Jan Westerdiep

May 6, 2014

Consider a compact domain $D \subset \mathbb{R}$, and a function $f : D \to \mathbb{R}$. Our goal is to *quickly* find *good* (piecewise) polynomial approximations to f. This immediately raises questions: what constitutes a good approximation, when is finding it considered quick, and how will we partition our domain? These are all valid questions which will be answered.

Part I

Theory

Chapter 0

Preliminaries

0.1 Norms

0.2 Lebesgue spaces

0.3 Limiting behaviour of functions: $\mathcal{O}$

0.4 Trees

Definition 1. A *tree* is a connected graph $G = (T, E)$ without any cycles. A *tree structure* is a way of representing the hierarchical nature of a structure in graphical form.

A vertex in a tree is often called a *node*. Sometimes, a special node exists – the *root node*, D. Trees with root nodes are called *rooted trees*.

Definition 2. In a rooted tree, each node has a *depth* d – the length of the path to the root. Given a node $\Delta \neq D$, one can define the *parent* to be the node $\Delta^+ \in T$ with $d(\Delta) = d(\Delta^+)+1$ such that $\{\Delta, \Delta^+\} \in E$. On the other hand, the *children* of Δ are all nodes that have Δ as their parent. Nodes that share a parent are called *siblings*. Nodes without children are called *leaves* (with $\mathcal{L}(T)$ denoting the set of leaves of T) and nodes that are not leaves are called *inner nodes* (with $\mathcal{I}(T)$ the set of inner nodes).

Lemma 1. *Every node except for D has a parent.*

Remark. A rooted tree immediately implies a certain hierarchy in which nodes with equal depth have equal rank.

With the notion of trees properly introduced, we can now further enlarge our toolset.

Definition 3. A *subdivision rule* s is a non-node-specific way to partition a given polygon $\Delta \subset D$ into 2 elements.

Example. Let $D = [0, 1] \subset \mathbb{R}$. One subdivision rule is to divide an interval in half: $[a, b]$ becomes $\{[a, \frac{a+b}{2}], [\frac{a+b}{2}, b]\}$.

Given a subdivision rule and a domain D, one can recursively construct an (infinite) tree $\mathcal{T}^*$ with root node D. In each step, take the set of leaves $\mathcal{L}(T_n)$ and create $T_{n+1} = T_n \sqcup s(\mathcal{L}(T_n))$. Taking the limit to infinity yields $\mathcal{T}^*$.

All trees generated by iterated use of the subdivision rule are subsets[1] of $\mathcal{T}^*$, and they all share a trait.

Lemma 2. *For a tree T generated by iterated use of the subdivision rule, it holds that $\mathcal{L}(T)$ – the set of leaves of T – is a partition of D.*

Definition 4. It is useful to point out that trees T of the above type can create *adaptive* partitions of D. This means that in some regions of D, a higher concentration of elements might be present. This means that we can *adapt* our generation algorithm to the problem at hand, creating a well-suited partition for this particular problem.

0.5 Polynomial approximation in one dimension

Given an interval $D = [a, b] \subset \mathbb{R}$ and a function $f : D \to \mathbb{R}$, one might wonder how to find good approximations to f within some function class. This question has no single answer, as both the space and the notion of a good approximation is as of yet undefined. One commonly used tactic to quantify a good approximation is to find a norm well-suited to your intentions.

To further narrow our search, we will only use a very special class of functions with interesting properties.

Definition 5. A polynomial is a function $p : \mathbb{R} \to \mathbb{R}$ of the following form:

$$p(x) = \sum_{k=0}^{n} a_k x^k.$$

The values a_k are the *coefficients* of p, and the largest power of x with nonzero coefficient is called the *degree* of p. The set of all polynomials is denoted by $\mathcal{P}$. When constrained to a domain $D \subset \mathbb{R}$, $\mathcal{P}(D) := \{q_{|D} : q \in \mathcal{P}\}$ denotes the set of all polynomials from D to $\mathbb{R}$.

These polynomials are useful instruments in dealing with function approximation. Given an interval $D = [a, b] \subset \mathbb{R}$ and a Lebesgue space $L^p(D)$, with $1 \leq p \leq \infty$, the following results hold.

