Numerical Methods for Astrophysics:

PYTHON SUMMARY

Michela Mapelli

Python. Why python?

PROS:

*** high-level language:**
written as you understand it rather than
as the computer understands it
→ simpler than low-level languages

* nearly **the best for PLOTS** (with matplotlib)

* lots of **mathematical** libraries (math, numpy, scipy..) and
libraries for **data handling** (pandas, astropy,..)

*** interpreted language:** does not need to be compiled and executed

You just need an INTERPRETER

CONS:

*** slow:** depending on application, might be 10 – 100 slower than Fortran

*** young and fast evolving:** your scripts become obsolete quickly

Python. Interpreter

1- With command line from terminal (my choice)

To interpret python, from the terminal type

python

then press enter and you are inside the python interpreter
Good scientific calculator

Or, for more complex scripts: write a script with your preferred **editor**
save it as as scriptname.py, then type

python scriptname.py

press enter and the python interpreter runs your script

Possible EDITORS: **emacs** (my choice), **gedit** (ubuntu default)

Python. Interpreter

2- User friendly interpreters? spyder

Scientific Python Development Environment, https://www.spyder-ide.org/

More than interpreter:

- editor
- graphical user interface to run scripts
  (for people who don't like the terminal)
- debugger

3- More sophisticated features? JUPYTER-NOTEBOOK

https://jupyter.org/

We will see it later during the course

Python. Variables and assignments

variables are the minimum building blocks in coding
- convey information about scalar quantities
- similar to variables in algebra but..

x = 1 is an assignment statement (in python you define and assign a variable at the same time)
different from c and c++!!! dangerous!!!

TYPES of variables:

INTEGER (examples -1, 0, 200003493094) – 32 bit

FLOAT or FLOATING POINT (1.5, 1e30, -1e20) – 64 bit

COMPLEX 1 + 2 i , but in python written 1 + 2 j

STRING (variable associated with characters)

assignment of a integer	x = 1 or x = int(1)
assignment of a float	x = 1.0 or x = float(1)
assignment of a string	x = “ciao” or x = str(ciao) or x = str(“ciao”)
	x = “123” or x = str(123) or x = str(“123”)

123 is a string if I assign it as a string!!

Python. Output and input statements

OUTPUT STATEMENT: the way the code prints some results

print(x)

Function print allows to do the output statement

INPUT STATEMENT: the way we assign the value of a variable through command line

x = input(“Enter the value of x: ”)

Function input allows to do the input statement from command line

I can specify the variable type

x = float(input(“Enter the value of x: ”))

Python. Arithmetic

ARITHMETIC OPERATORS IN PYTHON:

x+y	**addition**
x-y	**subtraction**
x*y	**multiplication**
x/y	**division**
xy**	**power**
x//y	**x divided by y and number rounded to nearest int**
x%y	**modulo of x (remainder of x after dividing by y)**

NOTE: you can do these operations also to strings
But they look much different from
arithmetic operations on numbers

```
x=”123”
y =”2”
x+y
print(x+y)
```
produces ‘1232’

Python. Arithmetic

Order of operations in python (and other languages)
~same as algebra

Multiplications and divisions before sums and subtractions
Powers before everything else
Round brakets ( ) change the order of operations

You do not have other kind of brakets

NOTE: THESE ARE ARITHMETIC ASSIGMENTS, NOT EQUATIONS!!!

```
x = 0
x = x**2 –  2
print(x)
```

If it were an equation I should solve
x2 – x – 2 = 0 which has two solutions: 2 and -1**

Instead prints gives -2

Python. Arithmetic

MODIFIERS (see c and c++):

x+=1	**equivalent to x = x + 1**
x –=2	**equivalent to x = x – 2**
x*=2.4	**equivalent to x = x * 2.4**
x/=7	**equivalent to x = x / 7**
X//=3.0	**equivalent to x = x // 3.0**

You can assign two or more variables with the same statement

```
x, y = 2.2, 3
```

Hence

```
x,y= y,x
```

means that we **swap** the values of the two variables

Python. EXERCISE

EXERCISE:

Use what you learned to calculate the distance covered in a (user provided) time t by a ball falling from a tower of (user provided) height h. Furthermore, calculate at what time t_2 the ball reaches the ground (the gravity constant $g = 9.81\ \mathrm{m\ s^{-2}}$).

