# On the Cost of Uniform and Nonuniform Algorithms

Erich Novak and Henryk Woźniakowski

October 1, 1997, Revised April 17, 1998

**Abstract**

We compare the costs of uniform and nonuniform algorithms for approximate solutions of continuous problems assuming the real number model. We show that, in general, there is no relation between these costs. That is, the class of uniform algorithms may be empty; moreover, even if this class is non-empty then the cost of any uniform algorithm may be arbitrarily larger than the minimal cost of nonuniform algorithms. We also provide conditions under which there exist uniform algorithms whose cost is basically the same as the minimal cost of nonuniform algorithms.

## 1 Introduction

We study the computation of $\varepsilon$-approximations to exact solutions of problems and assume the real number model without an oracle, as in Blum, Shub, Smale (1989), or with an oracle, as in Traub, Wasilkowski, Woźniakowski (1988) or, more formal, in Novak (1995). Such approximations are usually computed by *nonuniform* algorithms (machines, programs). For nonuniform algorithms, the error parameter $\varepsilon$ is regarded as a fixed number. The choice of a nonuniform algorithm depends on $\varepsilon$, and for different $\varepsilon$'s we have, in general, different algorithms. The error parameter $\varepsilon$ is *not* part of the input of the nonuniform algorithm.

The nonuniform algorithm $P_\varepsilon$ has a finite number of built-in (precomputed) parameters. The number of built-in parameters usually depends on $\varepsilon$ and may be large if $\varepsilon$ is small. The minimal cost of nonuniform algorithms that compute an $\varepsilon$-approximation is called the (nonuniform) complexity. For many problems we know tight bounds on the complexity.

Obviously, it is more interesting to consider *uniform* algorithms for which the error parameter $\varepsilon$ is part of the input. Such algorithms can be used for different $\varepsilon$'s and there is no need to change the algorithm when $\varepsilon$ varies. The uniform algorithm $P$ has also a finite number of built-in parameters but they do *not* depend on $\varepsilon$. We still want to compute an $\varepsilon$-approximation, but this time using the uniform algorithm $P$, which takes $\varepsilon$ as part of its input. Obviously, the cost of a uniform algorithm cannot be smaller than the minimal cost of nonuniform algorithms, i.e., the complexity.

It is tempting to define the *uniform* complexity as the minimal cost of uniform algorithms that compute an $\varepsilon$-approximation. There is, however, a problem with such a definition. First of all, the class of uniform algorithms that compute an $\varepsilon$-approximation may be empty. Furthermore, even if this class is non-empty, we still have a problem how to define the minimal cost. For nonuniform algorithms $P_\varepsilon$, the cost $c_{P_\varepsilon}$ is a number and the minimal cost

$$\inf\{c_{P_\varepsilon} : \ P_\varepsilon \text{ computes an } \varepsilon\text{-approximation}\}$$

is well-defined. For uniform algorithms $P$, the cost $c_P$ is a function of $\varepsilon$ which usually goes to infinity as $\varepsilon$ goes to zero. Intuitively, we would like to find a uniform algorithm $P^*$ whose cost function $c_{P^*}$ is minimal among the cost functions of all uniform algorithms, i.e.,

$$c_{P^*} = \inf\{c_P : \ P \text{ with input } \varepsilon \text{ computes an } \varepsilon\text{-approximation}\}.$$

But how to interpret the last infimum? One can try to interpret this pointwise, i.e.,

$$c_{P^*}(\varepsilon) \ = \ \inf\{c_P(\varepsilon) : \ P \text{ with input } \varepsilon \text{ computes an } \varepsilon\text{-approximation}\}.$$

Then $c_{P^*}$ is essentially the (nonuniform) complexity, but the last infimum may not be attained by a uniform algorithm.

This explains why we do not define uniform complexity in this paper. Instead, we compare the costs of uniform algorithms with the (nonuniform) complexity. We study the real number model and show that, in general, anything can happen:

- (i) For some problems, the class of uniform algorithms that compute an $\varepsilon$-approximation is empty even though the class of nonuniform algorithms is non-empty and the complexity is relatively small. We will provide conditions under which uniform algorithms exist.

- (ii) For some problems, the cost of any uniform algorithm is arbitrarily larger than the complexity.

- (iii) For some problems, there exist uniform algorithms with essentially the same cost as the complexity.

We now discuss these three cases in detail.

(i) The class of uniform algorithms is empty for certain problems for which the error is measured by a metric. The choice of the particular metric is crucial. This result holds for a metric that forces us to compute some components of the solution exactly. For this problem, the complexity is finite, and can be a slowly increasing function of $\varepsilon^{-1}$ depending on the specific form of the metric.

The essence of this result also holds for certain problems whose solution does not require oracles. The sequence $p_k = 2^{1/k}$ cannot be computed by a uniform (in $k$) algorithm, but trivially can be computed by nonuniform algorithms with small cost. Section 2 deals with the exact solution of problems without oracles within the BSS-model of computation.

In Section 3 we extend the results of Section 2 to problems in information-based complexity that require the use of oracles. We also show that the class of uniform algorithms is non-empty if the metric is replaced by a norm. The choice of the norm relaxes the demand that some components of the solution must be computed exactly. It turns out that it is now enough to approximate the components of the solution. This implies the existence of uniform algorithms.

Uniform algorithms may exist for problems with metrics. It is enough to assume that there exist nonuniform algorithms that are weakly continuous, see Section 4. Continuity means that a small change of the built-in parameters of the nonuniform algorithm causes a small change in its outputs.

(ii) Even if the class of uniform algorithms is non-empty, it may happen that for some problems the cost of uniform algorithms is arbitrarily larger than the (nonuniform) complexity. By "arbitrarily larger", we mean that the cost of any uniform algorithm that computes an $\varepsilon$-approximation can be of the form $\exp(\mathrm{comp}(\varepsilon))$, or $\exp(\ldots\exp(\mathrm{comp}(\varepsilon))\ldots)$ where $\mathrm{comp}(\varepsilon)$ denotes the complexity, and we can take arbitrarily many compositions of the exponent function. Such problems are presented in Sections 2 and 3. Once more, the choice of the metric is crucial. This bad property disappears if the metric is replaced by a norm.

(iii) We now discuss when uniform algorithms have essentially the same cost as the complexity. We prove that this holds for the approximation problem defined in Section 3, and for approximation of certain linear functionals defined in Section 5. In these two cases, we construct a uniform algorithm from a sequence of appropriately chosen nonuniform algorithms. We code the built-in parameters of the nonuniform algorithms into one single (magic) number which is used as a single built-in parameter of the uniform algorithm. When we run the uniform algorithm for a given $\varepsilon$, we decode from the magic number the built-in parameters of the nonuniform algorithm which are needed for this $\varepsilon$. To control the cost of decoding, we need a continuity assumption on the nonuniform algorithms.