Theorem 1. *For $1 \leq p < \infty$, the set $\mathcal{P}(D)$ is dense in $L^p(D)$.*

Proof. Our interval D is closed and bounded, thus by [1, Thm. 1.35] compact. Then by [1, Thm. 1.40], the set $\mathcal{P}(D)$ is dense in $C(D)$, the set of continuous functions from D to $\mathbb{R}$. Lastly, by [1, Thm. 1.62], $C(D)$ is dense in $L^p(D)$. Transitivity of denseness implies the result. □

Theorem 2 ([2, Thm. 8.1])**.** *The set $\mathcal{P}(D)$ is dense in $C(D)$ with respect to the ∞-norm $\|\cdot\|_\infty$.*

Remark. In the above theorem, we deliberately did not state that $\mathcal{P}(D)$ lies dense in $L^\infty(D)$! This would imply that $\overline{C(D)} = L^\infty(D)$. To see this cannot be the case, consider $D = [-1, 1], f(x) = \operatorname{sgn}(x)$. Any continuous function g with $\|f - g\|_\infty < \frac{1}{3}$ must have $g(x) < f(x) + \frac{1}{3} = -\frac{2}{3}$ for $x < 0$. Analogously, $g(x) > \frac{2}{3}$ for $x > 0$. By continuity, $g(0)$ must satisfy both $g(0) \leq -\frac{2}{3}$ and $g(0) \geq \frac{2}{3}$. Such a function cannot exist, and so we cannot get arbitrarily "close" to f using only continuous functions.

[1] Strictly speaking, a tree is an ordered pair (T, E). Because the edges are in our case easy to determine (by the implied parent-child relation), we will loosen our terminology and say that a tree *equals* its set of vertices T, in which case $T \subset \mathcal{T}^*$ implies that the assertion holds.

These two results help to show the vast importance of polynomial functions. The above norms are all interesting, with two elements standing out. The first is (perhaps) the most intuitive way of defining two functions to be "close", and the second holds a very special property.

0.5.1 ∞-norm

Given an $\epsilon > 0$ and a continuous f, we found with Theorem 2 that it is possible to find a polynomial p such that

$$\|f - p\|_\infty := \sup\{|(f-p)(x)| : x \in D\} < \epsilon.$$

If we however restrict ourselves to just polynomials of degree n or less (denoted by $\mathcal{P}^n$), this result no longer holds. It is therefore interesting to look at the lowest value that *is* obtainable, or more precisely:

$$\begin{gathered}\text{Given } f \in C(D) \text{ and } n \geq 0, \text{ find } p_n \in \mathcal{P}^n \text{ such that} \\ \|f - p_n\|_\infty = \inf_{q \in \mathcal{P}^n} \|f - q\|_\infty. \end{gathered} \tag{1}$$

Stelling 3.32 uit LinAna boek is zelfs nog breder: elke niet-lege convexe deelverz voldoet

Theorem 3 ([2, Thm. 8.2])**.** *The equality in* (1) *is attained for a polynomial of degree n, i.e. the infimum is a minimum.*

Because this p_n *minimizes* the *maximal* absolute value of $f(x) - q(x)$, this polynomial is often referred to as the *minimax polynomial*.

Theorem 4 ([2, Thm. 8.5])**.** *There is exactly one minimax polynomial p_n of degree n for every f on D.*

While we haven't shown the proof of Theorem 3 here, the reader is invited to read it and conclude that this proof is not *constructive* in the sense that it gives no *algorithm* to find the minimax polynomial. This makes for lesser practical use, as an implementation would like to actually *find* this polynomial. The most widely used solution is an *iterative* algorithm[2] – the Remez Algorithm [7] – that can find minimax approximations under certain conditions such as continuity of f. However, we will not look into this in more detail.

uitleggen waarom dit interessant is: orthogonaliteit enzo

0.6 2-norm

As stated earlier in §0.2, the space L^2 of twice-integrable functions holds a very special property, in the sense that it is the only Hilbert space of this family. As earlier, our set of polynomials $\mathcal{P}$ is dense in $L^2(D)$. When we confine ourselves to using polynomials of degree n or less, we get an analogous problem:

$$\begin{gathered}\text{Given } f \in L^2(D) \text{ and } n \geq 0, \text{ find } p_n \in \mathcal{P}^n \text{ such that} \\ \|f - p_n\|_2 = \inf_{q \in \mathcal{P}^n} \|f - q\|_2. \end{gathered} \tag{2}$$

Theorem 5 ([2, Thm. 9.2])**.** *Given $f \in L^2(D)$, there exists a unique polynomial $p_n \in P^n$ such that $\|f - p_n\|_2 = \inf_{q \in P^n} \|f - q\|_2$.*

[2] An iterative algorithm creates a sequence of improving approximate solutions to some problem.

