EXERCISE SET 6 – MATH-250 Advanced Numerical Analysis I

The exercise sheet is divided into two sections: quiz and exercises. The quiz will be discussed in the beginning of the lecture on Thursday, April 3. The exercises marked with ($\star$) are graded homework. The exercises marked with **(Python)** are implementation based and can be solved in the Jupyter notebooks which are available on Moodle/Noto. **The deadline for submitting your solutions to the homework is Friday, April 4 at 10h15.**

Quiz

Let T_n, $n = 0, 1, 2, \ldots$ denote the Chebyshev polynomials.

(a) For $m \geq n$ it holds $T_n(x)T_m(x) = T_{m+n}(x) + T_{m-n}(x)$.

☐ True ☐ False

(b) The $(n+m)$-th derivative of $T_n(x)T_m(x)$ at $x = 0$ is

☐ 0 ☐ $2^{n+m-2}(n+m)!$

☐ $2^{n+m}(n+m)!$ ☐ $(-1)^{n+m}(n+m)!$

(c) The Chebyshev interpolant of a nonnegative function is nonnegative.

☐ True ☐ False

Exercises

Consider $n+1$ points $x_0, x_1, \ldots, x_n$. Suppose the interpolant of some data $y_0, y_1, \ldots, y_n$ at these points is $p_n(x) = \sum_{i=0}^{n} a_i x^i$. One method to determine the coefficients $a_0, a_1, \ldots, a_n$ is to solve the linear system

$$V_n \mathbf{a}_n = \mathbf{y} \tag{1}$$

where V_n is the Vandermonde matrix defined by

$$V_n = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \tag{2}$$

and

$$\mathbf{a}_n = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

To facilitate the other exercises, write a function `interpolate_data` which takes as input an array $\mathbf{y}$ of $n+1$ values $y_0, y_1, \ldots, y_n$ and an array $\mathbf{x}$ of $n+1$ points $x_1, x_2, \ldots, x_n$; creates V_n with the Python function `numpy.vander/np.vander`; and solves the linear system (1) with `numpy.linalg.solve/np.linalg.solve`; and finally returns the coefficients $\mathbf{a}_n$ of the corresponding interpolating polynomial p_n.

Write a second function, `interpolate_function`, which takes as input a function f and does the same as `interpolate_data` on $y_0 = f(x_0), y_1 = f(x_1), \ldots, y_n = f(x_n)$.

Problem 1. (Python) Consider $n+1$ points $x_0, x_1, \ldots, x_n \in [-1, 1]$, the functions

$$f^{(1)}(x) = \frac{1}{1+9x^2}, \quad f^{(2)}(x) = \sin(x),$$

and the interpolating polynomials $p_n^{(1)}$ and $p_n^{(2)}$ which interpolate $f^{(1)}$ and $f^{(2)}$, respectively, at the points $x_0, x_1, \ldots, x_n$. Suppose $p_n^{(j)}(x) = \sum_{i=0}^{n} a_i^{(j)} x^i$, $j = 1, 2$.

(a) Use the Python function `numpy.vander/np.vander` to get the Vandermonde matrix V_n. For $n = 2, 3, \ldots, 40$, plot the condition number of the Vandermonde matrix $\kappa(V_n)$ uniformly distributed interpolation nodes and Chebyshev nodes on $[-1, 1]$. As will be seen later in the course, the condition number measures the sensitivity of a linear system to roundoff error. Large condition numbers usually mean that the accuracy of the computed solution is low.

Hint: The condition number can be computed with `numpy.linalg.cond/np.linalg.cond`.

(b) For $n = 10, 20, 30, 40$ compute the coefficients $\mathbf{a}_n^{(j)}$ of the interpolants of $f^{(j)}$, $j = 1, 2$ for uniformly distributed interpolation nodes and Chebyshev nodes. Use these coefficients to plot the evaluation of $p_n^{(j)}(x)$ at 500 evenly spaced values x. Compare them to $f^{(j)}$ for $j = 1, 2$. Explain what you observe.

Hint: You can evaluate a polynomial from its coefficients with `numpy.polyval/np.polyval`.

(c) Approximate the error

$$\max_{x \in [-1,1]} |f^{(j)}(x) - p_n^{(j)}(x)|$$

by replacing the maximum in $[-1, 1]$ with the maximum at 500 evenly spaced points in $[-1, 1]$. and plot it against $n = 2, 3, \ldots, 40$ for $j = 1, 2$.

Problem 2. (Python) In this exercise we will study the stability of the Lagrange interpolation polynomial on $n+1$ uniformly distributed nodes and on Gauss-Legendre nodes. Gauss-Legendre nodes are defined to be the zeros of the Legendre polynomials q_n, which can be otained with the Python function `scipy.special.roots_legendre/sp.special.roots_legendre`. Consider the function

$$f(x) = \sin(x) + x, \quad x \in [0, 10]$$

which we will interpolate on the nodes $x_0, x_1, \ldots, x_n$. Further define $y_i = f(x_i)$ for $i = 0, 1, \ldots, n$.

(a) For $n = 1, 2, \ldots, 15$, numerically compute the Lebesgue constant Λ_n for uniformly distributed nodes and plot the result. Based on the results obtained, formulate a conjec-

ture of the asymptotic behavior of the Lebesgue constant, e.g., $O(\log^c n), O(n^c), O(c^n)$ for some constant c.

(b) Plot the function f and the interpolation polynomials for $n = 4$ and $n = 15$ for uniformly distributed nodes.

(c) For $i = 0, 1, \ldots, n$ let ε_i be independent uniformly distributed random variables in $[-0.1, 0.1]$. For each i perturb $\tilde{y}_i = y_i + \varepsilon_i$. Repeat (b) with the new data $\tilde{y}_0, \tilde{y}_1, \ldots, \tilde{y}_n$. The function `numpy.random.uniform/np.random.uniform` in Python will be useful.

