Numerical Differentiation using local
Chebyshev-Approximation

Stefan H. Reiterer
February 5, 2021

Abstract

1 Motivation

In applied mathematics, especially in optimization, functions are often only provided as so called
"Black-Boxes" provided by software packages, or very complex algorithms, which make automatic
differentiation very complicated or even impossible. Hence one seeks the numerical approximation
of the derivative.

Unfortunately numerical differentiation is a difficult task in itself, and it is well known that
it is numerical instable. There are many works on this topic, including the usage of (global)
Chebyshev approximations. Chebyshev approximations have the great property that they con-
verge very fast, if the function is smooth. Nevertheless those approaches have several drawbacks,
since in practice functions are not smooth, and a global approximation needs many function
evaluations.

Nevertheless there is hope. Since functions in real world applications are most times smooth
except for finite points, corners or edges. This motivates to use a local Chebyshev approach,
where the function is only approximated locally, and hence the Chebyshev approximations still
yields a fast approximation of the desired function. We will study such an approach in this work,
and will provide a numerical example.

Disclaimer: This work was done as private research during my study years and is not related
to my current affiliation at all. No funding was received for this work.

2 Chebyshev Polynomials and One Dimensional Functions

First we recall some well known definitions. the following definitions and properties are taken
from [1].

Definition 2.1 (Chebyshev-Polynomials). For $n \in \mathbb{N}_0$ the $n$-th Chebyshev polynomial $T_n(x)$ :
$[-1,1] \to \mathbb{R}$ is given by $T_n(x) = \cos(n \arccos(x))$. From this definition we see that the Chebyshev
polynomial takes its extrema at

$x_k := -\cos\left(\frac{k\pi}{n}\right)$ for $k = 0,\ldots,n$.

The points $(x_k)_{k=0}^n$ are the so called Gauss-Lobatto Grid-points.
The best way to approximate functions via Chebyshev-polynomials lies in the following

**Proposition 2.2** (Chebyshev-Series & Co). Set \( \omega(x) := \frac{2}{\pi} (1 - x^2)^{-1/2} \), then

\[
\langle f, g \rangle_\omega := \int_{-1}^{1} f(x)g(x)\omega(x) \, dx
\]

forms the (weighted) scalar product of the Hilbert-Space

\[
L_{2,\omega} := \left\{ f : [-1, 1] \to \mathbb{R} \mid \|f\|_{L_{2,\omega}} = \sqrt{\langle f, f \rangle_\omega} < \infty \right\}.
\]

Then for a function \( f \in L_{2,\omega} \) the Chebyshev-Coefficients

\[
a_n := \langle f, T_n \rangle_\omega,
\]

exists, and the Chebyshev-Series

\[
a_0 + \sum_{n=1}^{\infty} a_n T_n(x),
\]

converges in the \( L_{2,\omega} \) sense to \( f \).

**Proof.** These well known facts follow from applying classical Fourier-theory to

\( \hat{f} = f(\cos(t)) \in L_2(0, \pi) \).

Since Chebyshev-Polynomials are “cosines in disguise” other important properties from Fourier-theory carry over to Chebyshev-series, like spectral convergence for smooth functions etc.

Another way to approximate a function is to use the Chebyshev-Interpolation Polynomial

\( C_n = \sum_{k=0}^{n} b_k T_k \), which is uniquely defined by \( f(x_k) = C_n(x_k) \) for \( k = 0, \ldots, n \).