Contrasting the previous: in the L^2 case, we can easily find this *polynomial of best approximation* $p_n(x) = c_0 + \ldots + c_n x^n$.

Proof (of Theorem 5). Since the 2-norm is nonnegative and $\mathbb{R}_+ \ni \xi \mapsto \xi^{1/2}$ is monotonic increasing, we can also minimise $\|f - p_n\|_2^2$ instead:

$$\begin{aligned}\|f - p_n\|_2^2 &= \int_0^1 [f(x) - p_n(x)]^2 dx \\ &= \int_0^1 [f(x)]^2 dx - 2\sum_{j=0}^n c_j \int_0^1 f(x)x^j dx + \sum_{j=0}^n \sum_{j=0}^n c_j c_k \int_0^1 x^{k+j} dx. \end{aligned} \tag{3}$$

When viewing $\|f - p_n\|_2^2$ as a function of $c_0, \ldots, c_n$ to $\mathbb{R}$, attaining the minimum implies that the gradient is zero. Therefore, its partial derivative wrt. c_l must be zero as well:

$$0 = \frac{\partial \|f - p_n\|_2^2}{\partial c_l} = -2\int_0^1 f(x)x^l dx + 2\sum_{k=0}^n c_k \int_0^1 x^{k+l} dx$$

so

$$\int_0^1 f(x)x^l dx = \sum_{k=0}^n c_k \int_0^1 x^{k+l} dx \tag{4}$$

for all l. This leads to a system of $n+1$ linear equations:

$$\sum_{k=0}^n M_{jk} c_k = b_j, \quad j = 0, \ldots, n, \quad M_{jk} = \int_0^1 x^{k+j} dx = \frac{1}{k+j+1}, \quad b_j = \int_0^1 f(x)x^j dx.$$

The matrix M_{jk} has nonzero determinant [5], so this system has a unique solution $(c_0, \ldots, c_n)$. □

The matrix M_{jk} is often referred to as the Hilbert matrix. There is however a slight problem with this construction: as we are solving a linear system, we are effectively computing the inverse of the Hilbert matrix. This inverse has *huge* elements and even for only slightly large n, is not able to be represented by 64-bit integers. To overcome this problem, we will look at another method, namely *orthogonal polynomials*.

0.6.1 Orthogonal polynomials

Definition 6. A sequence of polynomials $\{\varphi_j : j = 0, 1, \ldots\}$ is called a *system of orthogonal polynomials* on (a, b) if each φ_j is of degree j and

$$\int_a^b \varphi_k(x)\varphi_j(x)dx = 0 \text{ if } k \neq j.$$

If, in addition, this integral equals δ_{jk}, then the system is called *orthonormal*.

We will show that a system of orthogonal polynomials exists on any interval (a, b).
Let $\varphi_0(x) = 1$ and suppose φ_j has already been constructed for j up to $n \geq 0$. Then

$$\int_a^b \varphi_k(x)\varphi_j(x)dx = 0, \quad k \in \{0, \ldots, j-1, j, \ldots, n\}.$$

Define

$$q_{n+1}(x) = x^{n+1} - a_0\varphi_0(x) - \cdots - a_n\varphi_n(x), \quad a_j = \frac{\int_a^b x^{n+1}\varphi_j(x)dx}{\int_a^b \varphi_j^2(x)dx}.$$

It follows that

$$\begin{aligned}\int_a^b q(x)\varphi_j(x)dx &= \int_a^b x^{n+1}\varphi_j(x)dx - a_j\int_a^b \varphi_j^2(x)dx \\ &= 0, \quad 0 \le j \le n\end{aligned}$$

by using orthogonality of the sequence φ_j. With this choice of a_j we have ensured that q_{n+1} is orthogonal to all previous members of the sequence, and φ_{n+1} can be defined as any nonzero multiple of q_{n+1}. This procedure is called *Gram-Schmidt orthogonalisation.*

Theorem 6. *Given an orthogonal sequence $\{\varphi_j\}$ over (a,b) and a function $f \in L^2(a,b)$, the polynomial*

$$p_n(x) = \gamma_0\varphi_0(x) + \cdots + \gamma_n\varphi_n(x), \quad \gamma_n = \frac{\int_a^b f(x)\varphi_j(x)dx}{\int_a^b \varphi_j^2(x)dx}$$

is the unique polynomial of best approximation of degree n to f.

