MA453/553 – Classwork 3.

Profiling Cython Programs

Due: 10/02/2024

---

1. Download the file `classwork3.zip` from your Canvas course page. Save it in your work directory (`~/MA453/Classwork/cw3`) and unzip it. The file `particle_simulator.py` contains the original Python function `evolve()`

```
    def evolve(self, tmax, dt=0.00001):
        nsteps = int(tmax/dt)
        for i in range(nsteps):
            for p in self.particles:
                norm = (p.x**2 + p.y**2)**0.5
                v_x = (-p.y)/norm
                v_y = p.x/norm
                d_x = dt * p.ang_speed * v_x
                d_y = dt * p.ang_speed * v_y
                p.x += d_x
                p.y += d_y
```

and its improvements: `evolve_python()`, `evolve_numpy()` and `evolve_cython()`. Run the code with each of these functions by commenting/uncommenting function calls in the lines 129-132 and 179-186 and make sure you get the same results (trajectories).

```
$ cd ~/MA453/Classwork/cw3
$ python setup.py build_ext --inplace
$ python particle_simulator.py
```

2. Measure the timings of the code with the pure Python, NumPy and Cython versions.

```
$ python -m timeit -s "from particle_simulator import benchmark" "benchmark(100, 'python')"
$ python -m timeit -s "from particle_simulator import benchmark" "benchmark(100, 'numpy')"
$ python -m timeit -s "from particle_simulator import benchmark" "benchmark(100, 'numexpr')"
$ python -m timeit -s "from particle_simulator import benchmark" "benchmark(100, 'cython')"
```

Which version gives the best performance?

3. Why aren't we getting the best out of Cython function? Give a look to the file `cevolve.pyx`. The cython module `cevolve` uses the untyped version. Comment lines 9-20 and uncomment lines 23-47, rebuild the module and time the `benchmark()` again.

```
#def c_evolve(r_i, ang_speed_i, dt, nsteps):
#    v_i = np.empty_like(r_i)
#   for i in range(nsteps):
#        norm_i = np.sqrt((r_i ** 2).sum(axis=1))
#        v_i = r_i[:, [1, 0]]
#       v_i[:, 0] *= -1
#        v_i /= norm_i[:, np.newaxis]
#        d_i = dt * ang_speed_i[:, np.newaxis] * v_i
#        r_i += d_i
```

```
$ python setup.py build_ext --inplace
$ python -m timeit -s "from particle_simulator import benchmark" "benchmark(100, 'cython')"
```

How much improvement did you get?

```
def c_evolve(double[:, :] r_i, double[:] ang_speed_i,double dt, int nsteps):
    cdef int i, j, nparticles = r_i.shape[0]
    cdef double norm, x, y, vx, vy, dx, dy, ang_speed
    for i in range(nsteps):
        for j in range(nparticles):
            x = r_i[j, 0]
            y = r_i[j, 1]
            ang_speed = ang_speed_i[j]
            norm = sqrt(x ** 2 + y ** 2)
            vx = (-y)/norm
            vy = x/norm
            dx = dt * ang_speed * vx
            dy = dt * ang_speed * vy
            r_i[j, 0] += dx
            r_i[j, 1] += dy
```

4. Let us profile the Cython module `cevolve.pyx` with the `annotated view` option.

```
$ cython -a cevolve.pyx
```

(This generates an HTML file `cevolve.html`)

Open the file `cevolve.html` and check which lines do have more interpreter-related calls.

```
$ firefox cevolve.html
```

The white lines corresponds translated C code, you can click these lines to see the code.

5. In the line `v_y=x/norm`, Cython checks that computed norm is not zero, otherwise it raises a `ZeroDivisorError`, and in the line `r_i[j,0]`, Cython checks if the indexes are within the bounds of the array. Add the following two lines before the function `c_evolve()` in the file `cevolve.pyx` and measure the timing of `benchmark(1000,’cython’)`.

```
@cython.boundscheck(False)
@cython.cdivision(True)
def c_evolve(double[:, :] r_i,double[:] w_i,double dt,int nsteps):
```

```
$ python setup.py build_ext --inplace
$ python -m timeit -s "from particle_simulator import benchmark" "benchmark(100, ’cython’)"
```

6. Record all the timings from **2.**, **3** and **5.** in a file `timing.txt` and send me using the `mail` command:

```
$ mail -s "ma453:cw3" 453 < timings.txt
```