Physik auf dem Computer **SS 2014**

Worksheet 4: Interpolation

April 30, 2014

General Remarks

- The deadline for the worksheets is **Wednesday, 7 May 2014, 10:00** for the tutorial on Friday and **Friday, 9 May 2014, 10:00** for the tutorials on Tuesday and Wednesday.
- On this worksheet, you can achieve a maximum of 10 points.
- To hand in your solutions, send an email to
  - Johannes (`zeman@icp.uni-stuttgart.de`; Tuesday, 9:45–11:15)
  - Tobias (`richter@icp.uni-stuttgart.de`; Wednesday, 15:45–17:15)
  - Shervin (`shervin@icp.uni-stuttgart.de`; Friday, 11:30–13:00)

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	$h(x) = x^{-12} - x^{-6}$	$[1, 5]$

The Python program `ws4.py` and the IPython Notebook `ws4.ipynb` demonstrate how to create plots of the functions and their interpolating polynomials respectively interpolating splines using SciPy's functions.

Task 4.1: Interpolating Polynomials (4 points)

In this task, you should implement the interpolating polynomials.

- **4.1.1** (2 points) 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

  Redo the plots from the demo using this new class.
- **4.1.2** (2 points) Modify the Python program from any of the previous tasks such that it computes the interpolating polynomials at the Chebyshev nodes and create the same plots as in the previous tasks.

Task 4.2: Spline Interpolation (6 points)

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

- **4.2.1** (2 points) Write a class `CubicSplineInterpolation` that implements cubic spline interpolation. As in the previous tasks, the class shall provide `__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 different boundary conditions. Use the class to do the same plots as in the demo.

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

- **4.2.2** (2 points) Derive the equations for the *quadratic spline*, where the spline polynomial is defined as $P_i(x) = y_i + m_i(x - x_i) + M_i(x - x_i)^2$. The conditions for the quadratic spline are that it has the function value at the supporting points x_i and x_{i+1} and that the first derivative of the spline at x_{i+1} is the same for P_i and P_{i+1}.

- **4.2.3** (2 points) Write a class `QuadraticSplineInterpolation` that implements the quadratic spline interpolation. Redo the same plots as in the previous tasks with quadratic splines.