Although the Chebyshev-Series yields the best approximation there is a little known fact between the two approximation types, namely the

**Proposition 2.3.** Let \( f : [-1, 1] \to \mathbb{R} \) a function with

\[
f(x) = a_0 + \sum_{n=1}^{\infty} a_n T_n(x),
\]

and

\[
\left| a_0 \right| + \sum_{n=1}^{\infty} |a_n| < \infty.
\]

Further let for \( f_N(x) := a_0 + \sum_{n=1}^{N} a_n T_n(x) \),

\[
E_T(N) := \sup_{x \in [-1, 1]} |f(x) - f_N(x)|,
\]

be the truncation error, and Then

\[
E_T(N) \leq \sum_{n=N}^{\infty} |a_n|,
\]

2
and for the interpolation error

\[ \sup_{x \in [-1, 1]} |f(x) - C_N(x)| \leq 2E_T(N). \]

Hence the penalty for using interpolation instead of truncation is at most a factor of two! Additionally the coefficients \( b_k \) of the interpolation polynomial \( C_n \) are related to the exact coefficients \( a_k \) by the identity

\[
b_k = a_k + \sum_{j=1}^{\infty} a_{k+2jn} + a_{-k+2jn}. \tag{1}
\]

That means the approximated coefficients differ from the exact coefficients by the aliasing-error, and hence the error vanishes with \( O(a_n) \).

**Proof.** See [1, Thm. 6 & Thm. 21].

Since with Chebyshev interpolation we are close to the realm of the DFT, additional methods from signal processing, like denoising could be considered to handle numerical distortions.

To work with a function \( g : [a, b] \to \mathbb{R} \) a generalized Chebyshev interpolation for \( g \) can be achieved by using a linear transformation \( \varphi : [a, b] \to [-1, 1] \), and interpolate the function \( f = g \circ \varphi \).

Since computing the derivative of a polynomial can be done exactly we could use the derivative of \( C_N \) for some \( N \), as the numerical derivative of \( f \). This means

\[
f'(x) \approx C_N'(x).
\]

Then how about the errors? First recall the identity \( T'_n(x) = nU_{n-1} \). Hence for the truncation error we have

\[
f'(x) - f'_N(x) = E'_N(x) = \sum_{n=N}^{\infty} n a_n U_{n-1}(x).
\]

Considering that \( U_n \) has it’s extrema at \( \pm 1 \) with \( U_{n-1}(\pm 1) = (\pm 1)^n n \), we immediately see that

\[
\sup_{x \in [-1, 1]} |f'(x) - f'_N(x)| \leq \sum_{n=N}^{\infty} n |a_n| \sup_{x \in [-1, 1]} |U_{n-1}(x)| \leq \sum_{n=N}^{\infty} n^2 |a_n| = O(n^2 a_n).
\]

For the interpolation polynomial \( C_N \) we get by using the aliasing identity [1] and rearranging terms like in the proof of [1, Thm. 21] the error estimation

\[
|f'(x) - C'_N(x)| \leq 2 \sum_{n=N}^{\infty} n^2 |a_n| = O(2n^2 a_n).
\]

Hence we have the

**Proposition 2.4.** For \( f : [-1, 1] \to \mathbb{R} \) smooth we have the error estimation

\[
\sup_{x \in [-1, 1]} |f'(x) - C_N'(x)| \leq 2E_T'(N)O(2n^2 a_n).
\]

Hence the penalty for using interpolation instead of truncation when differentiating is at most a factor of two!
3 Practical Considerations in the 1D Case and Generalizations

In this section consider a function $f : [a, b] \rightarrow \mathbb{R}$, which is continuous and piecewise smooth, but not differentiable in a set of finite points $(y_k)_{k \in I} \subset [a, b]$ (with $I$ finite and allowed to be empty).

If we interpolate now the function $f$ on the interval $[a, b]$, we know from Fourier-Theory, that the error will converge only slowly to zero. We also note that it is easy to see that (from variable transformations) that error estimation depends on the length of the interval $[a, b]$. with the factor $(\frac{b-a}{2})^N$. Hence if one restricts the function $f$ only on a small interval, one achieves more precise interpolation results, with lesser degrees of the interpolation polynomial. If we restrict the function $f$ to some interval $[c, d] \subset [a, b]$, with

1. $d-c < 1$ and
2. $y_i \notin (c,d)$ for $i \in I$,

then the sequence of the Chebyshev interpolation polynomials $(C_N)_{N \in \mathbb{N}}$ converges very fast (with spectral convergence) and uniformly to the original function in the interval $[c, d]$.