As already mentioned, our construction of uniform algorithms requires one built-in parameter (a magic number) which captures the behavior of nonuniform algorithms for various

$\varepsilon$'s. It is usually very hard, if not impossible, to store such a number. This makes our results on uniform algorithms purely of theoretical interest.

To address the practical side, we note that the difference between uniform and nonuniform algorithms basically disappears if we do not let $\varepsilon$ go to zero, and instead consider $\varepsilon$ in the interval $[\varepsilon_{\text{low}}, \varepsilon_{\text{upp}}]$ with, say, $\varepsilon_{\text{low}} = 10^{-8}$ and $\varepsilon_{\text{upp}} = 10^{-2}$. Then no magic numbers are needed to construct a uniform algorithm which works well for all $\varepsilon \in [\varepsilon_{\text{low}}, \varepsilon_{\text{upp}}]$. Details are provided in Section 6.

In this paper, we concentrate on the error parameter $\varepsilon$ and we distinguish nonuniform and uniform algorithms by the way they treat $\varepsilon$. Obviously, there are other parameters which can play the same role as $\varepsilon$. For instance, suppose we want to find algorithms for multivariate integration for functions from the Sobolev subclass $W_M^{r,p}([0,1]^d)$. Here, $r$ denotes the smoothness of functions defined over the $d$-dimensional unit cube whose $r$th distributional derivatives are bounded in the $L_p$-norm by $M$. We now have four additional parameters: $r, p, d, M$. There are natural restrictions on them such as $r$ and $d$ are positive integers, $p \in [1, \infty]$, $M > 0$, and to assure continuity of functions we need to assume that $p > d/r$ for $d > 1$. Once more, nonuniform algorithms may take all these parameters as fixed. Then the nonuniform algorithm $P_{\varepsilon,r,p,d,M}$ should compute an $\varepsilon$-approximation to the integral $\int_{[0,1]^d} f(t)\, dt$ by sampling the function $f$ (oracle calls) and by performing arithmetic and branching operations. On the other hand, uniform algorithms may take all these parameters $\varepsilon, r, p, d, M$ as inputs and compute an $\varepsilon$-approximation to the integral by using oracle calls and by using arithmetic and branching operations. We believe that a similar analysis of the cost of nonuniform and uniform algorithms could be performed for the case of many parameters.

## 2 Computations without oracle

We consider the following specific problem. For a given sequence of real numbers $p = \{p_k\}$, we want to compute $S_p(n) = p_n$ for any $n = 1, 2, \ldots$. We use the BSS-model with the four standard arithmetic, copy and branching operations over the reals, see Blum, Shub, Smale (1989) for exact definitions.

We give a short and informal description of this model for the case when input and output consist of just one real number, as in our example. Any algorithm (machine) $P$ is of the following form. The algorithm $P$ may use a finite number of built-in parameters $x_{-m}, x_{-m+1}, \ldots, x_{-1}$, where $m$ is finite and may depend on $P$. Here the $x_{-i}$ are real numbers.

We input $x_0$ and for $j = 1, 2, \ldots, J$ the algorithm $P$ performs either an arithmetic operation

$$x_j = \mathrm{op}(x_{j_1}, x_{j_2})$$

with $j_1, j_2 \in [-m, j-1]$, and $\mathrm{op} \in \{+, -, \cdot, /\}$, or a copy operation

$$x_j = x_{j_1},$$

or a goto-operation

$$\text{if } \ x_{j_1} \mathrm{op}_1 \ x_{j_2} \ \text{ then goto X },$$

where $\mathrm{op}_1 \in \{ \leq , \geq , < , > , = \}$, and $j_i \in [-m, j-1]$.

In particular, the number $J$ of arithmetic operations may depend on the input. The output is $P(x_0) = x_J$. Each arithmetic and branching operation (i.e., a comparison together with a jump) has unit cost; copy operations are free. That is, we use the cost function as in Novak (1995), see also Novak and Woźniakowski (1996).

We now consider algorithms for computing $S_p$. We consider two cases and start with the *nonuniform case*. That is, for a given $n$ we are looking for an algorithm $P = P_n$ which may depend on $n$ and which computes $S_p(n)$. Clearly, for any sequence $p$, we can define the algorithm $P_n$ with the built-in parameter $x_{-1} = p_n$ and $P_n(n) = x_1 = x_{-1}$. Then $\mathrm{cost}(P_n, n) = 0$.

Consider now the *uniform case*. That is, we are now looking for a uniform algorithm $P$ whose built-in parameters do not depend on $n$ and for which

$$P(n) = S_p(n) = p_n, \quad \forall\, n.$$

The cost of the algorithm $P$ for $n$ is equal to the total number of arithmetic and branching operations needed to compute $P(n)$ and is denoted by $\mathrm{cost}(P, n)$.

**Theorem 2.1.** For some sequences $\{p_k\}$ there exists *no* uniform algorithm that computes $S_p$. This holds, in particular, for the sequence

$$p_k = 2^{1/k} \in [1,\, 2].$$

**Proof:** For the sequence $p_k = 2^{1/k}$, suppose that there exists a uniform algorithm $P$ that computes $S_p$. Since $P$ can only use a finite number of parameters $x_{-m}, \ldots, x_{-1}$, the input $x_0 = n$, and can perform only arithmetic and branching operations, the elements $S_p(n)$ must belong to the field

$$\mathbf{Q}(x_{-m}, \ldots, x_0).$$

It is well-known that the sequence of the $p_k = 2^{1/k}$ is not contained in a finitely generated *algebraic* field. In our case, however, the parameters $x_{-i}$ are not necessarily algebraic. Consider the field

$$K := \mathbf{Q}(x_{-m}, \ldots, x_0) \cap \overline{\mathbf{Q}},$$

where $\overline{\mathbf{Q}}$ is the field of all algebraic numbers. Since $2^{1/k}$ is algebraic, the sequence $p_k = 2^{1/k}$ belongs to $K$.

It is known that $K$ is finitely generated, i.e., $K$ is of the form

$$K = \mathbf{Q}(y_{-p}, \ldots, y_{-1})$$

with algebraic numbers $y_{-i}$. For this result, see Bourbaki (1959, chap. V, paragraph 5, exercise 7). However, as already mentioned, the sequence $p_k = 2^{1/k}$ is not contained in an algebraic field extension $\mathbf{Q}(y_{-p}, \ldots, y_{-1}) = K$. This is a contradiction. $\square$

Theorem 1 states that for some sequences $p = \{p_k\}$ there are no uniform algorithms. Obviously for some $p$, it is easy to find uniform algorithms that compute $S_p$ and their cost is small or even equal to zero, i.e., it may be the same as the cost of the nonuniform algorithm. Indeed, take $p_k = k$. Then $P(n) = x_0$ computes $S_p$ with cost zero.

We now show that for some sequences $p$, there exist uniform algorithms that compute $S_p$ although their cost can be arbitrarily large. This means that the costs of uniform and nonuniform algorithms can be arbitrarily different.