Remark. We conclude that the hurdle of inverting the Hilbert matrix is now overcome. In fact, we do not even have to invert *any* matrix any more and rely solely on the sequence of orthogonal polynomials.

je inverteert impliciet wel een matrix. uitzoeken

0.6.2 Legendre polynomials

We wish to create an orthonormal sequence for the interval $(-1,1)$. Beginning with $P_0(x) = 1$, we find $P_1(x) = q_1(x) = x$. Continuing on, we find

$$\begin{aligned}P_0(x) &= 1, \\ P_1(x) &= x, \\ P_2(x) &= \frac{1}{2}(3x^2 - 1), \\ P_3(x) &= \frac{1}{2}(5x^3 - 3x).\end{aligned}$$

These polynomials are called the *Legendre polynomials* and play a main role in the implementation of the *polynomial of best approximation.*

Replacing x by

$$\frac{2}{b-a}x + \frac{a+b}{a-b}$$

yields a sequence of orthogonal polynomials $\{\tilde{P}_n\}$ over (a,b) which we will call the *shifted Legendre polynomials.*

The following results will be used in the implementation.

Theorem 7 ([8, p. 743])**.** *The Legendre polynomials P_n satisfy the following recurrence relation:*

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$

Theorem 8 ([1, Ex. 3.28])**.** *The nth Legendre polynomial satisfies the following equality:*

$$\int_{-1}^{1} P_n(x)^2 dx = \frac{2}{2n+1}.$$

More generally, the nth shifted Legendre polynomial satisfies

$$\int_{a}^{b} \tilde{P}_n(x)^2 dx = \frac{b-a}{2n+1}.$$

Chapter 1

One-dimensional tree generation

Given r fixed, say we want to approximate a given function f over some interval $D \subset \mathbb{R}$. The previous sections provided us with two tools: firstly, with the subdivision rule s in hand, we can generate a plethora of partition trees T (recall that the leaves $\mathcal{L}(T)$ of T must always partition D – see Lemma 2). On each element of this partition, we can approximate f by a polynomial in $\mathcal{P}^r$ to create a piecewise polynomial approximation on D.

On each of these leaves Δ in $\mathcal{L}(T)$, we can find the polynomial of best approximation p_Δ, given that we 'know' how to find such a polynomial. The next definition specifies this.

Definition 7. The function $e : \mathcal{T}^* \to \mathbb{R}_{\geq 0}$ is a function assigning the best obtainable error by approximating f on $\Delta \in \mathcal{T}^*$ with functions in $\mathcal{P}^r$. If this function further satisfies the *subadditivity rule*

$$e(\Delta) \geq e(\Delta') + e(\Delta''), \quad \{\Delta', \Delta''\} = s(\Delta),$$

then it is called an *error functional*.

Example. A common choice of e is

$$e(\Delta) := \|f - p_\Delta\|_{2,\Delta}^2 = \min_{q \in \mathcal{P}^r} \|f - q\|_{2,\Delta}^2,$$

the square of the difference in 2-norm of f with the best polynomial approximation over Δ. This square is needed to ensure satisfaction of the subadditivity rule.

Definition 8. Given an error functional e, we can define the *total error* $E(T)$ of a tree as

$$E(T) := \sum_{\Delta \in \mathcal{L}(T)} e(\Delta).$$

Remark. Note that because of the subadditivity rule, it must hold that

$$T_1 \supset T_2 \implies E(T_1) \leq E(T_2).$$

This means that we can never increase our total error subdividing a node.

Given the functionals e and E and $m \geq 1$, consider the class $\mathcal{T}_m$ of all trees T generated from D with $\#\mathcal{I}(T) \leq m$ inner nodes. Define

$$E_m := \min_{T \in \mathcal{T}_m} E(T)$$

as the best obtainable error by m subdivisions. Of course, this error can be explicitly found by considering all $\mathcal{O}(2^m)$ trees in $\mathcal{T}_m$. This is however an exponential problem and therefore not suitable for real-life problems.

