Numerical Mathematics I, 2024, Unit 1

Deadline for discussion: First attempt: May 2nd – 17h // Resit: May 9th – 17h

Keywords: Lagrange interpolation, accuracy, stability & composite integration

IMPORTANT INSTRUCTIONS

- The deadline indicated above indicates when to upload your JuPyteR notebook (a single one, containing all answers) to the Assignments' page in Brightspace. Both students of the pair need to upload it. **If this is not the case, the students will receive the minimal grade. They can still run the discussion that day, but it will count as RESIT so no additional opportunity will be given.**
- The assessor will contact you to arrange a date and time for the discussion. The discussion should take place after the indicated deadline and before the respective test.
- All group partners will be interrogated, so be prepared for answering all questions. The same grade will be given to all students.
- The assessor may randomly choose some pair(s) to conduct the discussion individually.
- Make sure that you have done the Tutorial before starting the programming assignments. We HIGHLY recommend you to proceed in the order: Part A Tutorial→Programming, Part B, ...
- Please give us your feedback during and/or right after the discussion with the TA. THIS IS THE LINK TO THE FEEDBACK FORM.

The official material of the course are the lectures. You should start studying that in the first place. Sections A,B and C are based on Lectures A, B and C, respectively.

Tutorial

A Lagrange interpolation

We will consider the interpolation of some function f on the interval $[a, b]$.

Lagrange interpolation

1. Let $f(x) = \sin(x)$ on the interval $[0, 2\pi]$. Sketch f and the equidistant Lagrange interpolant of polynomial order $1, 2$ and 3.
2. Give the definition of the order n Lagrange interpolant $\Pi_n f$ defined on the nodes $a = x_0, \ldots, x_n = b$, in terms of the Lagrange basis functions

$$\varphi_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j}.$$

Show that it is an interpolant

$$\Pi_n f(x_i) = f(x_i), \quad i = 0, \ldots, n,$$

and verify that this holds in your sketch.

3. The Lagrange interpolant can also be written in terms of the centered coefficients c_i

$$\Pi_n f(x) = \sum_{i=0}^{n} c_i (x-m)^{n-i}, \tag{1}$$

where $m = (a+b)/2$. Find c_i in terms of the values of $f(x)$ at the interpolatory nodes for $n = 1$ and $n = 2$.

4. Show that equidistant Lagrange interpolation of order n satisfies the following error bound

$$|E_n f(x)| = |f(x) - \Pi_n f(x)| \leq \frac{\max_{y\in[a,b]} |f^{(n+1)}(y)|}{4(n+1)} \left(\frac{b-a}{n}\right)^{n+1}, \tag{2}$$

by following the following steps:

   (a) For fixed $x \in [a,b]$, we define the auxiliary function

   $$Y(t) = E_n f(t)\omega_{n+1}(x) - E_n f(x)\omega_{n+1}(t),$$

   where

   $$\omega_{n+1}(x) = \prod_{j=0}^{n} (x - x_j).$$

   Show that $Y(t)$ has at least $n+2$ roots on the interval $[a,b]$.

   (b) Using Rolle's Theorem (or the Mean Value Theorem), show that the $n+1$-st derivative $Y^{(n+1)}(t)$ has at least one root $\xi \in [a,b]$. Conclude that

   $$f^{(n+1)}(\xi)\omega_{n+1}(x) = E_n f(x)(n+1)!.$$

   (c) Show that for equidistant nodes x_i, using the transformation $x = a + (b-a)s/n$, it holds that

   $$|\omega_{n+1}(x)| \leq \left(\frac{b-a}{n}\right)^{n+1} \max_{s\in(0,n)} \prod_{j=0}^{n} |s-j|$$

   for $x \in (a,b)$

   (d) Using the fact that:

   $$\prod_{j=0}^{n} |s-j| \leq \frac{1}{4} n!$$

   show the statement in Equation (2).

B Piecewise interpolation and stability analysis

Accuracy of piecewise Lagrange interpolation

We denote by $\Pi_n^N f$ the piecewise Lagrange interpolant of order n using equidistant subintervals of size $H = (b-a)/N$, where N is the number of subintervals.