Let $d$ be a positive integer. Define $h(n,1) = 2^n$ and $h(n,d) = 2^{h(n,d-1)}$. Hence, $h(n,2) = 2^{2^n}$ and so on.

**Theorem 2.2.** For any integer $d$, there exist sequences $p = \{p_k\}$ such that

$$\inf_{P:\ P \text{ computes } S_p} \ \limsup_{n \to +\infty} \frac{\operatorname{cost}(P,n)}{h(n,d)} = 1.$$

**Proof:** Define $p_k = h(k, d+2)$. First of all we find a uniform algorithm that computes $S_p$ with cost $h(n,d)(1+o(1))$. Indeed, let $x_1 = 2$ and $x_{i+1} = x_i^2$, Then $x_i = 2^{2^{i-1}}$. Since, $h(n,d+2) = 2^{2^{h(n,d)}}$, we get $x_J = p_n$ for $J = h(n,d)+1$.

We can compute the number $J = h(n,d)+1$ in cost much smaller than the cost of $h(k,d+2)$ and then we perform the loop

$$\text{for } i := 1 \text{ step } 1 \text{ until } J \text{ do } \ x_i := x_i^2.$$

If the cost of this loop is $J$ then we are done. However, the cost of such a loop is basically $3J$, because we also need branchings and an integer operation like $i := i+1$. We can modify the loop in such a way that several operations of the form $x_i := x_i^2$ are performed before the next branching and the next integer operation. This way we can prove that the infimum in Theorem 2.2 is at most one.

Consider now an arbitrary uniform algorithm $P$ that computes $S_p$. The algorithm $P$ may use the built-in parameters $x_{-j}$, $j = 1, 2, \ldots, m$ and the input $n$. Let

$$a \;=\; \max\{n, |x_{-j}|, 1/|x_j|,\ j = 1, 2, \ldots, m \text{ for } x_{-j} \neq 0\,\}.$$

The algorithm $P$ computes $x_j = \mathrm{op}(x_{j_1}, x_{j_2})$ for $j_1, j_2 < j$ and $j = 1, 2, \ldots$. Since we can at most square the size of a number by one arithmetic operation, we have

$$|x_j| \;\leq\; a^{2^j}.$$

Hence, to compute $p_n$ the algorithm $P$ must compute at least $i$ arithmetic operations with

$$i \;\geq\; \log\log p_n \;-\; \log\log a \;=\; h(n,d) - \log\log n + \;O(1).$$

This yields that

$$\frac{\mathrm{cost}(P,n)}{h(n,d)} \;\geq\; \frac{i}{h(n,d)} \;\geq\; 1 + o(1)$$

and proves Theorem 2.2. $\square$

**Remark 2.1** The sequences $p$ for which Theorem 2.2 hold do not have to be unbounded or have large $p_k$. For instance, define

$$p_n \;=\; \frac{h(n, d+2)}{h(n, d+2) + 1} \;\in [0, 1).$$

Clearly, $h(n, d+2) = p_n/(1 - p_n)$. Thus, the cost of computing $p_n$ can only differ from the cost of computing $h(n, d+2)$ by two arithmetic operations. This shows that Theorem 2.2 also holds for the sequence $\{p_k\}$.

**Remark 2.2** We could also ask which sequences $p_k$ are computable in an approximate sense, i.e., whether there exists an algorithm which, on input $n \in \mathbb{N}$ and a positive $\varepsilon$, computes a number $P(n, \varepsilon)$ such that

$$|p_n - P(n, \varepsilon)| \leq \varepsilon.$$

It turns out that *all* sequences are computable in such an approximate sense. This can be proved via coding as is done in the proof of Theorem 3.1.

# 3 Computations with oracle

We now present a problem which requires us the use of oracles and for which the difference between the costs of uniform and nonuniform algorithms can be arbitrarily large.

Define

$$F = \{f = \{f_i\} : f_i \in \mathbb{R}\}$$

as the space of sequences. Define the metric $\rho$ over $F$ as

$$\rho(f, g) = (n + 1)^{-1},$$

where $n$ is chosen such that the first $n$ components of $f$ and $g$ agree, $f_i = g_i$ for $i = 1, 2, \ldots, n$, and $f_{n+1} \neq g_{n+1}$. (It is easy to check that $\rho$ is indeed a metric.)

Let $S_p : F \to F$ be given by

$$S_p(f) = [p_1 f_1, p_2 f_2, \ldots] \tag{1}$$

for some sequence of $p = \{p_k\}$ with nonzero $p_k$.

We wish to approximate $S_p$ by computing $U_p$ such that

$$\rho(S_p(f), U_p(f)) \leq \varepsilon,$$

where $\varepsilon \in (0, 1)$.

In order to compute $U_p$ we extend the model of computation of the previous section by allowing very simple oracles about $f$. Namely, we assume that $L_i(f) = f_i$ can be computed for any $i$ with unit cost.

Clearly, if we want to compute an $\varepsilon$-approximation we have to compute the first $n$ components of $f$, where

$$n = \left\lceil \varepsilon^{-1} - 1 \right\rceil .$$

The minimal number of oracle calls, which is called the information complexity, is about $\varepsilon^{-1}$.

Consider now the nonuniform case. In this case, the only input to the algorithm $P$ is $f$, and $\varepsilon$ is regarded as a built-in parameter. That is, $P = P_\varepsilon$ and $U_p(f) = P_\varepsilon(f)$ should be an $\varepsilon$-approximation for all $f \in F$.

It is obvious that in the nonuniform case we can find $P_\varepsilon$ that approximates $S_p$ to within $\varepsilon$ with cost proportional to $\varepsilon^{-1}$. Indeed, let $P_\varepsilon$ have built-in parameters $p_1, p_2, \ldots, p_n$ with $n = \lceil \varepsilon^{-1} - 1 \rceil$. Then we compute $f_i$ by oracle calls, multiply and output $p_1 f_1, \ldots, p_n f_n$. The $\mathrm{cost}(P, f)$ is proportional to $\varepsilon^{-1}$. Hence, the (nonuniform) complexity is also proportional to $\varepsilon^{-1}$. Consider now the uniform case. That is, the $\varepsilon$ is now part of the input and $U_p(f) =$

$P(f,\varepsilon)$ should be an $\varepsilon$-approximation for all $f \in F$ and $\varepsilon$. Take $f = 1$, i.e., $f = \{f_i\}$ with $f_i = 1$ for all $i$. Then $P(1,\varepsilon)$ must compute the first $n$ terms of the sequence $p$. It directly follows from the previous section that for some $p$ there exists no such uniform algorithm $P$, and that for some other sequences $p$, the cost of computing $P(1,\varepsilon)$ can be arbitrarily large. In particular, it can be exponentially larger than the minimal cost of the nonuniform case which is roughly $\varepsilon^{-1}$.