(d) Repeat (a)-(c) with Gauss-Legendre nodes.

Problem 3. Consider the interpolation of the function $f(x) = x^{-3}$ on $[3, 4]$ using 4 Chebyshev nodes. Denote the interpolation polynomial $p_3(x)$.

(a) Write down the numerical values of the 4 nodes at which p_3 interpolates f.

(b) Find an upper bound for the error $|f(x) - p_3(x)|$ which is valid for any x in the interval $[3, 4]$.

(c) How many digits of accuracy will you have when p_3 is used to approximate $f(x)$?

(d) Calculate $p_3(x)$ numerically in Python and plot the graph of the error and the upper bound of the error as a function of x on a semi-logarithmic scale. Compare the interpolating polynomial obtained using the Chebyshev nodes with the one using the equispaced nodes over the interval $[3, 4]$.

Problem 4. In this exercise T_n denotes the n^{th} Chebyshev polynomial in $[-1, 1]$.

(a) Show that T_n is even if n is even and T_n is odd if n is odd.

(b) T_n is only defined in $[-1, 1]$, but using the three-term recurrence relation one can extend its definition outside $[-1, 1]$. Show that for $|x| \geq 1$ we have

$$T_n(x) = \begin{cases} \cosh(n \operatorname{arccosh}(x)), & x \geq 1; \\ (-1)^n \cosh(n \operatorname{arccosh}(-x)), & x \leq -1. \end{cases}$$

($\star$) Problem 5.

We want to use a quadrature rule to compute the integral

$$\int_0^\infty f(x) \exp(-x) \, \mathrm{d}x. \tag{3}$$

The presence of an infinite integration interval makes it impossible to directly apply a standard quadrature rule. In applications, this can be addressed by truncating the interval of (3) to $[0, T]$ for some large $T > 0$, however, there are more elegant and usually more accurate methods such as the Gauss-Laguerre quadrature rule.

The basis of the Gauss-Laguerre quadrature are the Laguerre polynomials L_n and L_{n+1} defined below. We use the roots $r_i, i = 1, 2, \ldots, n$, of L_n as quadrature nodes and define the weights as

$$w_i = \frac{r_i}{(n+1)^2 L_{n+1}(r_i)^2},$$

allowing us to write the overall quadrature rule as

$$Q_n[f] = \sum_{i=1}^{n} w_i f(x_i). \tag{4}$$

(a) The Laguerre polynomials $L_0, L_1, \ldots$ are given by the three term recurrence

$$L_0(x) = 1, \quad L_1(x) = 1 - x, \quad (n+1)L_{n+1}(x) = (2n + 1 - x)L_n(x) - nL_{n-1}(x). \tag{5}$$

Implement a Python function `laguerre(degree)` which takes the desired degree of the Laguerre polynomial and returns a NumPy `Polynomial` object equal to $L_{\texttt{degree}}$ using (5) (you can import this class from `np.polynomial.polynomial.Polynomial` or use the alias `poly` in the Jupyter notebook we provided on Moodle). Use recursion for this implementation.

Hint: When implementing multiplications like $q(x) = (1 - x) * p(x)$ in Python you need to use a separate `Polynomial` object for the $1 - x$ factor.

(b) Internally, SciPy's `roots_laguerre` function uses an eigenvalue computation to find the roots of $L_n(x)$. In analogy to Theorem 2.12 and Exercise 2 of Series 5 we can use the recurrence (5) to find a tridiagonal matrix A such that

$$A = \begin{pmatrix} a_1 & b_1 & & & & \\ b_1 & a_2 & b_2 & & & \\ & b_2 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & & b_{n-1} & a_n \end{pmatrix}$$

with the sequences $(a_1, a_2, \ldots, a_n) = (1, 3, \ldots, 2(n-1)+1)$ and $(b_1, b_2, \ldots, b_{n-1}) = (-1, -2, \ldots, -n+1)$.

Show that any root λ of L_n is an eigenvalue of A. Implement a Python function `roots(degree)` that computes the roots of L_n.

Hint: You can use SciPy's `linalg.eigvals_banded` function. In the Jupyter notebook on Moodle you can directly run `eigvals_banded`. Make sure your function returns the proper eigenvalue for degree 1.

(c) Use the roots $r_1, r_2, \ldots, r_n$ you found in (b) and verify their quality by comparing the maximum absolute value of $L_n(r_i)$ to 0. Repeat this verification for the first $M > 0$ Laguerre polynomials and plot the results in an appropriate plot. Make sure you that M is not too large.

(d) Implement a Python function `gauss_laguerre(f, num_points)` that computes the Gauss-Laguerre quadrature rule Q_n given in (4). Compute the integral for the

function $f(x) = \sin(x)$. Compute the error with respect to the exact integral $\int_0^\infty \sin(x) \exp(-x)\, \mathrm{d}x = 0.5$ for your implementation of the quadrature points and weights, and that of SciPy using `roots_laguerre`. Use $n \in$ `np.arange(1, 25)`. Plot the errors in a plot of your choice. Measure the computational time both quadratures require with the `time` function and plot the elapsed times in a semi-logarithmic plot; use the function `plt.semilogy`/ `matplotlib.pyplot.semilogy`.

The following part is for your understanding and will not be graded: Why is your implementation so much slower than that of SciPy? If you are really interested, you can have a look at the Cython library (cython.org).

Remember to upload a scan `homework06.pdf` of your solutions and the completed Jupyter notebook `homework06.ipynb` corresponding to the homework to the submission panel on Moodle until Friday, April 4 at 10h15. To download your notebook from Noto, use `File > Download`. Only your submissions to Moodle will be considered for grading.