Physik auf dem Computer **SS 2013**

Worksheet 3: Taylor Series, Interpolating Polynomial and Splines

May 3, 2013

General Remarks

- Deadline is **Wednesday, 8th May 2013, 10:00**
- On this worksheet, you can achieve a maximum of 10 points.
- To hand in your solutions, send an email to
  - Olaf (`olenz@icp.uni-stuttgart.de`; Wednesday, 14:00–15:30)
  - Elena (`minina@icp.uni-stuttgart.de`; Wednesday, 15:45–17:15)
  - Tobias (`richter@icp.uni-stuttgart.de`; Friday, 15:45–17:15)
- Attach all required files to the mailing. If asked to write a program, attach the *source code* of the program. If asked for a text, send it as PDF or in the text format. We will *not* accept MS Word files!
- The worksheets are to be solved in groups of two or three people.
- The tutorials take place in the CIP-Pool of the ICP in Allmandring 3.

Throughout the worksheet, the following functions are used on the specified domains:

Name	**Definition**	**Domain**
Sine Function	$f(x) = \sin x$	$[0, 2\pi]$
Runge Function	$g(x) = \frac{1}{1+x^2}$	$[-5, 5]$
Lennard-Jones Function	$h(x) = x^{-12} - x^{-6}$	$[1, 5]$

In this worksheet, different *callable* objects are used. If a Python class defines a method `__call__(self,args)`, it is possible to call an instance of the object like a function:

```
>>> class HelloWorld:
...   def __call__(self, who):
...     print "Hello", who, "!"
>>> hello = HelloWorld()
>>> hello("Olaf")
Hello Olaf !
```

Task 3.1 (3 points): Taylor Polynomials

In this task, you are entitled to plot the Taylor polynomials of the different functions.

- 3.1.1 (2 points) Calculate the coefficients of the truncated Taylor series of the sine function $f(x)$ at $x_0 = 0$, the Runge function $g(x)$ at $x_0 = 0$ and the Lennard-Jones function at $x_0 = 1$ up to 6th degree. Use the Python class `numpy.poly1d` to define the k-th order Taylor polynomials of $f(x)$, $g(x)$ and $h(x)$ for $k \in \{3, 5, 10\}$ at arbitrary x.
- 3.1.2 (1 point) For each of the functions $f(x)$, $g(x)$ and $h(x)$, create a plot that shows the function and their respective k-th degree Taylor polynomials ($k \in \{3, 5, 10\}$) on the specifed domain.

Hints

- You may use Mathematica or Wolfram alpha to compute the Taylor series.
- The Python class `poly1d` is used as follows:

```
# f is the polynomial f(x) = 3*x**2 + 2*x + 1
# Note the order of the coefficients!
f = numpy.poly1d([3,2,1])
# Compute the value of the polynomial at x=42
print f(42)
```

- Take care to handle the argument of the Taylor polynomial of $h(x)$ correctly!

Task 3.2 (4 points): Interpolating Polynomials

Now you should plot interpolating polynomials of the three functions.

- 3.2.1 (1 point) For each of the functions $f(x)$, $g(x)$ and $h(x)$, create a plot over the specified domain that shows the function and its k-th degree interpolating polynomials (($k \in \{3, 5, 10\}$) for equidistant supporting points. Use the Python function `scipy.interpolate.lagrange` to do so.
- 3.2.2 (2 points) Write a Python program that does the same as the script from the previous task (3.2.1). However, this time you shall implement your own class `NewtonInterpolation` that does the interpolation using Newton's representation. The class shall define two methods:
  - `__init__(self,x,y)` does the same as `neville()` from the lecture script and stores the x and γ in class variables
  - `__call__(self,x)` does the same as `horner()` from the lecture script but does not require the parameters `gamma` and `x` but uses the stored class variables instead

- 3.2.3 (1 points) Modify the Python program either from task 3.2.1 or 3.2.2 such that it computes the interpolating polynomials at the Chebyshev nodes and create the same plots as in the previous tasks.

Task 3.3 (3 points): Spline Interpolation

In this task, you are to spline interpolate the functions.

- 3.3.1 (1 point) For each of the functions $f(x)$, $g(x)$ and $h(x)$, create a plot over the specified domain that shows the function and an interpolating cubic spline with $k \in \{3, 5, 10\}$ equidistant supporting points.
  Use the class `scipy.interpolate.interp1d` to do so.
- 3.3.2 (2 points) Write a class `SplineInterpolation` that implements cubic spline interpolation. As in previous tasks, the class shall provide the method `__init__(self,args)` to initialize the interpolation and `__call__(self,x)` to compute the value of the interpolating function at `x`. On the boundaries, set the second derivative of the spline to 0. Note that the splines in this class will not be identical to the splines in the previous task, as SciPy uses differemt boundary conditions.

Hints The method `SplineInterpolation.__init__()` has to compute the spline coefficients by first generating the defining linear equations (eq. (3.26) in the lecture script) and then solving them using `scipy.linalg.solve`.