**Remark 3.1**

Observe that the specific metric in our example can be modified in various ways. That is, if we define the metric $\rho$ as $\rho(f,g) = h(n+1)$ for some decreasing function $h$ with the same definition of $n$ as before, then the complexity is roughly $1/h^{-1}(\varepsilon)$. Hence, if $h$ quickly goes to zero, the complexity slowly goes to infinity as $\varepsilon$ goes to zero. For instance, for $h(x) = \exp(-x)$ the complexity is roughly $\ln 1/\varepsilon$. Still, no matter how we choose the function $h$, the class of uniform algorithms is empty for some sequences $p$. This shows that even if the problem is trivial in the nonuniform case, the class of uniform algorithms may be empty.

The essence of this example is that we used a metric which forces us to compute some components of $S_p(f)$ exactly. We now show that the situation is quite different if we replace the metric by a norm.

More precisely, consider the *approximation* problem defined as follows. Let $F = \{f \in l_2 \mid \|f\| \leq 1\}$ be the unit ball of $l_2$, and, let $S_p : l_2 \to l_2$ be given by

$$S_p(f) = [f_1 p_1, f_2 p_2, \ldots]$$

for a bounded positive sequence $p = \{p_k\}$, $p_k \leq M$ for some positive $M$. Here $\|f\|^2 = \sum_{i=1}^{\infty} f_i^2$. Without loss of generality, we assume that $M \geq 1$. To simplify further considerations, we also assume that we can compute exp, ln and ceilings with unit cost.

We want to compute $U_p$ which is an $\varepsilon$-approximation to $S_p$, i.e.,

$$\|S_p(f) - U_p(f)\| \leq \varepsilon, \quad \forall\, f \in F.$$

Here, $U_p$ uses a finite number of oracle calls about $f$. It is well-known that there exists $U_p$ such that $U_p(f)$ is an $\varepsilon$-approximation for an arbitrary positive $\varepsilon$ iff the sequence of the $p_k$ converges to zero. Hence, we assume

$$p_1 \geq p_2 \geq \ldots \geq p_n \to 0.$$

The minimal number of oracles to compute an $\varepsilon$-approximation, the information complexity, is equal to the minimal $n = n(\varepsilon)$ such that $p_{n+1} \leq \varepsilon$. We consider two cases:

a) The nonuniform case. We now assume that $\varepsilon > 0$ is fixed, and only $f$ from $F$ is input. We define the algorithm $P_\varepsilon$ with the built-in parameters $n(\varepsilon), p_1, p_2, \ldots, p_{n(\varepsilon)}$ such that $P_\varepsilon(f) = [f_1 p_1, f_2 p_2, \ldots, f_{n(\varepsilon)} p_{n(\varepsilon)}]$. Of course, we identify the output $P_\varepsilon(f)$ with the element

$$U_p(f) = [f_1 p_1, \ldots, f_{n(\varepsilon)} p_{n(\varepsilon)}, 0, 0, \ldots] \in l_2.$$

The cost of computing $P_\varepsilon(f)$ is proportional to $n(\varepsilon)$.

b) The uniform case. We now have an input $f$ from $F$ and a positive $\varepsilon$. If we have in mind the first example above then it is not clear under which conditions the uniform problem is solvable.

To get solvability we assume that $n(\varepsilon)$, or an upper bound of the same order, can be computed for an arbitrary $\varepsilon$ with cost at most of the order $n(\varepsilon)$. For instance, if $p_k = k^{-m}$ then $n(\varepsilon) = \lceil \varepsilon^{-1/m} \rceil = \lceil \exp(-\ln(\varepsilon)/m) \rceil$ can be computed with constant cost.

**Theorem 3.1.** The approximation problem is solvable in the uniform case iff the approximation problem is solvable in the nonuniform case, i.e., iff $\lim_{k\to\infty} p_k = 0$.

If $\lim_{k\to\infty} p_k = 0$ then there exists a uniform algorithm $P$ such that $P(f,\varepsilon)$ is an $\varepsilon$-approximation of $S_p(f)$ and

$$\mathrm{cost}(P(f,\varepsilon)) \;=\; O\left( n(\varepsilon/\sqrt{2})\,(1 + \log_2 M + \log_2 1/\varepsilon) \right)$$

with the constant in the big $O$ notation independent of $f, p, \varepsilon$ and $M$.

**Proof:** It is enough to prove that solvability in the nonuniform case implies solvability in the uniform case. The proof will be via coding. Since $p_n \le M$ then

$$p_n \;=\; \sum_{i=0}^{\lceil \log_2 M \rceil} p_{n,i} 2^i \;+\; \sum_{i=1}^{\infty} p_{n,-i} 2^{-i}, \quad \text{with } p_{n,i}, p_{n,-i} \in \{0,1\}.$$

Define the rational numbers $q_{n,k}$ by

$$q_{n,k} \;=\; \sum_{i=0}^{\lceil \log_2 M \rceil} p_{n,i} 2^i \;+\; \sum_{i=1}^{k} p_{n,-i} 2^{-i}, \qquad \forall\, n,k = 1,2,\ldots .$$

Clearly, $|p_n - q_{n,k}| \le 2^{-k}$ and $q_{n,k}$ can be represented by $b_{n,k} = \lceil \log_2 M \rceil + 1 + k$ bits.

We now code the double sequence $\{q_{n,k}\}$ into one real number $x_{-1}$ by diagonalization as follows. We order the sequence $\{q_{n,k}\}$ by $q_{1,1}, q_{1,2}, q_{2,1}, q_{1,3}, q_{2,2}, q_{3,1}, \ldots$. That is the number $q_{n,k}$ occupies the position $(n+k-2)(n+k-1)/2 + n$. Let

$$A_{n,k} \;=\; \{\, (i,j) \;:\; 1 \le i,j,\; i+j < n+k \,\text{or}\, i+j = n+k \,\text{and}\, i \le n \,\}$$

be the set of indices of the first $(n+k-2)(n+k-1)/2+n$ elements of $\{q_{n,k}\}$. Let

$$\mathrm{bits}_{n,k} \;=\; \sum_{(i,j)\in A_{n,k}} b_{i,j}$$

be the total number of bits of the first $(n+k-2)(n+k-1)/2+n$ elements of $\{q_{n,k}\}$. We compute $\mathrm{bits}_{n,k}$ as follows. We have $(n+k-2)$ indices of $q_{i,j}$ with $j=1$ and $(n+k-3)$ indices with $j=2$. Continue in this way, until we reach $j=k$ for which there are $n-1$ indices. We now consider the indices $j$ with $j>k$. We have $n-1$ indices with $j=k+1$ and so on till we have just one index with $j=n+k-1$. Thus we obtain

$$\mathrm{bits}_{n,k} = [(n+k-2)(n+k-1)/2+n]\cdot(\lceil\log_2 M\rceil+1)+\sum_{i=1}^{k} i(n+k-1-i)+\sum_{i=1}^{n-1}(n-i)(k+i).$$

Using the closed form formulas for the sums of $\sum_{i=1}^{k} i^j$ for various $k$ and $j=1,2$ we obtain

$$\begin{aligned}\mathrm{bits}_{n,k} \;&=\; ((n+k-2)(n+k-1)/2+n)\,\lceil\log_2 M\rceil+1\\ &\quad +(-4n-13k+6nk+n^3+3n^2+3nk^2+3n^2k+k^3)/6.\end{aligned}$$

Hence, $\mathrm{bits}_{n,k}$ can be computed with constant time.