In the following, we will look at several algorithms that (try to) tackle this problem. Each algorithm will operate in the same way: we have a tree T_j with $\#\mathcal{I}(T_j) = n$ inner nodes, and want to find which leaves to subdivide in order to generate T_{j+1}.

1.1 The naive approach

With our error functional e and leaves in place, why not just subdivide the leaves where the error is maximal? It might seem like a valid point at first.

Algorithm 1 (Greedy)**.** *For $j = 0$, $T_0 = D$ is the root node of $\mathcal{T}^*$. If T_j has been defined, examine all leaves $\Delta \in \mathcal{L}(T_j)$ and subdivide the leaves Δ with largest $e(\Delta)$ to produce T_{j+1}. In case of multiple candidates, subdivide all.*

This algorithm is called *greedy*, for it makes the best *local* choice, rather than considering the *global* problem at hand.

Example. Take a look at the function

$$f(x) = \begin{cases} \sin(16\pi x) & x \in [0,1] \\ 0 & x \in [1, 1\frac{1}{2}] \\ 1 & x \in [1\frac{1}{2}, 2] \end{cases}.$$

We will be approximating f with polynomials of degree 0 using the standard error functional $\|f - p_n\|_2^2$. The optimal subdividing strategy immediately subdivides the right half of the domain, effectively nulling the total error there. The greedy algorithm, will (without any effect whatsoever) subdivide the first half into tiny pieces before considering the right half.

dit stuk beter

1.2 h-tree generation: Binev 2004

The above algorithm shows a simple concept: adaptive refinement of a partition. The tree this approach generates is thusly called an h-tree for it refines h, the mesh grid size.

Instead of looking purely at the error functional e in Algorithm 1, a *modified error functional* is introduced in [3]. This modified error functional $\tilde{e}$ penalizes nodes that don't improve after a subdivision. This modification makes for provable performance enhancements, as we will shortly see.

Let $\tilde{e}(D) := e(D)$. Then, given that $\tilde{e}(\Delta)$ is defined, let $\tilde{e}$ for each child Δ_j of Δ be as follows:

$$\tilde{e}(\Delta_j) := q(\Delta), \quad q(\Delta) := \frac{\sum_{j=1}^{2} e(\Delta_j)}{e(\Delta) + \tilde{e}(\Delta)} \tilde{e}(\Delta). \tag{1.1}$$

Algorithm 2 (Binev2004[3, p. 204])**.** *For $j = 0$, $T_0 = D$ is the root node of $\mathcal{T}^*$. If T_j has been defined, examine all leaves $\Delta \in \mathcal{L}(T_j)$ and subdivide the leaves Δ with largest $\tilde{e}(\Delta)$ to produce T_{j+1}. In case of multiple candidates, subdivide all.*

Theorem 9 ([3, Thm. 5.2]). *There is an absolute constant $C > 0$ such that for each j, the output tree T_j of Algorithm 2 satisfies $E(T_j) \leq CE_m$ when $m \leq n/6$. Furthermore, the algorithm uses up to $(n+1)C$ arithmetic operations and computations of e to find this T_j.*

Theorem 10 ([4, Thm. 2]). *The constant C in the above theorem is equal to*

$$C = 1 + \frac{2(m+1)}{n+1-m}.$$

1.2.1 A better modified error functional: Binev 2007

Algorithm 2 can be improved by replacing the definition of the modified error in (1.1) in the following way. Let $\tilde{e}(D) := e(D)$ still, but for each child $\Delta_j \in \mathcal{C}(\Delta)$, let

$$\tilde{e}(\Delta_j) := \left(\frac{1}{e(\Delta_j)} + \frac{1}{\tilde{e}(\Delta)} \right)^{-1}.$$

The accompanying algorithm is now an exact restatement of the earlier result.

Algorithm 3 (Binev2007[4]). *For $j = 0$, $T_0 = D$ is the root node of $\mathcal{T}^*$. If T_j has been defined, examine all leaves $\Delta \in \mathcal{L}(T_j)$ and subdivide the leaves Δ with largest $\tilde{e}(\Delta)$ to produce T_{j+1}. In case of multiple candidates, subdivide all.*

Theorem 11 ([4, Thm. 4]). *Using terminology of Theorem 9: at each step of Algorithm 3, the output tree T satisfies*

$$E(T) \leq \left(1 + \frac{m+1}{n+1-m}\right) E_m,$$

whenever $m \leq n$.