Python. Packages and modules

PACKAGES: collections of useful functions and constants
which are not in the default version of python
→ you need to IMPORT them

import namepackage

For example

import math

Math contains

log	natural logarithm
log10	base-10 logarithm
exp	exponential
sin, cos, tan	sine, cosine, tangent (in radians)
asin, acos, atan	arcsine, arccosine, arctan (input in radians)

Python. Packages and modules

```
import math
A = math.log(110)
```

or

```
from math import log
A = log(110)
```

Python. Packages and modules

Some packages are so big that they contain multiple modules
Modules are sub-packages

For example numpy is a package and contains sub-packages
Example

```
import numpy as np
c = np.linalg.det(a)
```

Calculates the determinant of matrix a

Alternative forms

```
from numpy import linalg
c = linalg.det(a)
```

or

```
import numpy.linalg as linalg
c = linalg.det(a)
```

If you are interested only in det

```
from numpy.linalg import det
c = det(a)
```

Python. Containers: lists and arrays

Variables are scalar

*** but in physics/astrophysics we want VECTORS (eg position vector)**

* or we want to group together in the same structure several variables onto which we want to perform the same operation (e.g. I have 100 measurements of the same quantity and I want to calculate the mean)

DONE BY PYTHON CONTAINERS

LISTS, TUPLES, DICTIONARIES and ARRAYS

Python. Lists

LISTS in python are ordered lists of values

Each value in a list is called ELEMENT of the list

Lists can contain **elements of different types** (int,float,string,complex)

ASSIGMENT of a LIST:

```
r = [1., 15., 2., “sea”, 1e30]
```

or assign the variables first and then define the list as the container of these variables

```
x,y,z,a,b=1.,15.,2.,”sea”,1e30
r = [x,y,z,a,b]
print(r)
print(r[0])
print(r[4])
print(r[-1])
```

Python. Lists

If all elements of a list do not contain strings I can sum them

```
r = [1., 15., 2., 10.,3.]
a = sum(r)
```

I can remove elements from a list (lists can change their size!!!)

```
r.pop(1)
print(r)
```

I can insert elements inside a list (lists can change their size!!!)

```
r.insert(2,9.)
print(r)
```

I can add elements at the end of a list (lists can change their size!!!)

```
r.append(6.1)
print(r)
```

COMMON WAY TO ASSIGN A LIST IS START WITH EMPTY LIST AND THEN USE APPEND TO ASSIGN VALUES

```
r=[ ]
r.append(1.)
r.append(3.)
```

Python. Lists

WARNING: If you sum two lists you concatenate them

```
a = [1., 15.]
b = [2, 3]
c = a+b
```

c will be [1.,15.,2,3]

Python. Tuples

Similar to lists:

* can contain elements of different type

```
a=('word', 17.7, 2)
```

Note the round brackets to initialize tuples wrt square brackets for lists

* behave as lists during arithmetic operations
  i.e. a+a concatenates a to a

Different from lists:

* cannot change number of elements

Python. Arrays

LESS FLEXIBLE THAN LISTS:

1. **exist only in numpy package**
2. **the number of elements is fixed**
3. **the elements of an array must be of the same type**

GOOD REASONS TO USE ARRAYS for (astro)physics:

1. **can be two-dimensional as matrices**
2. **arrays behave like vectors and matrices in algebra**
   (no risk to concatenate while you think you are summing)
3. **arrays work faster than lists**

Python. Arrays

ASSIGNMENT OF AN ARRAY THROUGH ZEROS:

```
import numpy as np
a = np.zeros(4,float)
```

OR THROUGH A LIST:

```
b = [1.,2.]
c = np.array(b)
```

TO ASSIGN A MATRIX (m x n elements):

```
import numpy as np
a = np.zeros([2,3], float)
a[0,1] = -1.0
a[1,2] = 1.0
```

Python. Arrays

EXAMPLE OF DIFFERENCE LISTS/ARRAYS:

```
import numpy as np
a=[1.,2.]
a1=np.array(a)
b=[2.,3.]
b1=np.array(b)

c=a+b
c1=a1+b1
```

c1 is [3.,5.]
c is [1.,2.,2.,3.]

ARRAYS CAN BE SLICED:

```
import numpy as np
a=np.array([2.,3.,4.,5.,7.,9.,1.])
slice = a[1:4]
print(slice)
```

Produces 3.,4.,5.

Python. Important caveat about arrays and lists

```
x = np.zeros(4,float)
x2 = x
```

```
x = [0.,0.,0.,0.]
x2 = x
```

The assignment of a np.array x to another np.array x2 (or a list x to another list x2) does not make a copy of x into x2. Instead, the assignment statement makes x and x2 both POINT to the same address in memory.