From Proposition 2.4 we know that this then also applies to $C'_N$, since the error decays with $O(n^2 a_n)$. Nevertheless we also see that for practical use it is important to ensure the smoothness on the observed interval $[c, d]$, because of the term $n^2$. This leads to the following definition

**Definition 3.1.** Let $f : [a, b] \rightarrow \mathbb{R}$ be continuous and piecewise smooth, then we call the Chebyshev interpolation polynomial $C_{N,[c,d]} : [c,d] \rightarrow \mathbb{R}$ of $f|_{[c,d]}$ the local smooth Chebyshev polynomial iff

$$(y_k)_{k \in I} \cap (c,d) = \emptyset.$$

This motivates the following algorithm to compute derivatives of a function $f$ (if the points $Y := (y_k)_{k \in I}$ are known):

```python
# input: function f, and point x, estimated length h
# set of disallowed points Y, number of interpolation points N
def compute_derivative(f, x, h, Y):
    if x not in Y:
        # classical derivative
        c = x - h; d = x + h
        # ensure that we are smooth on interval [c,d]
        while c < a and b < d and intersection(Y, [c,d]) != {}:
            h = make_h_smaller(h)
            c = x - h; d = x + h
        CN = compute_interpolation(f, c, d, N)
        CN_prime = derive_polynomial(CN)
        return CN_prime(x)
    else:
        # subgradient
        c[0] = x - h; d[0] = x
        c[1] = x; d[1] = x + h
        while all([c[i] < a and b < d[i] and
                   intersection(Y, (c[i], d[i])) != {} for i in [0, 1]]):
            h = make_h_smaller(h)
            c[0] = x - h; d[0] = x
            c[1] = x; d[1] = x + h
            CN = compute_interpolation(f, c, d, N)
            CN_prime = derive_polynomial(CN)
            return CN_prime(x)
```

This concludes the 1D case and generalizations.
As one can see we also included more general derivatives based on directional derivatives like sub-gredients. It also would be possible to define a weak derivative witch is motivated by Chebyshev convergence theory defined by 

\[ f'(x) := \frac{1}{2} (f'(x-h) + f'(x+h)). \]

A question which still remains, is what to do when the set \( Y \) is unknown. This is still an open problem, but since one can observe the speed of the decay of the coefficients \( a_n \) (or \( b_n \)) it is possible to locate non differentiable points. There are several works on this topic from spectral theory.

Now to the \( M \) dimensional case: Consider a functional \( f : \Omega \subseteq \mathbb{R}^M \to \mathbb{R} \), which is continuous and piecewise smooth on the domain \( \Omega \). We can compute the directional derivatives in a direction \( h \in \mathbb{R}^M \) in a point \( x \in \Omega \) by deriving the one dimensional function \( g(t) := f(x + th) \). Hence the methods from the one dimensional case, can be carried over to the \( M \) dimensional setting.

4 Examples

as first we define the function \( f_1 \) given by (see also Figure 4):

\[ f_1(x) = \begin{cases} 
  x^4 & \text{if } x > 0, \\
  0 & \text{else}. 
\end{cases} \]

First we compare computing the first derivative with local Chebyshev approximation with finite differences given by

\[ f'(x) \approx \frac{f(x + h) - f(x - h)}{2h} =: f_h(x), \]

by computing the values at \( x = 0 \) and \( x = 0.5 \). See Table 4 for results. As we can observe the errors of \( f_h \) and \( C_{3.3h} \) are equal. This comes without surprise, since the derivative of the 3rd Chebyshev polynomial is exactly the central difference quotient. Next we will modify \( f_1 \) in the following way: For some \( \varepsilon > 0 \), and a randomly standard normal distributed variable \( X \) we define \( f_2 \) by