We define the number

$$x_{-1}=\sum_{m=1}^{\infty} b_m 2^{-m}$$

whose successive bits $[b_{m+1},b_{m+2},\dots,b_{m+b_{n,k}}]$ are equal to the bits of $q_{n,k}$ with $m=\mathrm{bits}_{n,k}-b_{n,k}+1$.

Observe that for large $n$ and $k$, the number $q_{n,k}$ approximates, $p_n$ which goes to zero. Hence, the initial bits of $q_{n,k}$ are zero. This means that some $b_m$'s for large $m$ are also zero. On the other hand, for a fixed $n$ and a large $k$, the number $q_{n,k}$ approximates a nonzero $p_n$, and therefore some bits of $q_{n,k}$ are one. This means that some $b_m$'s for large $m$ are also one. Hence, for all $m_0$ the sequence $\{b_m\}$ for $m\ge m_0$ is *not* constant. This proves that for all $i$, the number $2^i x_{-1}$ is *not* an integer, and

$$\lceil 2^i x_{-1}\rceil \;=\; \sum_{m=1}^{i} b_m 2^{i-m} \;+\; 1. \tag{2}$$

We are ready to define the algorithm $P$ such that $P(f,\varepsilon)$ computes an $\varepsilon$-approximation of $S_p(f)$. The algorithm uses the number $x_{-1}$ as a built-in parameter (a magic number) and consists of a couple of steps.

First, for a given $\varepsilon$, we compute $k = 1/2 - \ln \varepsilon / \ln 2$. Hence, $2^{-k} \le \varepsilon/\sqrt{2}$. Then we compute the index $n = n(\varepsilon/\sqrt{2})$ such that $p_{n+1} \le \varepsilon/\sqrt{2}$. The cost of computing $k$ and $n$ is at most proportional to $n(\varepsilon/\sqrt{2})$. Here, $\sqrt{2}$ may be the other built-in parameter.

We now recover $q_{1,k}, q_{2,k}, \dots, q_{n,k}$ from the number $x_{-1}$ by decoding. More precisely, $q_{j,k}$ can be obtained by recovering the $m+1, m+2, \dots, m+b_{n,k}$ bits of $x_{-1}$ with $m = \text{bits}_{j,k} - b_{j,k}$. Due to (2), the $i$th bit $b_i$ of $x_{-1}$ is equal to

$$b_i \;=\; \lceil 2^i x_{-1} \rceil \;-\; 2\lceil 2^{i-1} x_{-1} \rceil \;+\; 1$$

which can be computed with constant cost. Hence, the cost of computing all $q_{j,k}$ for $j = 1, 2, \dots, n$ is proportional to the total number of bits of all the $q_{j,k}$, which is of order $n(1 + \log_2 M + \log_2 1/\varepsilon)$.

Finally, we define

$$P(f, \varepsilon) \;=\; \sum_{j=1}^{n} f_j \, q_{j,k}$$

which can be computed by $n$ oracle calls and $n$ multiplications and additions. Observe that

$$\begin{aligned} \|S_p(f) - P(f,\varepsilon)\|^2 \;&=\; \sum_{j=1}^{n} (p_j - q_{j,k})^2 f_j^2 \;+\; \sum_{j=n+1}^{\infty} p_j^2 f_j^2 \\ &\le\; 2^{-2k} \sum_{j=1}^{n} f_j^2 + \frac{\varepsilon^2}{2} \sum_{j=n+1}^{\infty} f_j^2 \;\le\; \varepsilon^2 \|f\|^2 \;\le\; \varepsilon^2. \end{aligned}$$

This means that $P(f, \varepsilon)$ is an $\varepsilon$-approximation for arbitrary $f \in F$ and positive $\varepsilon$. The total cost of computing $P(f, \varepsilon)$ is of order $n(1 + \log_2 M + \log_2 1/\varepsilon)$. Hence, the proof is complete. $\square$

# 4 Continuity

In the first part of Section 3 we presented a problem for which there exists no uniform algorithm for computing an $\varepsilon$-approximation. We now show that uniform algorithms exist if we assume the existence of *continuous* or *weakly continuous* nonuniform algorithms. To define the concept of continuity we proceed as follows.

Let $S : F \to G$ be a operator, possibly nonlinear. Here $F$ is a given set and $G$ is assumed to be a metric space with the metric $\rho$. We assume that $G$ is a subset of

$$\mathbb{R}^* = \bigcup_{i=1}^{\infty} \mathbb{R}^i$$

consisting of finite sequences of real numbers. We want to approximate $S(f)$ for all $f$ from $F$ by means of oracles $L(f)$ for some functionals $L$, i.e., $L : F \to \mathbb{R}$. Assume that for all positive $\varepsilon$, there exists a nonuniform algorithm $P_\varepsilon$ which computes an $\varepsilon$-approximation. That is,

$$\sup_{f \in F} \rho\,(S(f), P_\varepsilon(f)) \;\le\; \varepsilon.$$

We assume that the built-in parameters of the nonuniform algorithm $P_\varepsilon$

$$g_{1,\varepsilon}, \ldots, g_{k(\varepsilon),\varepsilon}$$

are real numbers.

By continuity, we mean that a small change of the numbers $g_{i,\varepsilon}$ does not cause much change in the computed result $P_\varepsilon(f)$. More precisely, for $\delta > 0$ we define a nonuniform algorithm $P_{\varepsilon,\delta}$ in the following way. The algorithms $P_\varepsilon$ and $P_{\varepsilon,\delta}$ are identical except that the built-in real numbers $g_{i,\varepsilon}$ of $P_\varepsilon$ are replaced by built-in *rational* numbers $g_{i,\varepsilon}^\delta$ for $P_{\varepsilon,\delta}$ such that

$$|g_{i,\varepsilon} - g_{i,\varepsilon}^\delta| \le \delta, \qquad \forall\, i = 1, 2, \ldots, k(\varepsilon),\ \forall\, \varepsilon > 0. \tag{3}$$

We say that the nonuniform algorithm $P_\varepsilon$ is *continuous* if for any positive $\varepsilon$ there exists a positive $\delta$ such that for *all* rationals $g_{i,\varepsilon}^\delta$ satisfying (3) we have

$$\sup_{f \in F} \rho\,(S(f), P_{\varepsilon,\delta}(f)) \le 2\varepsilon. \tag{4}$$

We say that the nonuniform algorithm $P_\varepsilon$ is *weakly continuous* if for any positive $\varepsilon$ there exists a positive $\delta$ such that for *some* rationals $g_{i,\varepsilon}^\delta$ satisfying (3) we have (4).