Implication:

```
x=np.zeros(4,float)
x2=x
x[1]=1.0
print(x,x2)
```

Gives the outcome:

```
(array([ 0.,  1.,  0.,  0.]), array([ 0.,  1.,  0.,  0.]))
```

→ x2 is modified when you modify x

Python. Important caveat about arrays and lists

To make a copy of x into x2 you should use:
np.copy() for np.arrays and copy.copy() for lists

```
x=np.zeros(4,float)
x2=np.copy(x)
x[1]=1.0
print(x,x2)
```

```
import copy
x=[0.,0.,0.,0.]
x2=copy.copy(x)
x[1]=1.0
print(x,x2)
```

Give the outcomes:

```
(array([ 0.,  1.,  0.,  0.]), array([ 0.,  0.,  0.,  0.]))
```

```
([0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0])
```

→ **x2 is NOT modified when you modify x**

Python. Comments

Comments: parts of the code that are ignored by computer

Useful to understand what the program does

With python everything after a # is a comment

```
import numpy as np # import np package
a=[1.,2.]                #assign list a
a1=np.array(a)           #assign array a1
b=[2.,3.]                #assign list b
b1=np.array(b)           #assign array b1

c=a+b                    #sum a and b
c1=a1+b1                 #sum a1 and b1
```

Python. If statement

If statement used to do something only if a given condition is met

```
x=int(input("Enter an integer no greater than ten: "))
if(x>10):
     print("You entered an integer greater than ten.")
     print("Let me fix it for you.")
     x=10
print(x)
```

NOTE USAGE OF INDENTATION (very strict in python):

Operations that will be performed only if(x>10): need to be shifted to the right wrt previous lines with a TAB

Python. If statement

Examples of possible if conditions:

```
if(x==y):   checks if x is equal to y
if(x>y):    checks if x is larger than y
if(x>=y):   checks if x is larger than or equal to y
if(x<y):    checks if x is smaller than y
if(x<=y):   checks if x is smaller than or equal to y
if(x!=y):   checks if x is not equal to y
```

I can combine more conditions with the AND logical operator and/or with the OR logical operator

```
x=4
if((x>1) and (x<3)):
    print("the if statement with and gives x=", x)
if((x>1) or (x<3)):
    print("the if statement with or gives x=", x)
```

For c and c++ programmer: and instead of &&, or instead of ||

Python. while statement

While statement also checks if a condition is met

If it is met, the indented block is executed and then loops back to the beginning of the while statement

```
x=10
if(x>2):
      x-=1
      print("We are inside the if, x=", x)
print("We are out of the if, x=", x)
x=10
while(x>2):
      x-=1
      print("We are inside the while, x=", x)
print("We are out of the while, x=", x)
```

while is a statement but produces a simple loop

Python. for loops

for loop: a loop that runs through the elements of a list or array in turn

EXAMPLE 1:

```
r=[1., 3., 5.]
for i in range(len(r)): # loop over the integer i from 0 to len(r)
    print(r[i])
print("loop ended")
```

EXAMPLE 2:

```
r=[1., 3., 5.]
for i in range(1,len(r)): # loop over the integer i from 1 to len(r)
    print(r[i])
print("loop ended")
```

EXAMPLE 3:

```
r=[1., 3., 5.]
for i in range(1,len(r),2): # loop over the integer i from 1 to len(r)
    print(r[i])                                    # with steps of 2
print("loop ended")
```

Python. break statement

Allows to break out of a loop if a condition is met

EXAMPLE:

```
x=10
while(x>2):
      x-=1
      print("We are inside the while, x=", x)
      if x==5:
            break
print("We are out of the while, x=", x)
```

Useful if the loop is a very long one and I want to exit it as soon as I find the good value of x

The break statement is NESTED inside the while and the if statements

Python. continue statement

Allows to skip the rest of the indented block if a condition is met and jumps to the beginning of the loop

EXAMPLE:

```
x=10
while(x>2):
      x-=1
      print("We are inside the while, x=", x)
      if x==5:
            x-=1
            continue
      print("We are after the continue")
print("We are out of the while, x=", x)
```

Useful if the loop is a very long one and I want to exit it as soon as I find the good value of x

The continue statement is NESTED inside the while and the if statements

Python. Dictionaries

Collection of information, which is unordered, changeable and indexed

Similar to structures in C/C++
Useful to learn pandas

EXAMPLE:

```
mycat = {
    "color": "red",
    "fur": "short",
    "spots": "tabby"
}
```

KEYS: categories which define my dictionary and to which we want to assign a value
(color, fur, spots)

VALUES: values assigned to the keys
(red, short, tabby)

Python. Dictionaries

OPERATIONS on DICTIONARIES:

* print(dictionary-name)

```
print(mycat)
```

* access an item calling the key

```
x = mycat["color"]
x = mycat.get("color")
```

