

Python in Scientific Computing

Exercises

CSC – Tieteen tietotekniikan keskus Oy
CSC – IT Center for Science Ltd.

Python environment at CSC



- Python is available on all computing servers (hippu, murska, vuori, louhi)
- **module load python**
 - default version 2.6
 - Numpy
 - Scipy
 - Matplotlib
 - Mpi4py
- Python 2.4 in training class
 - Numpy, Scipy, ...

Exercise 1: Playing with the interactive interpreter



- 1) Start the interpreter by typing “python” on the command prompt
- 2) Try to execute some simple statements and expressions e.g.
 - `print “Hello! ”`
 - `1j**2`
 - `1 / 2`
 - `my_tuple = (1, 2, 3)`
 - `my_tuple[0] = 1`
- 3) Many Python modules and functions provide interactive help with a `help` command. Try to execute the following commands
 - `import math`
 - `help(math)`
 - `help(math.sin)`
- 4) Use the interpreter as calculator and with the help of the math module evaluate `cos(45°)`

Exercise 2: Python syntax and code structure:



1) Are the following valid statements?

```
names = ['Antti', 'Jussi']
x2y = 22
3time = 33
my-name = "Jussi"
next = my-name
open = 13
in = "first"
```

2) Are the following pieces of code valid Python

```
numbers = [4, 5, 6, 9, 11]
sum = 0
for n in numbers:
    sum += n
    print "Sum is now", sum
```

```
x = 11
test(x)

def test(a):
    if a < 0:
        print "negative number"
```

Exercise 3: Working with lists and dictionaries



- 1) Create a list with the following fruit names:
“pineapple”, “strawberry”, “banana”
Append “orange” to the list. List objects have a **sort()** function, use that for sorting the list alphabetically (e.g. `fruits.sort()`). What is now the first item of the list?
Remove the first item from the list
- 2) Create a list of even integers up to 10
Create a list of the odd integers up to 10. Use the **range()** function and slicing
Use slicing to extract the last three odd integers in the list
Combine the even and odd integer lists to reproduce list of all integers up to 10
- 3) Create a dictionary whose keys are the fruits
“pineapple”, “strawberry”, and “banana”. As values use numbers representing e.g. prices.
Add “orange” to the dictionary
Remove “banana” from the dictionary

Exercise 4: Control structures



- 1) Write a **for** loop which determines the squares of the odd integers up to 10
Write a **for** loop which sums up to the prices of the fruits of the Exercise 3.3
- 2) Fibonacci numbers are a sequence of integers defined by the recurrence relation
$$F[n] = F[n-1] + F[n-2]$$
with the initial values $F[0]=0$, $F[1]=1$
Create a list of Fibonacci numbers $F[N] < 100$ using a **while** loop
- 3) Write a control structure which checks whether an integer is negative, zero, or belongs to the prime numbers 3,5,7,11,17 and perform e.g. corresponding **print** statement.
Keyword **in** can be used for checking whether a value belongs to a list:

```
>>> 2 in [1,2,3,4]
True
```

Exercise 5: Obtaining input

- 1) Interactive input can be requested with the `input()` function:

```
>>> x = input("Meaning of life: ")
Meaning of life: 42
>>> x
42
```

Write a small program that calculates the area of circle based on the radius given by the user.

- 2) Modify the program so that a set of radii is read from the command line arguments.

Note that `sys.argv` contains the arguments as strings, use explicit type conversion with `float()` in order to obtain floating point numbers.

List comprehension is useful Python idiom here:

```
>>> strs = ["1", "2", "3"]
>>> numbers = [float(x) for x in strs]
>>> numbers
[1.0, 2.0, 3.0]
```

Exercise 6: Modules and functions



- 1) Write a function which calculates the arithmetic mean from a list of numbers.
- 2) Write a function that converts a polar coordinate representation (r, φ) into cartesian representation (x,y) . Write also a function which does the reverse transformation. The important formulas are:
$$x = r \cos(\varphi) \quad y = r \sin(\varphi) \quad r^2 = x^2 + y^2$$
Use the **math** module.
- 3) Implement the coordinate transformation functions in their own module '**polar.py**'. Import the functions into a main script, which reads arbitrary number of (x,y) value pairs from the terminal with the **input** function. Transform the cartesian coordinates into a polar representation and sort the polar coordinate pairs according to r . Finally, print out the cartesian coordinates corresponding to the sorted polar coordinates.

Exercise 7: Working with files



- 1) The file “exercise7_1.dat” contains list of (x, y) value pairs. Read the values into two lists **x** and **y**
- 2) The file “exercise7_2.txt” contains output from a simulation run where the geometry of CO molecule is optimized. One part of the output is the free energy during geometry optimization. Free energies are in the lines of type:

Free Energy: -16.47537

Read the free energies from the file and print out how much each energy differs from the final value.

