Worksheet 4: Lagrange Interpolation

May 15, 2017

General Remarks

- The deadline for handing in the worksheets is **Monday, May 22nd, 2017, 12:00 noon**.
- For this worksheet, you can achieve a maximum of 10 points.
- To hand in your solutions, send an email to your tutor:
  - Johannes Zeman `zeman@icp.uni-stuttgart.de` (Tue 15:45–17:15)
  - Michael Kuron `mkuron@icp.uni-stuttgart.de` (Wed 15:45–17:15)
  - Kai Szuttor `kai@icp.uni-stuttgart.de` (Thu 14:00–15:30)
- Please try to only hand in a single file that contains your program code for all tasks. If you are asked to answer questions, you should do so in a comment in your code file. If you are asked for graphs or figures, it is sufficient if your code generates them. You may as well hand in a separate PDF document with all your answers, graphs and equations.
- The worksheets are to be solved in groups of two or three people.

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` contains a class `Newton`, which implements polynomial interpolation using the Newton Algorithm[1]. It inherits a part of its constructor from the base class `Interpolation`, which accepts the x- and y-values of the support points required for interpolation. The corresponding values are then available as class members `support_x` and `support_y`. Its method `__call__(self,x)` implements the Horner scheme to evaluate the interpolating polynomial at arbitrary positions `x` and returns the results as a NumPy array.

The program also provides a convenience function `generate_plots()`, which is able to automatically generate plots of the functions specified in the table above together with the respective interpolating polynomials.

[1]The German wikipedia site about polynomial interpolation provides a good explanation of the algorithm, if you're interested.

Task 4.1: Lagrange Interpolation (5 points)

Extend the program by writing an additional Python class `Lagrange`, which uses the Neville-Aitkens scheme (as discussed in the lecture) to implement Lagrange interpolation.

The class has to fulfill the following requirements:

- **4.1.1** (1 point) It shall inherit its constructor (member function `__init__(self, ...)`) from the base class `Interpolation`.
- **4.1.2** (3 points) Similar to the example class `Newton`, it must be callable. This means that it must contain a method `__call__(self, x)`, which accepts a 1-d NumPy array `x` containing positions on which the interpolating polynomial shall be evaluated.
  In the `__call__(self, x)` method, implement the Neville-Aitkens scheme to evaluate the interpolating polynomial at all given positions `x` and return the result as a NumPy array.
  Do *not* use any functions from `scipy.interpolate` or similar to do so.
- **4.1.3** (1 point) Use the function `generate_plots()` to plot the interpolating polynomials of all functions given in the table above – computed by your new class `Lagrange` – for $n \in \{5,\ 10,\ 15\}$ support points.

Task 4.2: Chebyshev Nodes (5 points)

- **4.2.1** (1 point) One might assume that the approximation of a function by an interpolating polynomial should improve if the number of support points is increased. Based on the plots you generated in the previous task (or by the ones the unmodified program generates), can this assumption be valid in general? Give reasons for your answer.

In order to interpolate a function $f(x)$ on the interval $[a, b]$ using n support points, the Russian mathematician Pafnuty Chebyshev developed an equation to determine the position of support points x_k based on Chebyshev polynomials:

$$x_k = \frac{(a+b)}{2} + \frac{(a-b)}{2}\cos\left(\frac{2k+1}{2n}\pi\right), k = 0, 1, \dots n-1 \tag{1}$$

Such support points are usually referred to as Chebyshev nodes. Of course, the corresponding y-values are still determined as $y_k = f(x_k)$.

- **4.2.2** (1 point) What is the advantage of using Chebyshev nodes as supporting points for polynomial interpolation? You may want to search the web for this.
- **4.2.3** (2 points) Extend your program by implementing a function `chebyshev(a, b, n)` which returns a 1-d Numpy array containing n Chebyshev nodes on the interval $[a, b]$.
- **4.2.4** (1 point) Plot the interpolating polynomials of all functions using $n \in \{5, 10, 15\}$ Chebyshev nodes as support points. Compare the resulting plots to the previous plots, which used equidistant support points, and brifly comment on the differences.