Observe that for the problem $S_p$ given by (1) with an irrational $p_1$, there exists *no* continuous or weakly continuous nonuniform algorithm. In fact, the error of any nonuniform algorithm that uses rational built-in numbers is one. This shows that the assumption on continuity is essential for the existence of uniform algorithms.

We now illustrate continuous and weakly continuous algorithms by an example.

**Example 4.1** We discuss continuity for integration and quadrature formulas. We show that continuity and weak continuity depend on how the quadrature formulas are implemented. Consider the class

$$F = \{f \in C^1([0,1]) \;\mid\; \|f'\|_\infty \le 1\}$$

with $S(f) = \int_0^1 f(t)\,dt$, $\rho(a,b) = |a-b|$, and the quadrature formulas

$$P_\varepsilon(f) \;=\; Q_n(f) \;=\; \frac{1}{n} \sum_{i=1}^{n} f\left(\frac{2i-1}{2n}\right).$$

Here $n = \Theta(\varepsilon^{-1})$ and the error of $P_\varepsilon$ is at most $\varepsilon$.

Observe that any linear algorithm with finite worst case error must exactly integrate the constant functions $f_c(\cdot) = c$. In particular, this means that $P_\varepsilon(f_c) = S(f_c) = c$, $\forall\, c \in \mathbb{R}$. This condition, obviously, holds for $Q_n$.

Assume for simplicity that $n = 2$. An obvious way to compute $Q_2(f)$ is by the formula

$$Q_2(f) = g_1\, f(g_2) + g_3\, f(g_4)$$

with the four parameters $g_1 = g_3 = \frac{1}{2}$ and $g_2 = \frac{1}{4}$ and $g_4 = \frac{3}{4}$. If programmed this way then $P_\varepsilon$ is *not* continuous. Indeed, if we use $g_i^\delta$ such that $g_1^\delta + g_3^\delta \neq 1$ then (4) does not hold since the left-hand of (4) is infinite.

Let us now return to arbitrary $n$. We have to guarantee that the sum of the weights is 1. Clearly, for *some* rational approximations $g_i^\delta$ we have $g_1^\delta + g_2^\delta + \ldots + g_n^\delta = 1$. In fact we can now even take $g_i^\delta = 1/n$. So $P_\varepsilon$ is *weakly* continuous.

To obtain continuity we can eliminate one of the parameters $g_i$. For instance, we take $g_{2i-1} = 1/n$ for $i = 2, 3, \ldots, n$ and $g_{2i} = (2i-1)/(2n)$ for $i = 1, 2, \ldots, n$. Then we compute $g_1 = 1 - g_3 - g_5 - \ldots - g_{2n-1}$. A small error in the parameters $g_2, g_3, \ldots, g_{2n}$ now leads to a small error in the result, which yields continuity of $P_\varepsilon$.

To illustrate further this point, let us consider integration for the class

$$F \;=\; F_k \;=\; \{ f \in C^k[0,1] \;\mid\; \|f^{(k)}\|_\infty \le 1 \}.$$

Then

$$Q_n(f) \;=\; \sum_{i=1}^{n} g_i f(t_i)$$

has finite worst case error iff $Q_n$ is exact for all polynomials of degree less than $k$. Furthermore, for $n = \Theta(\varepsilon^{-1/k})$ there exist $g_i = g_{i,\varepsilon}$ and $t_i = t_{i,\varepsilon}$ for which $P_\varepsilon = Q_n$ has error at most $\varepsilon$.

Hence, for distinct sample points $t_i$, the parameters $g_i$ must satisfy the linear equations

$$Q_n(t^j) \;=\; \sum_{i=1}^{n} g_i t_i^j \;=\; S(t^j) \;=\; \frac{1}{j+1}, \qquad j = 0, 1, \ldots, k-1.$$

To obtain continuity of $P_\varepsilon$, we have to guarantee that the perturbed knots $t_{i,\varepsilon}^\delta$ and weights $g_{i,\varepsilon}^\delta$ satisfy the same system of linear equations, i.e.,

$$\sum_{i=1}^{n} g_{i,\varepsilon}^\delta \,(t_{i,\varepsilon}^\delta)^j \;=\; \frac{1}{j+1}, \qquad j = 0, 1, \ldots, k-1. \tag{5}$$

Hence, if all $g_{i,\varepsilon}$ and $t_{i,\varepsilon}$ are regarded as built-in parameters of $P_\varepsilon$ then $P_\varepsilon$ is *not* continuous since not all small rational perturbations $g_{i,\varepsilon}^\delta$ and $t_{i,\varepsilon}^\delta$ satisfy (5). On the other hand, $P_\varepsilon$ *is* weakly continuous since some small rational perturbations $g_{i,\varepsilon}^\delta$ and $t_{i,\varepsilon}^\delta$ satisfy (5). As before, we can obtain continuity of $P_\varepsilon$ by eliminating, say, the first $k$ parameters $g_1, g_2, \ldots, g_k$ and compute them as the solution of the linear equations

$$\sum_{i=1}^{k} g_i (t_{i,\varepsilon}^\delta)^j = \frac{1}{j+1} - \sum_{i=k+1}^{n} g_{i,\varepsilon}^\delta\,(t_{i,\varepsilon}^\delta)^j, \qquad j = 0, 1, \ldots, k-1.$$

We stress that such elimination of built-in parameters is *not* always needed. For instance, if $F_k$ is replaced by the smaller class

$$\widetilde{F}_k = \{ f \in C^k[0,1] \mid \|f^{(r)}\|_\infty \le 1 \text{ for } r = 0, \ldots, k \}$$

then we can implement quadrature formulas in the obvious way and still get continuous algorithms. $\square$

We are ready to prove that weak continuity implies the existence of uniform algorithms. This will be shown by coding as in the proof of Theorem 3.1.

**Theorem 4.1** Assume that for each positive $\varepsilon$ we can approximate $S$ to within $\varepsilon$ by weakly continuous nonuniform algorithms $P_\varepsilon$. Then there exists a uniform algorithm which computes an $\varepsilon$-approximation of $S$.

**Proof:** To obtain a uniform algorithm, we consider the sequence of weakly continuous nonuniform algorithms $\widetilde{P}_{2^{-k}}$ with error $2 \cdot 2^{-k}$. Since each $\widetilde{P}_{2^{-k}}$ needs only finitely many rational built-in numbers, we can code these numbers as well as the whole program $\widetilde{P}_{2^{-k}}$ into a single (natural or rational) number $G_k$. As in the classical theory of computable functions, this number $G_k$ can be called a *Gödel number*. All these Gödel numbers can be coded into one (magic) real number $G$ and a *universal BSS-machine* can run (simulate) a computation $\widetilde{P}_{2^{-k}}(f)$ on an input $G$, $f$ and $\varepsilon$ with $k = \lceil \log_2 2/\varepsilon \rceil$. See Blum, Shub, Smale (1989) for the construction of a universal machine over the reals. $\square$

# 5 Linear Functionals

Theorem 4.1 does not address the cost of uniform algorithms. In this section we analyze the cost of uniform algorithms in terms of the minimal cost of nonuniform algorithms. This will be done for linear functionals. That is, $S : F_1 \to \mathbb{R}$ is a linear functional and $F_1$ is a linear normed space of real functions. We want to approximate $S(f)$ for all $f$ from a given subset $F$ of $F_1$. We assume that $F$ is convex and balanced. By oracles, we now mean function values.