Remark. Comparing Theorems 9 and 10 with the above, we conclude that the new algorithm gives better approximations.

1.2.2 Ensuring linear complexity

Algorithms 2 and 3 have linear complexity in the sense that the amount of operations needed to find T_j is linear in the amount of subdivisions (or equivalently, the amount of inner nodes n). However, sorting a list to find a certain element – in our case, finding the node with largest (modified) error – is inherently a problem of complexity $\mathcal{O}(n \log n)$.

To overcome this problem, it is suggested to use binary bins: for each node, find an integer κ such that $2^\kappa \leq \tilde{e}(\Delta) < 2^{\kappa+1}$ and place Δ in bin κ. The highest nonempty bin now becomes the set of nodes to subdivide. Theorem 9 still holds true in this case [3, p. 207].

1.3 *hp*-tree generation: Binev 2013

Adaptive approximation by piecewise polynomials can be generalized in different ways. One of the most investigated forms of it is the *hp*-approximation in which the local size of elements of the partition and the degree of the polynomials may vary, but the total number of degrees of freedom is controlled.

One motivation for this generalization is that sufficiently smooth functions are best approximated by high-order polynomials, while h-refinement is better suited for non-smooth functions. To profit from both sides, Binev introduced an hp-tree generating algorithm.

literatuur

Given a tree T, we define an hp-tree T^{hp} by decorating each leaf $\Delta \in \mathcal{L}(T)$ with polynomial space $\mathcal{P}^{r(\Delta)}$ of a *node-specific* degree $r(\Delta)$. The number of degrees of freedom in this tree is now

$$\mathcal{N}(T^{hp}) = \sum_{\Delta \in \mathcal{L}(T)} r(\Delta).$$

Of course, we want to approximate some function f. To accomplish this, we will now develop a framework to work with.

1.3.1 Needed definitions

Recall the error functional e from Definition 7. We will extend this to the following.

Definition 9. The function e_k is a function

$$e_k : \mathbb{N} \times \mathcal{T}^* \to \mathbb{R}_{\geq 0}$$

assigning the best obtainable error by approximating f on $\Delta \in \mathcal{T}^*$ with functions in $\mathcal{P}^k$.

If this function further satisfies the subadditivity rules

$$e_k(\Delta) \geq e_{k+1}(\Delta) \qquad \text{and} \qquad e_1(\Delta) \geq e_1(\Delta') + e_1(\Delta''), \quad \{\Delta', \Delta''\} = s(\Delta),$$

then this function is called the *local error of approximation*.

Definition 10. The *total error* of an hp-tree T^{hp} is defined as

$$E(T^{hp}) := \sum_{\Delta \in \mathcal{L}(T^{hp})} e_{r(\Delta)}(\Delta).$$

Definition 11. Given a function f, the best hp-approximation with up to n degrees of freedom is defined as

$$E_n^{hp} := \inf_{\mathcal{N}(T^{hp}) \leq n} E(T^{hp}).$$

Dit heeft nog geen eenduidige definitie

The goal is to find a coarse-to-fine algorithm that analyzes the errors at the current tree and decides how to define the next tree with the degrees of freedom increased by one.

Definition 12.

Let T_N^h with root node D and number of leaves $\#\mathcal{L}(T_N^h) = N$.

Definition 13. For each $\Delta \in T_N^h$, define $T_N^h(\Delta)$ as the maximal subtree of T_N^h with root node Δ.

Definition 14. The order $r(\Delta)$ of a node in T_N^h is defined as number of leaves of $T_N^h(\Delta)$: $r(\Delta) := \#\mathcal{L}(T_N^h(\Delta))$.

Definition 15. Define the modified error functional $\tilde{e}^h$ as follows: $\tilde{e}^h(D) := e_1(D)$, and for each non-root node Δ with children Δ':

$$\tilde{e}^h(\Delta') := \left(\frac{1}{e_1(\Delta')} + \frac{1}{\tilde{e}^h(\Delta)} \right)^{-1}.$$

At each step of the (to be stated) Binev2013 algorithm, we will start with a tree $T = T_N^h$ and gradually 'trim' it to generate a new tree.

Definition 16. Given T and $\Delta \in \mathcal{I}(T)$, we define the *total error of* Δ to be

$$E_T^{hp}(\Delta) := \sum_{\Delta' \in \mathcal{L}(T_N^h(\Delta) \cap T)} e_{r(\Delta')}(\Delta').$$

With these formulas in hand, we can now formulate Binev2013.