* change a value

```
mycat["color"] = ‘black’
```

* loop over the keys or the values or both

```
for x in mycat:
    print(x)

for x in mycat.values():
    print(x)

for x,y in mycat.items():
    print(x,y)
```

Python. Dictionaries

OPERATIONS on DICTIONARIES:

* check if a key exists in a dictionary

```
if “color” in mycat:
    print(mycat)
```

* add a new key to an existing dictionary

```
mycat[“age”] = 7.0
print(mycat)
```

* remove a key to an existing dictionary

```
mycat.pop(“age”)
print(mycat)
```

* copy a dictionary into another

```
yourcat = mycat.copy()
```

* create a dictionary with dict() function

```
mycat = dict(color="red", fur="short", spot="tabby", age=7)
```

Python. Dictionaries

OPERATIONS on DICTIONARIES:

* create nested dictionaries (dictionaries of dictionaries):

```
mycats = {
    "ettore" : {
    "color" : "white",
    "fur" : "short",
    "age" : 10
    },
    "ezzelino" : {
    "color" : "red",
    "fur": "short",
    "age": 7
    }
}
print(mycats)
```

```
ettore = {
    "color" : "white",
    "fur" : "short",
    "age" : 10
}
ezzelino = {
    "color" : "red",
    "fur": "short",
    "age": 7
}
mycats = {
    "ettore" : ettore,
    "ezzelino" : ezzelino
}
print(mycats)
```

You find these examples in examples/python/dictionary_example.py

Python. Functions

Functions are sets of instructions

In python can be

*** built-in functions:**

I can call them if I am in the python interpreter

e.g. print() or input()

*** functions that live in packages:**

I should import the package to call them

e.g. math.log(), numpy.zeros()

*** user-provided functions:**

the programmer defines them

The example of a very simple function is in examples/python/simple_def.py

Calculate the square of a variable

Python. Functions

Example of a more complex user-provided function:
examples/python/lookback.py

Calculates look-back time

The look-back time is the difference between the age of the Universe now (at observation) and the age of the Universe at the time the photons were emitted by a celestial body

Expression of look-back time if curvature $\Omega k = 0$

$$t_{\mathrm{lb}} = \frac{1}{H_0} \int_0^z \frac{\mathrm{d}z'}{(1+z')\left[(1+z')^3\Omega_{\mathrm{M}} + \Omega_{\Lambda}\right]^{1/2}}$$

H_0 ~ 67 km/s/Mpc ΩM ~ 0.27 Ω_{Λ} ~ 0.73

Python. Functions

1. scipy package with math libraries

   scipy.integrate to integrate functions

2. scipy.integrate.quad integrates functions numerically

   using the fortran library QUADPACK

3. alternative way to define small functions:

```
lambda x: 1./((1.+x)*(OmegaM*(1.+x)**3.+OmegaL)**0.5)
```

   Equivalent to

```
def integrand(x):
    OmegaM=0.2726 #omega matter
    OmegaL= 0.7274 #omega lambda
    f=1./((1.+x)*(OmegaM*(1.+x)**3.+OmegaL)**0.5)
    return f
```

Python. Functions

User defined functions can be imported as packages

For example

examples/python/lookback3.py

examples/python/lookback3_main.py

from lookback3 import *

Python. EXERCISE on user-defined functions

Proper distance D_p: distance travelled by the light on a given time.

It is simply the lookback time times c (speed of light)

$$D_{\rm p} = c\, t_{\rm lb} = \frac{c}{H_0} \int_0^z \frac{{\rm d}z'}{(1+z')\,\left[(1+z')^3\,\Omega_{\rm M} + \Omega_\Lambda\right]^{1/2}}$$

Comoving distance D_c: distance that does not change in time due to the expansion of the Universe (the expansion of the Universe, 1/(1+z) has been factored out)

$$D_{\rm c} = \frac{c}{H_0} \int_0^z \frac{{\rm d}z'}{\left[(1+z')^3\,\Omega_{\rm M} + \Omega_\Lambda\right]^{1/2}}$$

Luminosity distance D_L: expressed by the relationship between luminosity and flux

$$D_{\rm L} = \frac{c}{H_0}\,(1+z) \int_0^z \frac{{\rm d}z'}{\left[(1+z')^3\,\Omega_{\rm M} + \Omega_\Lambda\right]^{1/2}} = (1+z)\,D_{\rm C}$$

Python. EXERCISE on user-defined functions

EXERCISE:

Write a python script to calculate the comoving distance and the luminosity distance given the redshift. Use scipy.integrate.quad for the integration (as in the previous example).
Suggestion: the expression of the comoving distance (if $\Omega_K = 0$) is the following:

$$D_C(z) = \frac{c}{H_0} \int_0^z \frac{dz'}{\left[(1+z')^3 \Omega_M + \Omega_\Lambda\right]^{1/2}}, \qquad (2)$$

The expression of the luminosity distance (if $\Omega_K = 0$) is the following:

$$D_L(z) = \frac{c}{H_0} (1+z) \int_0^z \frac{dz'}{\left[(1+z')^3 \Omega_M + \Omega_\Lambda\right]^{1/2}} = (1+z)\, D_C(z) \qquad (3)$$

Python. Reading from and writing to files

For ascii files:

numpy.loadtxt or numpy.genfromtxt

```
#see examples/python/read_file.py
import numpy as np

fname="mass_evol.txt"  #input the filename as a string

time,m,mHe,mCO = np.genfromtxt(fname,dtype="float", \
comments="#", usecols=(0,1,3,5), unpack=True)

print(m)
```

fname: input filename
dtype: optional, variable type
comments: optional, does not consider everything after the argument
usecols: optional, which columns you want to store in variables
unpack=True: optional, splits the output per columns

Python. Reading from and writing to files

Self-made function to use less RAM and faster:

```
#see examples/python/read_file2.py
def readfast(fname, N):
    f=open(fname,"r")
    (time,m,mHe,mCO) = (
        np.zeros(N,dtype="float"),
        np.zeros(N,dtype="float"),
        np.zeros(N,dtype="float"),
        np.zeros(N,dtype="float"))
    i=0
    for linetext in f:
        if(linetext[0]==str("#")):
            continue
        word_list = linetext.split()
        #split splits a line in elements
        if(word_list[0]!=str("#")):
            time[i]=np.float(word_list[0])
            m[i]=np.float(word_list[1])
            mHe[i]=np.float(word_list[3])
            mCO[i]=np.float(word_list[5])
            i=i+1
    #end [ for linetext in f ]
    f.close()
    return (time, m, mHe,mCO)
```

Python. Reading from and writing to files

To be called by the main as

```
fname="mass_evol.txt"  #input the filename as a string
N0= 103 #number of lines of the file
(evoltime,mass,massHe,massCO) = readfast(fname,N0)
```

Python. Reading from and writing to files

Writing an ascii file can be done as follows

```
#see examples/python/read_file2.py
fname="output.txt" #output filename as a string
f=open(fname,"w")  #I create file with filename
                   #the new file will be for writing (w)
f.write("# Time(Myr), Mass(Msun), He(Msun), CO(Msun) \n")
for i in range(len(evoltime)):
     f.write(str(evoltime[i])+" "+str(mass[i])+" "+ \
str(massHe[i])+" "+str(massCO[i])+"\n")
```

Python. Regular expressions

Useful when file you want to read is a messy bunch of strings and numbers

A regular expression (or RE) specifies a string or a set of strings that you want to look for in a file

```
#see examples/python/regex.py
import os #This module provides a portable way of
          # using operating system dependent functionality
          # it is needed to call re
import re # regular expression module

m = re.compile('^The mass is 3 Msun')
# the string I want to look for: the mass is 3 Msun
# ^ means that the string should be at the start of a row

fname=str('file_name.txt')
f=open(fname)

for s in f: #s is a generic string in f
     tosearch = m.search(s) # I search for string m in s
     if(tosearch != None):
          print(tosearch)
```

Python. Regular expressions

```
#see examples/python/regex2.py
import os #This module provides a portable way
          # of using operating system dependent functionality
          # it is needed to call re
import re # regular expression module

m = re.compile('^The mass is (\d+) Msun')
# the string I want to look for:
# The mass is some integer number Msun


fname=str('file_name.txt')
f=open(fname)

for s in f: #s is a generic string in f
     tosearch = m.search(s) # I search for string m in s
     if(tosearch != None):
               mass = tosearch.group(1)
               print(mass)
```

Python. Reading files with regular expressions

Everything in regex can be simplified to

\s+ at least one space
\S+ at leas one non space

Python. Reading files with regular expressions

```
import os
import re

m = re.compile('^(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)')

fname=str('five_columns.txt')
f=open(fname)

col1=[]
col2=[]
col3=[]
col4=[]
col5=[]

for s in f:
     tosearch = m.search(s)
     if(tosearch != None):
             col1.append(tosearch.group(1))
             col2.append(tosearch.group(2))
             col3.append(tosearch.group(3))
             col4.append(tosearch.group(4))
             col5.append(tosearch.group(5))
```