\[ f_2(x) = f_1(x) + \varepsilon X. \]
and then try to differentiate it at 0.5. The results can be seen at Table 4. The errors are maximal values after taking some samples. There is some space for improvement here, e.g., better statistics, but we get the picture... We observe the following behavior: While $f_h$ and $C_{3,\pm h}$ behave like expected, we see that making $h$ smaller, does not make the approximation quality of $C_{5,\pm h}$ and $C_{7,\pm h}$ better, but even worse. This can be explained by the fact, that for smooth $f$, we know that the function behaves like its Taylor polynomial of lower order, while the higher order terms are negligible. This also means that computing a higher order Chebyshev polynomial does not make sense, and we even get into more trouble. Hence when choosing a suitable $h$ for the Chebyshev approximation should be not too small, and yet not too big.

Finally we try the Local Chebyshev Method on the 2D Rosenbrock function

$$R_{a,b}(x) = (a - x_0)^2 + b(x_1 - x_0^2)^2,$$

with parameters $a = 1$ and $b = 100$, and try to find the minimum with help of gradient methods. The argmin is $x_0 = (1, 1)$ Additionally we add some disturbances $\delta(x)$ and $\varepsilon(x)$ in the interval $[-1e^{-6}, 1e^{-6}]$. While $\delta$ is some randomly normal distributed variable, $\varepsilon$ is a jump function which changes it’s sign (which is fatal for finite differences). As optimization method we used Steepest Descent with Armijo-rule and $\|\nabla f\| < 1e^{-3}$ as terminating criterion. We used $h = 1e^{-6}$ for the finite difference method, and order 5 polynomials with $h = 1e^{-4}$ for the Local Chebyshev Method. The number of iterations and the computed results (rounded up to 3 digits) can be seen in Table 4. The results suggest, that computing the gradient with the Local Chebyshev Method makes the algorithm more robust. Also it was observed that the stepsize $h$ should be

| $x$  | $h$    | $f_h$ | $C_{3,\pm h}$ | $C_{5,\pm h}$ | $C_{7,\pm h}$ |
|-----|--------|-------|---------------|---------------|---------------|
| 0.5 | $1e^{-1}$ | $2e^{-2}$ | $2e^{-2}$ | $5.5e^{-9}$ | $4.2e^{-8}$ |
| 1e-2 | $2.4e^{-6}$ | $2e^{-4}$ | $4e^{-7}$ | $8e^{-7}$ |
| 1e-3 | $2.4e^{-6}$ | $2e^{-6}$ | $1.8e^{-6}$ | $7.3e^{-6}$ |
| 1e-4 | $1e^{-6}$ | $4e^{-6}$ | $2e^{-5}$ | $2e^{-5}$ |
| 1e-5 | $2.8e^{-6}$ | $9e^{-5}$ | $2e^{-5}$ | $7e^{-4}$ |
| 1e-6 | $7e^{-5}$ | $1e^{-3}$ | $4e^{-5}$ | $2e^{-3}$ |

Table 2: Errors for $f_2$
| Function     | Method          | Iteration Numbers | Result         |
|--------------|-----------------|-------------------|----------------|
| $R_{a,b}$    | Exact Gradient  | 1427              | (0.999, 0.998) |
|              | Finite Differences | 1427           | (0.999, 0.998) |
|              | Local-Chebyshev | 1428              | (0.999, 0.998) |
| $R_{a,b} + \delta$ | Finite Differences | 19999 (interrupted) | (0.986, 0.973) |
|              | Local-Chebyshev | 1090              | (0.996, 0.992) |
| $R_{a,b} + \epsilon$ | Finite Differences | 19999 (interrupted) | (0.486, 0.234) |
|              | Local-Chebyshev | 1565              | (0.9999, 0.998) |

Table 3: Iteration numbers for optimization

not too small.

References

[1] Boyd, John P. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.