1.3.2 The Algorithm

Algorithm 4 (Binev2013[6])**.** *Create an initial tree T_1^h with one node, namely D. Then, in each step N of the algorithm:*

1. *Set $T = T_N^h$.*
2. *For each leaf $\Delta \in \mathcal{L}(T)$, set $\hat{e}(\Delta) = \tilde{e}^h(\Delta)$.*
3. *For each inner node $\Delta \in \mathcal{I}(T)$ – using a reverse level-order traversal – if $e_{r(\Delta)}(\Delta) < E_T^{hp}(\Delta)$:*
   (a) *Make Δ a leaf of T by removing its children.*
   (b) *For each leaf $\Delta' \in \mathcal{L}(T_N^h(\Delta))$, set $\hat{e}(\Delta') = \hat{e}(\Delta') \cdot \frac{e_{r(\Delta)}(\Delta)}{E_T^{hp}(\Delta)}$.*
4. *Create T_N^{hp} from T by giving each leaf $\Delta \in \mathcal{L}(T)$ degree $r(\Delta)$.*
5. *Subdivide the leaves Δ with highest $\hat{e}(\Delta)$, thus creating T_{N+1}^h.*

uitleg moven naar eigen ding? en misschien preciezer zijn?

Explanation. The above algorithm can be motivated by the following.

1. We create a new temporary tree T as to not edit T_N^h (which we still need intact for Step 4).
2. An initial value for each leaf is then set, which we will be multiplying in Step 3b.
3. We traverse $\mathcal{L}(T)$ by means of a reverse level-order traversal (in other words, level-by-level, fine-to-coarse) to comply with the definition of $E_T^{hp}(\Delta)$.

   Of course, we are trying to minimize the total error. The condition ensures that we make the optimal choice concerning Δ: either we assign $r(\Delta)$ to Δ (i.e., we refine p), or do nothing (i.e., refine h).

   In the do-nothing case, the fine-to-coarse traversal ensures that we have already found a good refinement strategyfor Δ and its children. Else,

   (a) we trim the tree, and
   (b) the modified error of descendant leaves of Δ is multiplied by a value in $[0, 1)$ (i.e. lowered) to lessen the likelihood to be subdivided.

beter verwoorden?

Waarom precies deze multiply?

4. We then use the original T_N^h to see which nodes have been merged.
5. This step is of course the induction step.

Theorem 12 ([6]). *Let T_N^{hp} be the hp-tree found by Algorithm 4. Then, for $n \leq N$ we have*

$$E(T_N^{hp}) \leq \left(1 + \frac{2(n-1)}{N+1-n}\right) E_n^{hp}.$$

Assuming that the calculation of $e_k(\Delta)$ requires $\mathcal{O}(1)$ operations (i.e., does not depend on k), the complexity of Algorithm 4 is at most $\mathcal{O}(N^2)$.

Remark. Comparing this result with Theorem 11 makes it seem as if the new algorithm is worse in both terms of complexity and upper bound. However, looks are deceiving, as Algorithm 4 can create *hp*-trees whereas Algorithm 3 creates *h*-trees. The big difference is that we are now able to approximate a much larger class of functions.

Stelling/stuff hierover vinden

Bibliography

[1] Bryan Rynne M.A. Youngson, *Linear Functional Analysis*, Springer, 2008.

[2] Endre Süli David Mayers, *An Introduction to Numerical Analysis*, Cambridge University Press, 2003.

[3] Peter Binev and Ronald DeVore, *Fast computation in adaptive tree approximation*, Springer-Verlag, 2004.

[4] Peter Binev, *Adaptive Methods and Near-Best Tree Approximation*, Oberwolfach Report 29, 2007.

[5] M.D. Choi, *Tricks or Treats with the Hilbert Matrix*, Amer. Math. Monthly 90, 301-312, 1983.

[6] Peter Binev, *Instance optimality for hp-type approximation*, Oberwolfach Report 39, 2013.

[7] R.M. Aarts, C. Bond, P. Mendelsohn and E.W. Weisstein, *Remez Algorithm*, MathWorld, `http://mathworld.wolfram.com/RemezAlgorithm.html`.

[8] George B. Arfken, Hans J. Weber, *Mathematical Methods for Physicists*, Elsevier Academic Press, 2005.