Lab1_solutions

September 6, 2017

1 Solutions for exercises of lab 1

1.1 Exercise 1

Compute first 30 Fibonacci numbers, which are defined by equations $f_0 = 1$; $f_1 = 1$; $f_i = f_{i-1} + f_{i-2}$, $i \geq 2$.

Solution Fist make sure that the package NumPy is imported by executing the following line

```
In [19]: import numpy as np
```

Let us make our code more universal by by introducing a variable n which specifies how many values we want. For solving the problem we will give it the value 30, but it can be easily changed if we want to see a different number of Fibonaccy numbers.

```
In [20]: n=30
```

Create a vector with n elements to store the Fibonacci numbers, filled initially with ones (then first two elements (with indexes 0 and 1) have correct values

```
In [21]: f=np.ones(n)
```

Now we need to repeat a line for computation $ f_{i}=f_{i-1}+f_{i-2}$ for i=2,3,...,n-1 A complect of such i values can be produced with several several commands, for exampel `range(2,n)` or `arange(2,n)`. For this course, there is not much difference which one to use in the case of `for` cycle but outside of definition of `for` cycles we need the second command. If you want to know the detailed differences of the commands, please ask. Let us compute all the values with a `for` cycle and print out the values:

```
In [22]: for i in range(2,n):
             f[i]=f[i-1]+f[i-2]
         print(f)
```

```
[  1.00000000e+00   1.00000000e+00   2.00000000e+00   3.00000000e+00
   5.00000000e+00   8.00000000e+00   1.30000000e+01   2.10000000e+01
   3.40000000e+01   5.50000000e+01   8.90000000e+01   1.44000000e+02
   2.33000000e+02   3.77000000e+02   6.10000000e+02   9.87000000e+02
   1.59700000e+03   2.58400000e+03   4.18100000e+03   6.76500000e+03
   1.09460000e+04   1.77110000e+04   2.86570000e+04   4.63680000e+04
   7.50250000e+04   1.21393000e+05   1.96418000e+05   3.17811000e+05
   5.14229000e+05   8.32040000e+05]
```

The answers are printed out in the scientific notation of the form $a \cdot 10^b$, where a is the number before e and b is the number after e, so 2.33000000e+02 is actually $2.33 \cdot 10^2 = 233$.

1.2 Exercise 2

Generate $n = 20$ values from the standard normal distribution and store the values in a vector `x`. Compute the values of the vector `y` defined by

$$y_0 = x_0;\ y_{n-1} = x_{n-1};\ y_i = \frac{x_{i-1} + x_i + x_{i+1}}{3},\ i = 1, 2, \ldots, n-2.$$

Solution Generating values for vector x is easy, it can be don with the command `randn(n)`. Each time we execute the command, we should get `n` new values from the standard normal distribution and usually this is what we want. But in order to to compare the results of two different solutions (the sample solution and your own solution), it is possible to fix a set of the random numbers by executing the command `random_seed(a_number)` before using `randn(n)` - after using this command (with the same integer `a_number`) we always get the same result for generated random numbers.

```
In [23]: n=20
         np.random.seed(200)
         x=np.random.randn(n)
         print(x)
```

```
[-1.45094825  1.91095313  0.71187915 -0.24773829  0.36146623 -0.03294967
 -0.22134672  0.47725678 -0.69193937  0.79200593  0.07324913  1.30328603
  0.21348149  1.01734895  1.91171178 -0.52967163  1.84213516 -1.05723508
 -0.86291629  0.2376315 ]
```

By adding `np.random.seed(200)` to your own code just before using `randn(n)`, you should get exactly the same numbers.

Define a vector y filled with n zeros and set the first and last elements:

```
In [24]: y=np.zeros(n)
         y[0]=x[0]
         y[n-1]=x[n-1]
```

Now it is possible to compute the values for y_i, $i = 1, \ldots, n-1$ like in the case of the previous exercise with a for cycle like `for i in range(1,n-1):  y[i]=(x[i-1]+x[i]+x[i+1])/3`

but it is a much better idea to avoid for cycles by using vector operations. There are many ways to achieve this, but for beginners the simplest one is to use the property of scipy package that instead of a single index we can use vectors of indexes. For example, if i is a vector with components 2,3,4, then x[i] is also a vector with components x[2],x[3],x[4]. Adding a number to a vector adds the number to all elements of the vector, so i+2 has components 4,5,6 and x[i]+x[i+2] gives a vector with three components x[2]+x[4],x[3]+x[5],x[4]+x[6] and it turns out that this approach is faster than computing the numbers one by one by in a for cycle if the number of terms computed is not very small.