- 3) The file “exercise7_3.txt” contains a short piece of text. Determine the frequency of words in the file, i.e. how many times each word appears. Print out the ten most frequent words

Read the file line by line and use the **split()** function for separating a line into words.

The frequencies are stored most conveniently into a dictionary, but for sorting the dictionary has to be converted into a list of (key, value) pairs with the **items()** function:

```
>>> words = {"foo" : 1, "bar" : 2}
>>> words.items()
[('foo', 1), ('bar', 2)]
```



- 4) The file “CH4.pdb” contains the coordinates of methane molecule in a PDB format. The file consists of header followed by record lines which contain the following fields: record name(=ATOM), atom serial number, atom name, x-,y-,z-coordinates, occupancy and temperature factor. Convert the file into XYZ format: first line contains the number of atoms, second line is title string, and the following lines contain the atomic symbols and x-, y-, z-coordinates, all separated by white space. Write the coordinates with 6 decimals.

Exercise 8: Steering a simulation with Python



- 1) Write a Python program which does loop over different values of x (use e.g. a self-specified list). At each iteration, write an “input file” of the form (here $x=4.5$):

```
#!/bin/csh
set x=4.500000
set res=`echo "$x^2" | bc`
echo "Power 2 of $x is $res"
```

As a “simulation program” use `csh` and execute the input file at each iteration with the help of `os` module:

```
os.system('csh inp > out')
```

Read (x,y) pairs from the “output file” **out** and save them for later processing.

Exercise 9: Simple usage of classes



- 1) Define a class for storing information about an element. Store the following information:

name, symbol, atomic number, atomic weight

Look www.weblements.com for data and construct instances for few elements. Store the instances as values of dictionary whose keys are the element names

- 2) The file “solvents.dat” contains some information about different solvents. Define a class for storing the information. Read the data from the file and construct a list of solvents. Sort the solvents according to density and print out the names. Hint: use [sorted](#) -function as well as [attrgetter](#) -function from [operator](#) module.

Exercise 10: Simple numpy usage



- 1) Use the data of Exercise 7.1 and fit a second order polynomial to the data using `numpy.polyfit()`
- 2) Generate a 10 x 10 array whose elements are uniformly distributed random numbers using `np.random` module
 - extract every second element from the fifth column
 - extract every second element from the fifth row
 - calculate the mean and standard deviation of the above two vectors using `np.mean` and `np.std`
- 3) Investigate the behavior of the statements below by looking the values of the arrays **a** and **b** after assignments

```
a = np.arange(5)
b = a
b[2] = -1
b = a[:]
b[1] = -1
b = a.copy()
b[0] = -1
```



4) Construct two symmetric 2 x 2 random matrices A and B (hint: a symmetric matrix can be constructed easily from a general matrix as $A_{\text{sym}} = A + A^T$)

Calculate the matrix product of $C = A * B$ using `numpy.dot()`

Calculate the eigenvalues of matrix C with `numpy.linalg.eigvals()`

Exercise 11: Numerical computations with Python



- 1) Integral of function $f(x)$ in the interval $[a,b]$ can be calculated with trapezoidal method as follows:

$$\int_a^b f(x) dx \approx \frac{b-a}{n} \left[\frac{f(a) + f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + k \frac{b-a}{n}\right) \right].$$

Write a function which takes as an argument the function to be integrated, the end points **a** and **b** as well as the number of points **n**.

As test functions use $f(x) = 1 + x^2$ and $f(x) = \exp(-x^2)$

Test how the integral varies with the number of integration points. Try to avoid **for** loops.

- 2) Integrals can be calculated also with the Monte-Carlo method as

$$\int_a^b f(x) dx \approx \frac{b-a}{n} \sum_{k=1}^n f(x_k)$$

where x_k are uniformly distributed random numbers between **a** and **b**. Evaluate the integrals of previous Exercise with Monte-Carlo method using **numpy.random** module

3) Poisson equation in one dimension is

$$\frac{d^2 \phi}{dx^2} = -\rho$$

where ϕ is the potential and ρ is the source. The equation can be discretized on uniform grid, and the second derivative can be approximated by the finite differences as

$$\frac{d^2 \phi(x_i)}{dx^2} = \frac{\phi(x_{i+1}) - 2 * \phi(x_i) + \phi(x_{i-1}))}{h^2}$$

where h is the spacing between grid points.

Using the finite difference representation, the Poisson equation can be written as

$$\phi(x_{i+1}) - 2 * \phi(x_i) + \phi(x_{i-1}) = -h^2 \rho(x_i)$$

The potential can be calculated iteratively by making an initial guess and then solving the above equation for $\phi(x_i)$ repeatedly until the differences between two successive iterations are small (this is so called Jacobi -method).