5. Sketch the piecewise Lagrange interpolant of order 1 for $f(x) = \sin(x)$ on the interval $[0, 2\pi]$ using four equidistant subintervals.

6. Sketch the piecewise Lagrange interpolant of order 2 using two equidistant subintervals.
7. Show that the piecewise Lagrange interpolant of order n satisfies the following upper bound (use Equation (2))

$$|E_n^N f(x)| = |\Pi_n^N f(x) - f(x)| \leq \frac{\max_{y\in[a,b]} |f^{(n+1)}(y)|}{4(n+1)} \left(\frac{b-a}{Nn}\right)^{n+1}. \tag{3}$$

8. How many times the function f has to be evaluated for computing the coefficients of the Lagrange interpolation of order n? Why?
9. How many times the function f has to be evaluated for computing the coefficients of the piecewise Lagrange interpolation of order n, using N subintervals?

Stability analysis

Here we consider the situation where we interpolate perturbed function values $\hat{f}(x_i) = f(x_i) + \epsilon(x_i)$, for some perturbation $\epsilon(x)$. Such a perturbation can come from e.g. measurement errors, round-off errors, approximation errors or a combination thereof.

1. Let the Lebesgue constant Λ_n be given by

   $$\Lambda_n = \max_{x\in[a,b]} \sum_{i=0}^{n} |\varphi_i(x)|.$$

   Show that the following inequality holds

   $$|\Pi_n f(x) - \Pi_n \hat{f}(x)| \leq \Lambda_n \max_{x\in[a,b]} |\epsilon(x)|,$$

   where $\Pi_n \hat{f}$ interpolates the perturbed function values $\hat{f}(x_i)$. It can be shown that for equidistant interpolatory nodes the Lebesgue constant grows 'almost exponentially fast' in n

   $$\Lambda_n \simeq \frac{2^{n+1}}{en\log n + \gamma}, \qquad e \simeq 2.7183, \qquad \gamma \simeq 0.5772.$$

2. Show that the total error satisfies the following upper bound (use Equation (2))

   $$|\Pi_n \hat{f}(x) - f(x)| \leq \frac{\max_{x\in[a,b]} |f^{(n+1)}(x)|}{4(n+1)} \left(\frac{b-a}{n}\right)^{n+1} + \Lambda_n \max_{x\in[a,b]} |\epsilon(x)|. \tag{4}$$

   Hint: use the triangle inequality.

3. Show that for piecewise interpolation we similarly find (use Equation (3))

   $$|\Pi_n^N \hat{f}(x) - f(x)| \leq \frac{\max_{x\in[a,b]} |f^{(n+1)}(x)|}{4(n+1)} \left(\frac{b-a}{Nn}\right)^{n+1} + \Lambda_n \max_{x\in[a,b]} |\epsilon(x)|. \tag{5}$$

C Integration

A piecewise Lagrange interpolant $\Pi_n^N f$ of f can be used to approximately compute the integral of f over its interpolation interval

$$\int_a^b f(x)dx \approx \int_a^b \Pi_n^N f(x)dx.$$

1. Use your previously made sketches to show how (piecewise) interpolation can be used for approximate integration.
2. Show that the antiderivative of the Lagrange polynomial $\Pi_n f$ (see (1)) is given by

$$\int \Pi_n f dx = \sum_{i=0}^{n} \frac{c_i}{n+1-i}(x-m)^{n+1-i}. \tag{6}$$

3. What is the degree of exactness of a numerical integration method? How does the degree of exactness relate to the order used in the polynomial approximation used to develop the integration method?
4. The Trapezoidal formula is given by

$$\int_a^b f dx \approx \frac{b-a}{2}(f(a)+f(b)).$$

   (a) Show that exact integration of the equidistant Lagrange interpolating polynomial of order 1 yields the Trapezoidal formula (use your answer to Question A.3 for $n=1$).

   (b) The error using the Trapezoidal formula is given by

$$E_1(f) = -\frac{(b-a)^3}{12} f^{(2)}(\xi), \quad \xi \in [a,b].$$

   Show that the degree of exactness of the Trapezoidal formula is 1.

