Scientific Programming
(Wissenschaftliches Programmieren)

Exercise 4

1. Fibonacci numbers as function

- Create a Python-script / IPython-notebook which generates Fibonacci numbers.
- The first two elements of the Fibonacci series are 1, otherwise each element is the sum of the previous two elements.
- The script should contain a function, which returns the Fibonacci-numbers as a list.
- The function should be called with one argument only (the number of the Fibonacci numbers to calculate)
- Example call:

```
fibonacci(10)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

2. Generalized Fibonacci numbers

- Create a Python-script / IPython-notebook which generates “generalized” Fibonacci numbers of order *N*.
- The first *N* elements of the generalized Fibonacci series should 1 one, otherwise each element should be the sum of the previous *N* elements.
- The script should contain the a function, which takes the number of terms to generate and the order as arguments and returns an **array** with the generalized Fibonacci numbers. The order argument should be optional with a default value of 2.
- Example calls:

```
fibonacci(10)
array([ 1,  1,  2,  3,  5,  8, 13, 21, 34, 55])

fibonacci(10, 3)
array([  1,   1,   1,   3,   5,   9,  17,  31,  57, 105])

fibonacci(10, order=3)
array([  1,   1,   1,   3,   5,   9,  17,  31,  57, 105])
```

3. Matrix multiplication

- In order to exercise the matrix access, create a function which multiplies two matrices. For the purpose of this exercise, code the matrix multiplication algorithm by hand without using the numpy.dot() or numpy.matmul() functions.
- Test your function for various matrices by comparing the results against those obtained by numpy.dot().
- Example:

```
m1 = np.array([[1, 2, 3], [4, 5, 6]])
m2 = np.array([[1, 2], [3, 4], [5, 6]])
matrix_product(m1, m2)
array([[ 22.,  28.],
       [ 49.,  64.]])
```

4. *Optimized generalized Fibonacci number algorithm

- Try to optimize the “generalized” Fibonacci number generator, so that instead of summing up *N* terms in order to create each new element, it should only use 2 arithmetic operations to generate each new element. *N* is the order of the generalized Fibonacci numbers.

5. *Matrix determinant*

- Write a function which calculates the determinant of a square matrix. It should take the matrix as argument and return the determinant.
- Example:

```
mtx = np.array([[1.0, 2.0], [3.0, 4.0]])
determinant(mtx)
-2.0
```

- Try it with various 2x2 and 3x3 matrices and compare the result to those obtained by the numpy.linalg.det() function.