FYSS5120 Efficient Numerical Programming - Demo 1

This is a warm up, no need to return any solutions.

1. Start the Python interpreter and import packages

```
$ python
>>> import numpy as np
>>> import scipy as sc
>>> import matplotlib.pyplot as plt
```

2. The code demo1_factorials.py. shows a recursive computation of factorials and how it's converted to an iteration. Try computing the factorial of 1000.

3. Let's compute Fibonacci numbers and test cache decorators. The code demo1_cachedecorator.py shows how the module `cProfile` is used in a script (the command line usage is in the lecture notes).
Fibonacci numbers 0, 1, 1, 2, 3, 5, 8 ... are sums of two previous numbers, and one usually starts from the bottom. The n:th Fibonacci number can be computed from top down in a recursive function `fib(n)`, but then the interpreter has to keep track of the tree structure: `fib(n)` calls `fib(n-1)` and `fib(n-2)` etc. This long *call stack* makes recursion slow. The n:th Fibonacci number can be computed iteratively, as in

demo1_fiboiter.py

```
def fibonacci_iter(n):
    i, j  = 0, 1 # sequence starts with 0, 1
    while n > 0:
        i, j, n = j, i+j, n-1
    return i

if __name__=='__main__':
    for n in range(31):
        print(n,fibonacci_iter(n))
```

The iterative function traverses from bottom to top, and every stage holds enough information to continue.

Take-home messages:

- Recursion is fine, unless it's serious a bottleneck and you always stay below the recursion limit
- If you can, convert recursion to iteration
- Consider using a cache decorator from module `functools`

Vesa will talk about B-splines (link to scipy.interpolate.BSpline).