   (c) Show that the composite Trapezoidal formula is second-order accurate in H

$$|E_1^H(f)| \leq H^2 \frac{(b-a)\max_{y\in[a,b]}|f^{(2)}(y)|}{12}.$$

5. Simpson's formula is given by

$$\int_a^b f dx \approx \frac{b-a}{6}(f(a)+4f(m)+f(b)).$$

   (a) Show that exact integration of the equidistant Lagrange interpolating polynomial of order 2 yields Simpson's formula (use your answer to Question A.3 for $n=2$)

   (b) The error using Simpson's formula is given by

$$E_2(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi), \quad \xi \in [a,b].$$

   Show that the degree of exactness of Simpson's formula is 3.

(c) Show that the composite Simpson formula is fourth-order accurate in H

$$|E_2^H(f)| \leq H^4 \frac{(b-a)\max_{y\in[a,b]}|f^{(4)}(y)|}{2880}.$$

6. Show that

$$\int_0^1 \sin^2(x)dx = \frac{1-\cos(1)\sin(1)}{2}.$$

This will be useful for computing the integration error in the Lab.

Lab session

IMPORTANT INSTRUCTIONS

- A python notebook with a skeleton code is given to you as part of the material of this unit. You can upload it in your Google Drive and work on the assignment using Google Colab or you can edit it locally in your laptop (please be aware that no instructions on the installation of python and/or on the setup of any code editor will be given to you during this course).
- Do not forget to import all the modules you need to make your code work at the very beginning of the notebook (e.g. `numpy` etc.). Try not to overwrite variables to avoid inconsistency: in case of sudden nonsense outputs, the first thing to try is restarting the runtime and run all again, then you proceed with the debugging phase. Coding simple test cases to check on each of your functions is another helpful tip for debugging your code.
- For each assignment question, you can add a formatted text cell to include (part of) your answer. Be sure to include everything you think it is necessary for the TA to understand your explanation during the discussion, e.g. references to tutorial exercises, references to specific formulas, computations, step-by-step small proves etc. Pointing at a formula with no further explanation is not enough of a justification for your answers.
- The lab is graded per pair of students. It is however expected that each of the students is capable of answering each of the questions: explain to each other what the other does not (yet) understand.
- Per discussion question, half of the points is rewarded for correctness and presentation quality of the results obtained in Lab session. The other half of the points is rewarded for understanding of the theory discussed in the Tutorial and the link with the obtained results, for this you will need your answers to the related Tutorial questions.
  The final grade Lab_1 is determined as

  $$\text{Lab}_1 = 1 + \text{Code}_1 + 8\frac{\text{Points}_1}{19} \in [1, 10],$$

  where $\text{Code}_1 \in \{0, 1\}$ rewards good quality code, namely:

  1. Descriptive variable and function names make the code easy to read and understand.
  2. Functions clearly structure and organize the code, encapsulating tasks/code segments to improve readability and proneness to errors and to enable code reusability, and break up large sections (so-called "spaghetti code").
  3. The code is well annotated with comments[*].

[*]Helpful comments are ones that help explain code segments the meaning of which is not immediately clear (e.g., from good variable/function naming), or why (as opposed to how) something has to be done. Bad comments are those that state the obvious (e.g., the comment `#loop` before a `for` loop or `#check if x is greater than y` before `if x > y:`).

4. Avoid inconsistencies in the naming of functions and variables (choose a format and stick to it, e.g., `min_max_search()` vs. `minMaxSearch()`), avoid mixing single and double quotes for strings, and inconsistent indenting of code blocks.
5. Functions MUST NOT use global variables!
6. Output formatting: printed output should be easy to read and understand, e.g.:
   ```
   Computing time of this function is:  10.3 secs
   ```
   or
   ```
   N = 10, error = 0.0012343234
   ```
   Plots need to appear with labels and legends.

