NumCSE

Autumn Semester 2017

Prof. Rima Alaifari

Exercise sheet 9
Function Approximation

P. Bansal

Problem 9.1: Adaptive polynomial interpolation

For approximating a given function by a polynomial interpolant upto a desired tolerance, position of the interpolation nodes is crucial. Here, we look at an **a posteriori** adaptive strategy that employs a **greedy algorithm** to build the set of interpolation nodes based on intermediate interpolants.

Templates: `adapPolyIpol.cpp`, `newtonIpol.hpp`

The greedy algorithm for adaptive polynomial interpolation is described below:

> Given a function $f : [a, b] \mapsto \mathbb{R}$, start with an initial node set $\mathcal{T}_0 := \{\frac{1}{2}(b + a)\}$. Based on a fixed finite set $\mathcal{S} \subset [a, b]$ of **sampling points,** augment the set of nodes as
>
> $$\mathcal{T}_{n+1} = \mathcal{T}_n \cup \left\{ \operatorname*{argmax}_{t \in \mathcal{S}} |f(t) - \mathsf{I}_{\mathcal{T}_n}(t)| \right\}, \tag{9.1}$$
>
> where $\mathsf{I}_{\mathcal{T}_n}$ is the polynomial interpolation operator for the node set $\mathcal{T}_n$, until
>
> $$\max_{t \in \mathcal{S}} |f(t) - \mathsf{I}_{\mathcal{T}_n}(t)| \leq \texttt{tol} \cdot \max_{t \in \mathcal{S}} |f(t)| \; . \tag{9.2}$$

First, we need a function which computes the interpolating polynomial for a given set of nodes.

(a) Implement a C++ function

```
VectorXd divDiff(const VectorXd& t, const VectorXd& y);
```

which computes the coefficients of the polynomial interpolant using divided differences, refer tablet notes. Here, `t` and `y` are the interpolation nodes and the corresponding function values.

(b) Implement a C++ function

```
template <class Function>
VectorXd adapPolyIpol(const Function& f, double a, double b,
                      double tol, int N,
                      Eigen::VectorXd& adaptive_nodes);
```

that implements the algorithm described above. The arguments are: the function handle `f`, the interval bounds `a`, `b`, the relative tolerance `tol`, the number `N` of *equidistant* sampling points to compute the error:

$$\mathcal{S} := \left\{ a + (b - a)\frac{j}{N}, \; j = 0, \ldots, N \right\} .$$

and the final set of interpolation nodes returned in `adaptive_nodes`. For each intermediate set $\mathcal{T}_n$, `adapPolyIpol` should compute the error:

$$\epsilon_n := \max_{t \in \mathcal{S}} |f(t) - \mathsf{T}_{\mathcal{T}_n}(t)| \tag{9.3}$$

and return these error values in a vector.

Remark: Use a suitable lambda function for the type `Function` and use the function `intPolyEval` defined in `newtonIpol.hpp` to evaluate the polynomial interpolant.

(c) For $f_1(t) := \sin(e^{2t})$ and $f_2(t) = \frac{\sqrt{t}}{1+16t^2}$ plot ϵ_n versus n, the number of interpolation nodes, in the interval $[a, b] = [0, 1]$ using `N=1000` sampling points and `tol = 1e-6`.

Problem 9.2: Chebyshev polynomials and their properties

In this problem, we will examine Chebyshev polynomials and a few of their many properties.
Templates: `bestApproxCheb.cpp`

Let $T_n \in \mathcal{P}_n$ be the n-th Chebyshev polynomial and $\xi_0^{(n)}, \ldots, \xi_{n-1}^{(n)}$ be the n zeros of T_n, where

$$\xi_j^{(n)} = \cos\left(\frac{2j+1}{2n}\pi\right), \quad j = 0, \ldots, n-1. \tag{9.4}$$

Define a family of discrete L^2 semi-inner products (i.e. not conjugate symmetric):

$$(f,g)_n := \sum_{j=0}^{n-1} f(\xi_j^{(n)}) g(\xi_j^{(n)}), \quad f,g \in C^0([-1,1]) \tag{9.5}$$

Also define a special weighted L^2 semi-inner product:

$$(f,g)_w := \int_{-1}^{1} \frac{1}{\sqrt{1-t^2}} f(t)g(t)\,dt \quad f,g \in C^0([-1,1]) \tag{9.6}$$

(a) Show that the Chebyshev polynomials are orthogonal w.r.t. the semi-inner product defined in Eq. (9.6), i.e. $(T_k, T_l)_w = 0$ for every $k \neq l$.

Hint: Use the trigonometric identity $2\cos(x)\cos(y) = \cos(x+y) + \cos(x-y)$.

Consider the following statement:

Theorem 9.7.

The family of polynomials $\{T_0, \ldots, T_n\}$ is an orthogonal basis of $\mathcal{P}_n$ with respect to the inner product $(\cdot,\cdot)_{n+1}$ defined in Eq. (9.5).

(b) Prove Thm. 9.7. **Hint:** Use the relationship of trigonometric functions and the complex exponential.

(c) Implement a C++ code to numerically test the assertion of Thm. 9.7.

Use the following code for evaluating Chebyshev polynomials based on their recursive definition:

C++11-code 9.1: Evaluate Chebyshev polynomials

```cpp
vector<double> chebPolyEval(const int &n,const double &x)
{
    vector<double> V={1,x};
    for (int k=1; k<n; k++)
        V.push_back(2*x*V[k]-V[k-1]);
    return V;
}
```

(d) Given a function $f \in C^0([-1,1])$, find the best approximation $q_n \in \mathcal{P}_n$ of f in the discrete L^2-norm:

$$q_n = \underset{p \in \mathcal{P}_n}{\operatorname{argmin}} |f - p|_{n+1} ,$$

where $|v|_{n+1} = \sqrt{(v,v)_{n+1}}$ for any v. Represent q_n as an expansion in Chebyshev polynomials:

$$q_n = \sum_{j=0}^{n} \alpha_j T_j \,, \tag{9.8}$$

for suitable coefficients $\alpha_j \in \mathbb{R}$. The task boils down to determining the coefficients α_j.

(e) Implement a C++ function that returns the vector of coefficients $(\alpha_j)_j$ in Eq. (9.8) given a function f:

```
template <typename Function>
void bestApproxCheb(const Function &f, Eigen::VectorXd &alpha)
```

Note that the degree of the polynomial is indirectly passed with the length of the output `alpha`. The input `f` is a lambda-function, example:

```
auto f = [] (double & x) {return 1/(pow(5*x,2)+1);};
```

(f) Test `bestApproxCheb` for the function $f(x) = \frac{1}{(5x)^2+1}$ and $n = 20$. Approximate the supremum norm of the approximation error by sampling on an equidistant grid with 10^6 points.