Introduction to High Performance Scientific Computing

Autumn, 2017

Lecture 4

Imperial College
London

Prasun Ray

16 October, 2017

Assessment (OLD)

3 Programming assignments
HW1: Assigned 23/10, due 2/11 (**20%**)
HW2: Assigned 6/11, due 16/11 (**20%**)
HW3: Assigned 20/11, due 29/11 (**15%**)

1 Programming Project (45%)
Assigned 30/11, due 15/12

Submitting HW1 **commits you to the course**

CDT students: Will be contacted about assessment separately

Assessment (Updated)

3 Programming assignments
HW1: Assigned 23/10, due 2/11 (**20%**)
HW2: Assigned 6/11, due 16/11 (**20%**)
HW3: Assigned 20/11, due 30/11 (**20%**)

1 Programming Project (40%)
Assigned 1/12, due 15/12

Submitting HW1 **commits you to the course**

CDT students: Will be contacted about assessment separately

Importing/running code

Consider our mysqrt.py file

- **Within python terminal, can import module:**

```
In [60]: import mysqrt
```

- **To 're-import' module after making changes to code, use** importlib.reload**:**

```
In [61]: import importlib

In [62]: importlib.reload(mysqrt)
```

Importing/running code

Consider our mysqrt.py file

- **Within python terminal, can import module:**

```
In [60]: import mysqrt
```

- **To 're-import' module after making changes to code, use** importlib.reload**:**

```
In [61]: import importlib

In [62]: importlib.reload(mysqrt)
```

- **Can also use** run **command:**

```
In [63]: run mysqrt
```

- **Scripts can be executed from Unix terminal:**

```
$ python mysqrt.py
```

Timing and performance

There are a few simple approaches for getting timing information:

- **Can use** run -t **or** run -p **when running scripts in ipython terminal**
- **Can also use** timeit **command, e.g.** IN [01]: timeit mysqrt.py

Timing and performance

There are a few simple approaches for getting timing information:

- **Can use** run -t **or** run -p **when running scripts in ipython terminal**
- **Can also use** timeit **command, e.g.** IN [01]: timeit mysqrt.py
- **Set *timers* in your code using the** time **module, *e.g.***

```
#Newton's method
   t1 = time.time() #start timer 1
   tp1 = time.process_time() #start timer 2
   for i in range(imax):
      x1 = x0/2 + a/(2*x0)
      print("iteration %d, x = %16.14f" %(i+1,x1))
      del_x = abs(x1-x0)
      if del_x < tol:
         print("converged!")
         break
      x0 = x1
   t2 = time.time() #t2-t1 gives wallclock time
   tp2 = time.process_time() #tp2-tp1 gives cpu time -- depends on number of cores!
```

Random walks and Brownian motion

- **Consider equations of the form:**

  $X(t+dt) = X(t) + F[X(t),dt]$

- **Here, X and F, are both random variables**

- **Simplest example: F = +/- dx based on flip of a coin → random walk**
  - $\langle X(t+n*dt) \rangle = 0$
  - $Var\{X(t+n*dt)\} = v\ t$ **where** $v = dx^2/dt = constant$

- **If we consider particle motion, F should be a *continuous* random variable**

Numpy example: Brownian motion

- **Consider Brownian motion on a line:**

$$X(t+dt) = X(t) + \sqrt{dt}\,\mathcal{N}(0,1)$$

- **What is the best way to compute a single realization with T steps?**
- **To compute an an ensemble of M realizations?**
- **Simplest approach: two for loops**

Numpy example: Brownian motion

- **Consider Brownian motion on a line:**

$$X(t+dt) = X(t) + \sqrt{dt}\,\mathcal{N}(0,1)$$

- **What is the best way to compute a single realization with T steps?**
- **To compute an an ensemble of M realizations?**
- **Simplest approach: two for loops**
- **However, with interpreted languages, avoid for loops whenever possible!**
  - ***vectorize* your code**