A Lagrange interpolation

Implement `lagrangeInterp` which returns the centered coefficients c_i of the equidistant Lagrange interpolating polynomial $\Pi_n f$ in (1) for a given function f, an interval $[a, b]$ and polynomial order $n > 0$. For the computation of the centered coefficients you may use numpy's built-in `polyfit` by subtracting m to the values of the nodes `x` when calling the function. Verify that the values of the centered coefficients are the same you computed by hand in the tutorial for $n = 1$ and $n = 2$.

```
        ## Lagrange interpolation
        #
        # INPUT:
        # f          scalar-valued function
        # interval   interpolation interval [a,b]
        # n          interpolation order
        #
        # OUTPUT:
        # coeff      centered coefficients of Lagrange interpolant

        def lagrangeInterp(f, interval, n):
```

Consider $f(x) = \sin(\frac{x}{p})$ on the interval $[0, 2p\pi]$, with $p \in \mathbb{N}$. You are given the code of a Python function `evaluateInterpolant` which evaluates a single (or piecewise) Lagrange interpolant. In a single figure plot the function f and the interpolating polynomials $\Pi_1 f(x), \dots, \Pi_4 f(x)$ on the given interval. The TA will set a value of p during the discussion, so make your whole code dependent on p. Then make a new figure, and using `semilogy` plot the error of the interpolating polynomials $|f(x) - \Pi_1 f(x)|, \dots, |f(x) - \Pi_4 f(x)|$ as a function of x.

Discussion question A.1: ***(2 point(s), see also Tutorial: A.2)***
For each value of n, at how many points is the error between interpolant and function (very close to) zero in the plots? How can this number of points be explained from the theory?

For $\Pi_1 f, \dots, \Pi_{15} f$ compute the maximum error on the given interval, and plot that maximum error in the y-axis in terms of the polynomial order n in the x-axis using `loglog`. Also include in the plot the right-hand-side of Equation (2).

Discussion question A.2: ***(2 point(s), see also Tutorial: A.4)***
Which of both curves (true error and right-hand-side of Equation (2)) is larger? How can this be explained from the theory?

B Piecewise interpolation and stability analysis

Piecewise Lagrange interpolation

Implement a Python function `piecewiseInterp` which computes the centered coefficients (centered per subinterval) of the piecewise Lagrange interpolant $\Pi_n^N f$, together with the corresponding subintervals for a given function f, an interval $[a, b]$, a specified polynomial order n and the number of subintervals N. For $N = 1$ the coefficient output has to be equivalent to that of `lagrangeInterp`.

```python
## Piecewise interpolation
#
# INPUT:
# f              f scalar-valued function
# interval       interpolation interval [a, b]
# n              interpolation order
# N              number of subintervals
#
# OUTPUT:
# coeffs         i-th row are centered coefficients of i-th
  subinterval
# intervals      i-th row are begin and endpoint of i-th
  subinterval

def piecewiseInterp(f, interval, n, N):
```

Consider f as before and keep the code depending on p. In a single figure, plot the function f and the piecewise Lagrange interpolants of order $1, 2, 3$ and 4 using $N = 4$ subintervals. Make another figure with the logarithm of the error as a function of x, for the piecewise interpolants with $n = 1, \ldots, 4$ at a fixed number of subintervals $N = 4$.

For each order $n = 1, \ldots, 4$, plot the maximum error over the whole interval v/s the number of subintervals $N = 2^l$, for $l = 0, \ldots, 10$ using `loglog`, together with the right-hand side of Equation (3).

Discussion question B.1: ***(2 point(s), see also Tutorial: B.7)***
For fixed n, each of the error curves $E_n^N f$ should result in (approximately) a straight line as a function of N in the double logarithmic plot. Explain that in view of the theory, and in particular why the slope of that lines changes with changing n. Show the error curves for $n = 1, 2, 3, 4$ in the same plot.

Comparison

Thus far we have seen two methods for increasing the interpolation accuracy: increasing the order n, or increasing the number of subintervals N. A fair comparison between those methods considers the maximum error as function of the number of function evaluations needed. In a single figure, using `loglog`, plot the following computed errors v/s the number of required function evaluations :

- One curve for the maximum (global) Lagrange interpolation error

$$\max_{x\in[a,b]} |E_n f(x)|,$$

where $n = 1, \ldots, 20$ is varied in order to obtain different number of functions evaluations.