Let the (nonuniform) complexity be

$$\mathrm{comp}(\varepsilon) \;=\; O\left(\varepsilon^{-1/q} \cdot (-\log \varepsilon)^{\alpha}\right), \quad 0 < \varepsilon < 1/2 \tag{6}$$

for some positive $q$ and nonnegative $\alpha$.

It is known that the complexity in the nonuniform case is obtained by *linear* algorithms using nonadaptive function values, see e.g., TWW(1988). That is, there exists an integer $n = n(\varepsilon) = O(\varepsilon^{-1/q}(-\log \varepsilon)^{\alpha})$ and a linear $P_\varepsilon$ such that

$$P_\varepsilon(f) = \sum_{i=1}^{n} g_{2i-1,\varepsilon}\, f(g_{2i,\varepsilon}) \tag{7}$$

has worst case error $\sup_{f \in F} |S(f) - P_\varepsilon(f)| \leq \varepsilon$.

We assume that the built-in parameters $g_{i,\varepsilon}$ of $P_\varepsilon$ are uniformly bounded,

$$|g_{i,\varepsilon}| \leq M, \qquad \forall\, i = 1, 2, \ldots, 2n(\varepsilon),\ \forall \varepsilon > 0, \tag{8}$$

for some $M \geq 2$.

We assume that $P_\varepsilon$ is weakly continuous. That is, for a positive $\varepsilon$ there exists a positive $\delta$ such that the linear algorithm

$$P_{\varepsilon,\delta}(f) = \sum_{i=1}^{n} g^{\delta}_{2i-1,\varepsilon}\, f(g^{\delta}_{2i,\varepsilon})$$

with some rational $g^{\delta}_{i,\varepsilon}$ satisfying $|g_{i,\varepsilon} - g^{\delta}_{i,\varepsilon}| \leq \delta$ has error at most $2\varepsilon$,

$$\sup_{f \in F} |S(f) - P_{\varepsilon,\delta}(f)| \leq 2\varepsilon. \tag{9}$$

Without loss of generality we may assume that the perturbed parameters $g^{\delta}_{i,\varepsilon}$ are bounded by $M$, i.e., $|g^{\delta}_{i,\varepsilon}| \leq M$. We additionally assume that

$$O(\log M + \log n) \tag{10}$$

bits are sufficient to represent each $g^{\delta}_{i,\varepsilon}$ exactly.

We add in passing that for the classes $F_k$ discussed in Example 4.1 is is relatively easy to check that (10) holds.

**Theorem 5.1** Under the above assumptions (6-10) there exists a uniform algorithm $P$ which computes an $\varepsilon$-approximation to $S$ with cost

$$\mathrm{cost}(\varepsilon) \;=\; O\left(\varepsilon^{-1/q}\cdot(-\log\varepsilon)^{\alpha+1}\right).$$

That is, the cost of $P$ is at most $\log 1/\varepsilon$ larger than the (nonuniform) complexity.

**Proof:** The idea of the proof is the same as in the proof of Theorem 3.1. Define a positive $c_2$ such that

$$n \;=\; n(\varepsilon) \;=\; \lceil c_2\varepsilon^{-1/q}(-\log\varepsilon)^{\alpha}\rceil$$

implies that the worst case error of $P_\varepsilon$ given by (7) is at most $\varepsilon$. For $\varepsilon_k = 2^{-(k+1)}$, compute

$$n_k \;=\; \lceil c_2 2^{(k+1)/q}(k+1)^{\alpha}\rceil.$$

The nonuniform algorithm $P_{\varepsilon_k}$ uses the built-in parameters $g_{i,\varepsilon_k}$ for $i = 1, 2, \ldots, 2n_k$. For $\delta = \delta_k = c_1\varepsilon_k^p$, take the rational numbers $g_{i,\varepsilon_k}^{\delta_k}$ satisfying (9) and (10).

We code the double sequence $\{g_{i,\varepsilon_k}^{\delta_k}\}$, $i, k = 1, 2, \ldots$, into the number $x_{-1}$ as in the proof of Theorem 3.1.

We are ready to define the uniform algorithm $P$ that uses the number $x_{-1}$ as a built-in parameter. For a given input $\varepsilon$, we compute the smallest $k$ for which $2\varepsilon_k \le \varepsilon$. Then we decode $g_{i,\varepsilon_k}^{\delta_k}$ for $i = 1, 2, \ldots, 2n_k$ from the number $x_{-1}$ with cost proportional to

$$2n_k\,(\log M \;-\; \log\varepsilon) \;=\; O\left(\varepsilon^{-1/q}(-\log\varepsilon)^{\alpha+1}\right).$$

Then we define

$$P(f,\varepsilon) \;=\; P_{\varepsilon_k,\delta_k}(f) \;=\; \sum_{i=1}^{n_k} g_{2i-1,\varepsilon_k}^{\delta_k} f(g_{2i,\varepsilon_k}^{\delta_k})$$

which can be computed in cost proportional to $n_k = O(\varepsilon^{-1/q}(-\log\varepsilon)^{\alpha})$. The error of $P_{\varepsilon_k,\delta_k}$ is at most $2\varepsilon_k \le \varepsilon$. Hence, $P(f,\varepsilon)$ is an $\varepsilon$-approximation of $S(f)$ for all $f \in F$. This completes the proof. $\square$

# 6 Practical Considerations

We do not claim that the positive results of Sections 3–5 given by Theorems 3.1, 4.1 and 5.1 are always practical. All these results are based on some kind of coding and to use such uniform algorithms one has, in general, to find out and store at least one *magic* real number.

This magic number is needed *with arbitrary precision*, i.e., exactly. Such an assumption is not realistic in practical computation where usually floating point arithmetic is used, and all numbers are rounded and only their approximations are used. This suggests that one should also study algorithms where only integer or rational numbers are allowed as built-in parameters, see Hemmerling (1998).

In this section we want to discuss a more practical question. In many cases, the error parameter $\varepsilon$ does not really go to zero. Instead, we want to compute a solution *quickly* with modest accuracy $\varepsilon$. This modest accuracy means that $\varepsilon$ is restricted to the interval $[\varepsilon_{\text{low}}, \varepsilon_{\text{upp}}]$. A typical interval could be something like $[\varepsilon_{\text{low}}, \varepsilon_{\text{upp}}] = [10^{-8}, 10^{-2}]$.

