# L01 - python basics

September 9, 2018

**Author: Nico Grisouard, nicolas.grisouard@physics.utoronto.ca**

*Supporting textbook chapters for week 1: 2, 3 and 4.3*

This is an example of "lecture notes". As you will quickly find out, this course is by nature a lab course. Therefore, my "lecture notes" will not often follow the linear progression of regular lecture notes. This is particularly true for this first lecture, in which I merely want to give you pointers to do the first lab.

## 1 Machine error: round-off error.

Under what circumstances is the following possible?

$$(x + y) + z \neq x + (y + z)$$

Let's try it in Python:

```
In [1]: x = 1e20
        y = -1e20
        z = 1.

In [2]: (x+y)+z

Out[2]: 1.0

In [3]: x+(y+z)

Out[3]: 0.0
```

What happened?
Round-off error!

## 2 Algorithmic error: instability

Consider this system representing phasor rotation in the complex plane:

$$\dot{Z} = i\omega Z, \quad \text{given} \quad Z_0 = Z(t = 0).$$

Solution is $Z(t) = Z_0 \exp(i\omega t)$.

How can we solve it numerically?

Taylor expansion:

$$\dot{Z}(t) = \frac{Z(t+\Delta t) - Z(t)}{\Delta t} + H.O.T. = i\omega Z(t).$$

Suggests algorithm: * Start with $Z(t=0) = Z_{old}$, * $Z_{new} = (1 + i\omega\Delta t)Z_{old}$, * repeat.
**Let's code it up and see**

In [ ]:
```
# Unstable solution to dz/dt
from math import pi

Z = complex(1.0, 0.0)  # z0 initial condition
omega = 1.0  # angular frequency 2*pi/period
dt = 2*pi/(200*omega)  # time step is 200 per period
# print initial information
print('t = {0}, Z = {1}, |Z| = {2}'.format(0.0, Z, abs(Z)))
```

In [ ]:
```
for k in range(201):  # update z
    Z *= complex(1.0, dt)  # complex(1.0, dt) is 1+i*dt

# print final information
print('t = {0}'.format(k*dt))
print('Z = {0}'.format(Z))
print('|Z| = {0}'.format(abs(Z)))
```

What happened? What is the problem? Why did it happen?
Next time!

# 3 Typical approach for solving a problem

1. Start with math model, often but not always continuous.
2. **Discretize:** set up discrete arrays of independent variables (e.g., $x$, $t$), dependent variables (e.g. $v(t)$, $a(t)$), and define operators on these variables ($dv/dt$, $ma$...).
3. **Initialize** parameters and variables appropriately.
4. **Evalutize:** run algorithm to operate on these variables.
5. **Analyze:** some extra processing of the raw results, figures...

See example:

In [ ]:
```
# Example to illustrate general procedure for mathematical modeling.
# Given position x(t) of a particle undergoing SHO,
# calculate velocity and acceleration using simple finite difference
import numpy as np  # import numpy
import matplotlib.pyplot as plt  # import figure functions
```

In [ ]:
```
# 2. Discretize

#Define time grid and dependent variables
t = np.linspace(0, 10, 101)  # time array
N = len(t)  # number of time steps
```

```
x = np.zeros(N)  # array of positions
v = np.zeros(N-1)  # array of velocities
a = np.zeros(N-2)  # array of accelerations
```

```
In [ ]: # 3. Initialize: define signal on discretized grid
        x = 3.0*np.sin(t)
```

```
In [ ]: # 2. discretize and 4. evalutize (here, apply algorithm)
        # Define velocity using finite differences: v = Delta x/Delta t
        for k in range(len(x)-1):
            v[k] = (x[k+1]-x[k])/(t[k+1]-t[k])

        # Define acceleration using finite differences: a = Delta v/Delta t
        for k in range(len(x)-2):
            a[k] = (v[k+1]-v[k])/(t[k+1]-t[k])
```

```
In [ ]: # 5. Analyze
        # print results
        print("t is ", t)
```

```
In [ ]: print("x is ", x)
```

```
In [ ]: print("a is ", a)
```

```
In [ ]: # plot results
        plt.figure()

        plt.subplot(3, 1, 1)
        plt.plot(t, x)
        plt.xlabel('t')
        plt.ylabel('x')

        plt.subplot(3, 1, 2)
        plt.plot(t[:N-1], v)
        plt.xlabel('t')
        plt.ylabel('v')

        plt.subplot(3, 1, 3)
        plt.plot(t[:N-2], a)
        plt.xlabel('t')
        plt.ylabel('a')

        # plt.savefig('T01.pdf')  # saves a pdf figure on disk
```

# 4 Pseudo-code

## 4.1 General principles

- pseudocode is the planned version of your code, written in plain English ($\neq$ programming language)

- You should write one before starting any code.
- It should describe your algorithm.
- It helps ensure that your planned logic for the algorithm is sound.
- In the previous example: the enunmeration (points 1-5) was the skeleton of one.

- Real text or comments for your real code in the end.
- **Keep a copy of it intact** so you can refer to it when you are coding.
- Coding = your pseudocode $\to$ specific programming language. Be able to take your pseudocode and convert it into any typical programming language.

- Pseudocode: somewhat personal. You do you.
- Concise, logical, step-by-step.
- Start with **brief** overview of what this piece of code will do.

Examples for sequential stuff: * Input: `READ`, `OBTAIN`, `GET` * Initialize: `SET`, `DEFINE` * Compute: `COMPUTE`, `CALCULATE`, `DETERMINE` * Add one: `INCREMENT`, `BUMP` * Output: `PRINT`, `DISPLAY`, `PLOT`, `WRITE`

Examples for conditions and loops: * `WHILE`, `IF-THEN-ELSE`, `REPEAT-UNTIL`, `CASE`, `FOR`

Should also include calling functions: * `CALL`

### 4.2 Pseudo-code, example 1

Convert polar to Cartesian coordinates from keyboard input:

$r, \theta\ (^\circ) \to x, y$.

Let's try:

1. From keyboard, read radius $r$ and save.
2. From keyboard, read angle $\theta$ in degrees and save.
3. Do the conversion from degrees to angles ($\theta_r = \pi\theta/180$).
4. Compute $(x, y) = r(\cos\theta_r, \sin\theta_r)$.
5. Print result to screen.

Alternative: write the pseudo-code code comments.

```
In [ ]: # Re-import numpy, in case I am executing cells in random order
        import numpy as np
```

```
In [ ]: # 1. From keyboard, read the radius and save.
        r = float(input("Enter r:"))
```

```
In [ ]: # 2. From keyboard, read de angle $\theta$ in degrees and save.
        theta_deg = float(input("Enter theta in degrees:"))
```

```
In [ ]: # 3. Do the conversion from degrees to angles ($\theta_r = \pi \theta/180$).
        theta_r = np.pi*theta_deg/180.
```

```
In [ ]: # 4. Compute $(x, y) = r(\cos\theta_r, \sin\theta_r)$.
        x = r * np.cos(theta_r)
        y = r * np.sin(theta_r)
```

```
In [ ]: # 5. Print result to screen.
        print("x = ", x, "y = ", y)
```