Use

$$\rho(x) = 2x^2 - 1$$

as a source together with boundary conditions $\phi(0)=0$ and $\phi(1)=0$ and solve the Poisson equation in the interval $[0,1]$.

Compare the numerical solution to the analytic solution

$$\phi(x) = \frac{-x}{3} + \frac{x^2}{2} - \frac{x^4}{6}$$

The Poisson equation can be solved more efficiently with the conjugate gradient method which is a general method for the solution of linear systems of type $Ax = b$.

Interpret the Poisson equation as a linear system and write a function which evaluates the second order derivative (i.e. the matrix – vector product Ax). You can assume that the boundary values are zero.

Solve the Poisson equation with the conjugate gradient method and compare its performance to the Jacobi method. The pseudo-code for the conjugate gradient method is

```

 $\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$ 
 $\mathbf{p}_0 := \mathbf{r}_0$ 
 $k := 0$ 
repeat
   $\alpha_k := \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$ 
   $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$ 
   $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$ 
  if  $\mathbf{r}_{k+1}$  is sufficiently small then exit loop end if
   $\beta_k := \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$ 
   $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$ 
   $k := k + 1$ 
end repeat
The result is  $\mathbf{x}_{k+1}$ 

```



Exercise 12: Simple plotting

- 1) Plot to the same graph **sin** and **cos** functions in the interval $[-\pi/2, \pi/2]$. Use Θ as x-label and insert also legends. Save the figure in .png format.
- 2) The file “csc_usage.dat” contains the usage of CSC servers by different disciplines. Plot a pie chart about the resource usage.
- 3) The file “contour_data.dat” contains cross section of electron density of benzene molecule. Make a contour plot of the data. Try both contour lines and filled contours. Hint: data can be read with numpy as

```
data = np.loadtxt(filename)
```
- 4) The file “atomization_energies.dat” contains atomization energies for a set of molecules, calculated with different approximations. Make a plot where the molecules are in x-axis and different energies in the y-axis. Use the molecule names as tick marks for the x-axis

Exercise 13: Using Scipy

- 1) Use the **integrate** module for evaluating the integrals of Exercise 11.1
- 2) Choose some special function from Scipy and investigate its behaviour in the chosen interval
- 3) Solve the Poisson equation of Exercise 11.3 with the help of Scipy. Define the matrix-vector product and solve the linear system with conjugate gradient method. Plot the numerical solution together with the analytic solution.
- 4) Find the minimum of the Rosenbrock function

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} (x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$$

by utilizing also the gradient:

$$\frac{\partial f}{\partial x_j} = 200(x_j - x_{j-1}^2) - 400x_j(x_{j+1} - x_j^2) - 2(1 - x_j)$$

$$\frac{\partial f}{\partial x_0} = -400x_1(x_1 - x_0^2) - 2(1 - x_0)$$

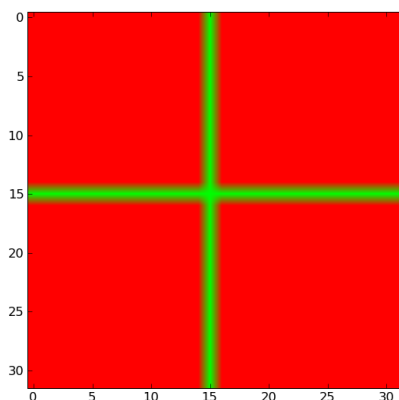
$$\frac{\partial f}{\partial x_{N-1}} = 200(x_{N-1} - x_{N-2}^2)$$

Exercise 14: Game of Life

- 1) Game of life is a cellular automaton devised by John Conway in 70's,
http://en.wikipedia.org/wiki/Conway's_Game_of_Life.
Game consists of two dimensional orthogonal grid of cells. Cells are in two possible states, **live** or **dead**. Each cell interacts with its eight neighbours, and at each time step the following transitions occur.
 - Any live cell with fewer than two live neighbours dies, as if caused by underpopulation
 - Any live cell with more than three live neighbours dies, as if by overcrowding
 - Any live cell with two or three live neighbours lives on to the next generation
 - Any dead cell with exactly three live neighbours becomes a live cell

The initial pattern constitutes the seed of the system, and the system is left to evolve according to rules. Deaths and births happen simultaneously.

Implement the Game of Life using Numpy, and visualize the evolution with Matplotlib (e.g. *imshow*). Try first 32x32 square grid and cross-shaped initial pattern,



Try also other grids and initial patterns (e.g. random pattern).



Exercise 15: Mpi4py

- 1) Try to implement a ping-pong test with Mpi4py. Ping-pong test measures the efficiency of communication between two processes. One process sends data to other process, and after receiving the other process sends the data back. Measure the time spent in the communication with different sized Numpy arrays as well as with e.g. Python lists or dictionaries.
- 2) Try to parallelize Game of Life by distributing the grid along one dimension to different processors.