So a good solution for computing y is as follows:

```
In [25]: i=np.arange(1,n-1)
         y[i]=(x[i-1]+x[i]+x[i+1])/3
         print(y)
```

```
[-1.45094825  0.39062801  0.791698    0.27520237  0.02692609  0.03572328
  0.07432013 -0.14534311  0.19244111  0.0577719   0.72284703  0.53000555
  0.84470549  1.04751407  0.79979637  1.0747251   0.08507615 -0.0260054
 -0.56083996  0.2376315 ]
```

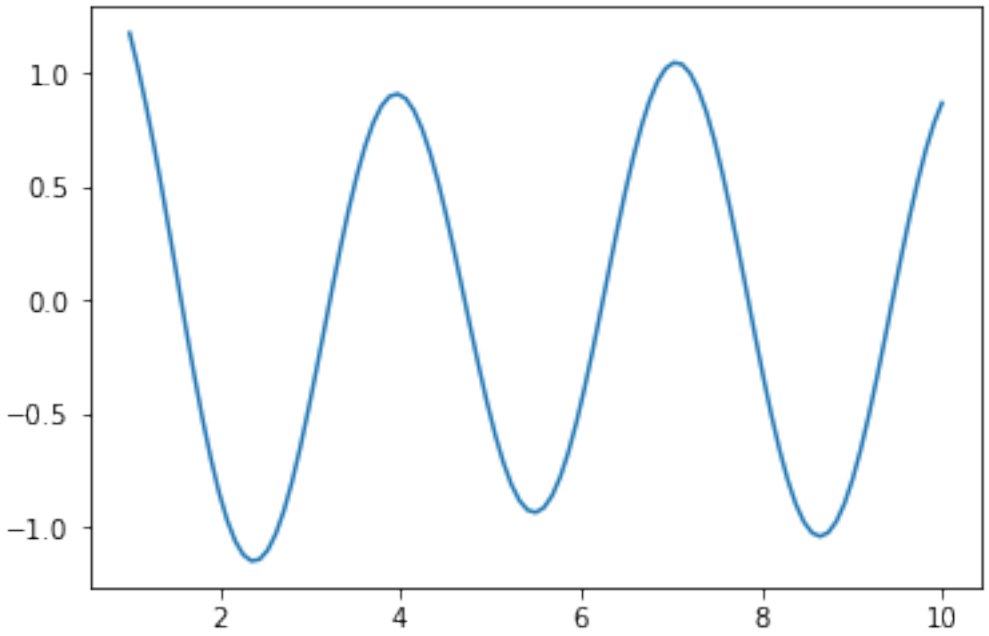
for n=20 the difference in speed of the two approaches is not very large but for n=100 the second approach is approximately 6 times faster than the first one, for n=1000 the second approach is approximately 20 times faster.

1.3 Exercise 3

Define a function $g(x) = \sin(2x) + \frac{\cos(x)}{2x}$. Plot the graph of the function for $1 \leq x \leq 10$ by using the values of the function at 101 points.

Solution

```
In [26]: def g(x):
             return np.sin(2*x)+np.cos(x)/(2*x)
         n=100
         x=np.linspace(1,10,n+1)
         import pylab
         pylab.plot(x,g(x))
         pylab.show()
```



If you want to make your graph nicer, you can use additional commands like `xlabel`, `ylabel`,`title` from pylab and use them (like `pylab.xlabel('x')` on a separate line) to set labes for axes and the title of the graph before using the command `pylab.show`.

1.4 Exercise 4

Write a function `simpleMax(f,a,b,n)` that computes approximately the maximal value of the function f over the interval $[a,b]$ by evaluation it at the points a, $a+h$, $a+2h,\ldots,b$, where $h=\frac{b-a}{n}$. Use the function `simpleMax` to compute approximately the maximal value of the function g from the previous exercise for $x\in[1,3]$ in the case $n=200$ (answer should be 1.1794485797597516).

Solution

Again it is possible to solve the problem by using vectorized commands. A good way is to compute first a vector `x` containing all x values we want to use, then compute with a single command the values of the function f at those points (just by `f(x)`) and then compute the maximal value of elements of a vector by the function `amax`.

```
In [27]: def simpleMax(f,a,b,n):
             x=np.linspace(a,b,n+1)
             return np.amax(f(x))
         print(simpleMax(g,1,3,200))
```

```
1.17944857976
```

1.5 Exercise 5

Write a function `Simple2dMax(f,a,b,m,c,d,n)` which computes approximately the maximum of a function of 2 variables by computing the maximal value at the points

$$x_i=a+ih_1,\ y_j=c+jh_2,\ i=0,\ldots,m,\ j=0,\ldots,n,$$

where $h_1=\frac{b-a}{m}$, $h_2=\frac{d-c}{n}$. Test the correctness of the function by computing the maximal value of $u(x,y)=x^2+xy-2y^2$ over the unit square $0\le x\le 1$, $0\le y\le 1$ in the case $m=10$, $n=20$ (the answer should be 1.125).

Solution

It is a little tricky exercise if one wants to use vector operations to have an efficient code. It is important to understand that most functions work nicely in scipy if one of the arguments is replaced by a vector but using a vector for sevaral arguments at the same time does not usually give us what we want. A reasonable approach is to use one for cycle to let one of the variables to take values one by one, but use vector commands to compute maximum over all values of the other variable.

```
In [28]: def simple2dMax(f,a,b,m,c,d,n):
             x=np.linspace(a,b,m+1)#all x values
             y=np.linspace(c,d,n+1)#all y values
             maxval=f(a,c) #current maximal value
             for j in range(n+1):
                 max_for_current_y=np.amax(f(x,y[j]))
```

```
            maxval=max(maxval,max_for_current_y)#max for all y we have considered so far
        return maxval
    def u(x,y):
        return x**2+x*y-2*y**2
    print(simple2dMax(u,0,1,10,0,1,20))
```

```
1.125
```