- One curve for each $n = 1, \dots, 4$: the maximum piecewise Lagrange interpolation error

$$\max_{x\in[a,b]} |E_n^N f(x)|,$$

where $N = 2^l$, $l = 0, \dots, 10$, is varied in order to obtain different number of functions evaluations.

Discussion question B.2: ***(3 point(s), see also Tutorial: A.4, B.7, B.9)***

(a) Which method of increasing the interpolation accuracy leads to a smaller number of function evaluations for a given accuracy? Justify your answer using the error estimation (3).

Consider now the interpolation of the perturbed function $\hat{f}(x) = f(x) + 10^{-3}\cos(\frac{8x}{p})$, $p \in \mathbb{N}$. Note that the total error is now given by $\Pi_n \hat{f}(x) - f(x)$ and $\Pi_n^N \hat{f}(x) - f(x)$ for Lagrange and piecewise Lagrange interpolation respectively. Hence you should use (4) and (5) as the corresponding upper bounds.
Create separate plots with:

- $f(x)$ and $\Pi_n \hat{f}(x)$ for $n = \{4, 20\}$.
- $f(x)$ and $\Pi_n^N \hat{f}(x)$ for $n = 1$, $N = \{4, 20\}$.
- $\max_x(\Pi_n \hat{f}(x) - f(x))$ v/s n, with $n = 1, \dots, 20$. Include also in the plot the right-hand-side of (4). Use `loglog`.
- $\max_x(\Pi_n^N \hat{f}(x) - f(x))$ v/s N, for $n = 1, 2, 3, 4$ with $N = 2^j$, j=0, …, 10. Include also in the plot the right-hand-side of (5). Use `loglog`.

Discussion question B.3: ***(5 point(s), see also Tutorial: B.1, B.2, B.3)***

(a) Describe the behavior of $\max_x(\Pi_n \hat{f}(x) - f(x))$ with respect to n and explain it in view of the theory.

(b) Describe the behavior of $\max_x(\Pi_n^N \hat{f}(x) - f(x))$ with respect to N and explain it in view of the theory.

(c) In case that you would know that the "measured" function values are perturbed, but you just know that the perturbations correspond to random variables at each node every time, and therefore they will change every time you measure them, which combination of N and n would (most likely) lead to the a more similar error when repeating the measurements? Justify your answer in view of the theory.

C Integration

We will now apply the previously developed interpolation functions to the approximate computation of an integral.

Composite integration

Given the centered coefficients c_i obtained from `piecewiseInterp` we can use (6) for the exact integration of the interpolating polynomial. Implement `compositeIntegr` which approximately computes the integral of some input function f using piecewise Lagrange interpolation of order n on N subintervals.

```python
        ## Composite integration
        #
        # INPUT
        # f                 f scalar-valued function
        # interval          interpolation interval [a, b]
        # n                 interpolation order
        # N                 number of subintervals
        #
        # OUTPUT
        # num_integral      approximate integral

        def compositeIntegr(f, interval, n, N):
```

Discussion question C.1: ***(2 point(s), see also Tutorial: C.4, C.5)***
Give a few *numerical examples* using the function `compositeIntegr` you have just implemented to show that the Trapezoidal and Simpson's rules have degree of exactness 1 and 3 respectively. Justify your answer referring to the theory.

Discussion question C.2: ***(3 point(s), see also Tutorial: C.4, C.5)***

(a) For each $n = 1, \dots, 5$: compute the approximate integral of $f(x) = \sin^2(x)$ over the interval $[0, 1]$ using $N = 2^0, \dots, 2^8$ intervals, and compute the error for each N. Plot using `loglog` the error as function of the number of subintervals for each $n = 1, \dots, 5$.

(b) What could be the degree of exactness for each of the plotted values of n based on the error curves you just computed. How could you justify your answer from the theory? How could you test numerically if your hypothesis is correct?

Please give us your feedback during and/or right after the discussion with the TA. THIS IS THE LINK TO THE FEEDBACK FORM.