How can we efficiently solve such a problem with restricted $\varepsilon$? We assume that we know efficient nonuniform algorithms $P_{\varepsilon_j}$ for some $\varepsilon_j$ from the interval $[\varepsilon_{\text{low}}, \varepsilon_{\text{upp}}]$ such that $P_{\varepsilon_j}$ computes an $\varepsilon_j$-approximation. The cost of computing $P_{\varepsilon_j}(f)$ for any $f$ from $F$ is denoted by $\text{cost}(\varepsilon_j)$ for some nonincreasing function cost. Efficiency of $P_{\varepsilon_j}$ means that $\text{cost}(\varepsilon_j)$ is comparable to the complexity.

An obvious approach is to combine a fixed number of nonuniform algorithms $P_{\varepsilon_j}$ into one uniform algorithm $P$. More precisely we proceed as follows. For some integer $n$, we define

$$c = \left(\frac{\varepsilon_{\text{upp}}}{\varepsilon_{\text{low}}}\right)^{1/n} \tag{11}$$

and

$$\varepsilon_j = c^{j-1} \cdot \varepsilon_{\text{low}}, \qquad j = 1, 2, \ldots, n+1.$$

Hence, $\varepsilon_1 = \varepsilon_{\text{low}}$ and $\varepsilon_{n+1} = \varepsilon_{\text{upp}}$.

If we want to solve the problem for any $\varepsilon \in [\varepsilon_{\text{low}}, \varepsilon_{\text{upp}}]$ then we replace $\varepsilon$ by $\varepsilon_j \le \varepsilon$ with the largest $j \in \{1, \ldots, n\}$ and compute

$$P(f, \varepsilon) := P_{\varepsilon_j}(f).$$

Observe that the error of $P$ is at most $\varepsilon_j \le \varepsilon$, and the cost of computing $P(f, \varepsilon)$ is

$$\text{cost}(\varepsilon_j) + \tilde{c} \;\le\; \text{cost}(\varepsilon/c) + \tilde{c},$$

where $\tilde{c}$ is a small constant needed to branch to the right $P_{\varepsilon_j}$.

Hence, if $c$ is not too large and the function cost varies relatively slowly, the cost of the uniform algorithm $P$ is comparable to the complexity.

Furthermore, the built-in parameters of $P$ consist of the built-in parameters of $P_{\varepsilon_j}$ for $j = 1, 2, \ldots, n$. The total number of the built-in parameters of $P$ is in many cases comparable to the maximal number of the built-in parameters used by $P_{\varepsilon_j}$. Indeed, this is the case when

$n$ is not too large. We stress that for the uniform algorithm $P$ there is no need to store magic numbers since we do not need to code or decode anything when we run $P(f, \varepsilon)$.

To illustrate this discussion, we take some specific numbers. Let $\varepsilon_{\text{low}} = 10^{-8}$, $\varepsilon_{\text{upp}} = 10^{-2}$ and $n = 12$. Then $c = \sqrt{10}$. Finally, let $\text{cost}(\varepsilon) = \text{comp}(\varepsilon) = \varepsilon^{-1/2}$. Then the cost of the uniform algorithm is at most $10^{1/4}$ times more than the complexity, and it uses at most 12 times as many built-in parameters as the nonuniform algorithms.

**Remark 6.1** Optimal algorithms $P_\varepsilon$ do not, in general, have much in common for different $\varepsilon$. In many cases, however, one can define a family $\{P_\varepsilon\}_\varepsilon$ of almost optimal algorithms by a single rule. Often there exists a basic version of an algorithm which uses $k$ information functionals that can be scaled down to smaller intervals (or finite elements). The scaled version uses $n = k \cdot m$ functionals, $m \in \mathbb{N}$, and the whole family of these algorithms can be easily generated by a single uniform algorithm and is almost optimal for each $\varepsilon$. In such a case, we do not need coding and magic numbers, and still get good cost bounds for all $\varepsilon$, not only for $\varepsilon \in [\varepsilon_{\text{low}}, \varepsilon_{\text{upp}}]$. We give an example.

Consider the integration problem over the class $F_k$ as defined in Example 4.1. It is well-known that the (nonuniform) complexity is of the order

$$\text{comp}(\varepsilon) \asymp \varepsilon^{-1/k}.$$

As a uniform algorithm $P$ we can take straightforward composite quadrature formulas. The cost of the uniform algorithm $P$ with input $f \in F_k$ and $\varepsilon \in (0,1)$ is again

$$\text{cost}(P, f, \varepsilon) \asymp \varepsilon^{-1/k},$$

so it is easy to obtain the same order by a uniform algorithm.

By means of composite quadrature formulas, together with Clenshaw-Curtis formulas, we can even allow $k$ as an input and obtain a uniform algorithm $P$ with

$$\text{cost}(P, f, k, \varepsilon) \le c_1 \cdot \varepsilon^{-1/k} + c_2\, k \log k.$$

The constants $c_i$ do not depend on $\varepsilon$, $k$ and $f$ and again no magic number is needed, see Novak (1995).

# Acknowledgment

Our special thanks go to Steve Smale. We started this project during the Park City conference, Utah, 1995, organized by Steve Smale and continued when the second author visited

Steve Smale at the City University of Hong Kong. The discussions with Steve Smale were very helpful and resulted in a number of issues which are addressed in this paper.

We also thank K. Ritter, J. F. Traub, G. W. Wasilkowski and A. G. Werschulz for valuable comments on this paper and J. Köhn for his help with reference [2].

The first author (E.N.) was supported by a Heisenberg scholarship of the DFG and the second author (H.W.) was partially supported by the National Science Foundation.

## Literature

[1] L. Blum, M. Shub, S. Smale (1989): On a theory of computation and complexity over the real numbers: NP completeness, recursive functions and universal machines. Bull. Amer. Math. Soc. **21**, 1–46.

[2] N. Bourbaki (1959): Algèbre, Chap. V. Hermann, Paris.

[3] A. Hemmerling (1998): Computability of string functions over algebraic structures. Math. Logic Quarterly **44**, 1–44.

[4] E. Novak (1995): The real number model in numerical analysis. J. Complexity **11**, 57–73.

[5] E. Novak, H. Woźniakowski (1996): Topological complexity of zero finding. J. Complexity **12**, 380–400.

[6] J. F. Traub, G. W. Wasilkowski, H. Woźniakowski (1988): Information-Based Complexity, Academic Press, New York.

**Author's Addresses:**

Erich Novak, Mathematisches Institut, Universität Erlangen-Nürnberg,
Bismarckstr. 1 1/2, 91054 Erlangen, Germany, email: novak@mi.uni-erlangen.de

Henryk Woźniakowski, Department of Computer Science, Columbia University,
New York, NY 10027, USA, and Institute of Applied Mathematics, University of Warsaw,
ul. Banacha 2, 02-097 Warszawa, Poland, email: henryk@cs.columbia.edu