.dlopen('./abo_lib.so')
```

Then, call it:

```
angle_between_objects = lib.AngleBetweenTwoObjects(5, 13, 31.7, -8, 13, 30,
6, 44, 13.4, -16, 41, 11, b'H')

print(f"Angle between the two objects is
{angle_between_objects.angleDegrees} degrees
{angle_between_objects.angleMinutes} minutes
```

```
{angle_between_objects.angleSeconds} seconds")
```

Result:

```
Angle between the two objects is 23.0 degrees 40.0 minutes 25.86 seconds
```

Wrap Up

I hope this introduction to optimization and language binding for Python was helpful. Remember, you can see complete implementations by visiting the GitHub repo here. If you encounter problems, have questions, or have requests, you can either open an issue or join one of the discussions in the repo. Feedback is always appreciated. Thanks!

python