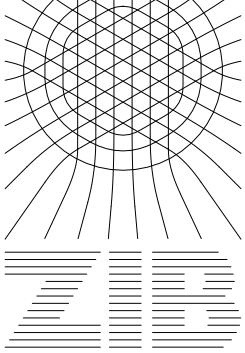
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Str. 10, D-10711 Berlin - Wilmersdorf



Wolfram Koepf

Efficient Computation of Orthogonal Polynomials in Computer Algebra

Preprint SC 95–42 (December 1995)

Efficient Computation of Orthogonal Polynomials in Computer Algebra

Wolfram Koepf

koepf@zib-berlin.de

1 Introduction

Orthogonal polynomials can be calculated by computation of determinants, by the use of generating functions, in terms of Rodrigues formulas, by iterating recurrence equations, calculating the polynomial solutions of differential equations, through closed form representations and by other means.
In computer algebra systems all these methods can be implemented. Depending on the application one might need

1. one (or many) of these polynomials in any form or specifically in expanded form,
2. the exact rational value of one of these polynomials at a certain rational point,
3. or a decimal approximation of the value of one of these polynomials at a certain point.

In this article, we give an overview about the efficiency of the above methods in Maple, Mathematica, and REDUCE. As a noncommercial package we include the MuPAD system. MuPAD is freely distributed for non-commercial use within the scientific community.
Primarily we study the implementation of the Chebyshev polynomials of the first kind as an example case.
First, we consider the builtin implementations of the Chebyshev polynomials in these systems. Next we study the classical algorithms beginning with the slow ones, and leading to the efficient ones. Finally, we finish with a new algorithm based on a divide and conquer approach which has a remarkable complexity.
In particular, we will show that

- to obtain the expanded form of one of the Chebyshev polynomials an iterative use of its power series representation is most efficient,
- for numerical purposes (both rationally exact, and decimal approximation) a divide and conquer approach that is available for Chebyshev polynomials is much preferable. This approach, however, is not efficient if the expanded form of the polynomial is needed.

We present all algorithms as short Maple programs. The other implementations of this article may be obtained from the author.

2 The Chebyshev Polynomials

The Chebyshev polynomials $T_n(x)$ of the first kind are defined by

$$T_n(\cos t) = \cos(nt)\ , \qquad \text{hence} \qquad T_n(x) = \cos(n \arccos x)\ . \tag{1}$$

They form a family of polynomials that are orthogonal with respect to the scalar product

$$\langle f, g\rangle := \int_{-1}^{1} f(x)\, g(x)\, \frac{dx}{\sqrt{1-x^2}}$$

with the weight function $(1-x^2)^{-1/2}$, and with the standardization $T_0 = 1$ and

$$\langle T_n, T_n\rangle = \int_{-1}^{1} T_n^2(x)\, \frac{dx}{\sqrt{1-x^2}} = \pi \quad (n \geq 1)\ .$$

n	Kbytes
10	0.04 kB
100	3.6 kB
1000	153 kB
2000	528 kB
3000	1364 kB

Table 1: The size of $T_n(x)$

$T_n(x)$ form polynomials with integer coefficients whose size grows rapidly with increasing n. The coefficient of x^n equals 2^{n-1}, for example. Hence the expanded polynomials need a lot of storage space. Table 1 shows the byte sizes of $T_n(x)$ in input form.
The Chebyshev polynomials have the nice property $T_n(1) = 1$. This can be used to check the accuracy of the numerical computations. For further details about these (and other families of orthogonal) polynomials we refer the reader to [2], §22, [6], [7] and [8].
All timings are given in CPU-seconds truncated to three digits, and were calculated on a SUN Sparc 10 with 85 MByte memory under SunOS 4.1.3 with the versions Maple V.3, Mathematica 2.2, REDUCE 3.6[1] and MuPAD 1.2.2. We issued the statements in separate sessions to avoid the influence of memory configurations, in particular the use of remember tables. The $\times$ sign in our tables indicates that there was no response within one hour CPU-time, or memory overflow occurred. Numerical calculations were done with 16 significant digits if not stated otherwise.
The Chebyshev and other classical families of orthogonal polynomials are accessible in Maple (`orthopoly[T]`), Mathematica (`ChebyshevT`), REDUCE (`load specfn; ChebyshevT`) and MuPAD (`orthpoly::chebyshev1`).
Table 2 shows the calculation times of $T_n(x)$ by the builtin procedures. All four systems give the ouput as expanded polynomials. Tables 3–4 show the calculation times of $T_n(1)$ in exact and approximate modes, respectively. Note that REDUCE with `on rounded` did not calculate accurate approximations for large n, indicated in Table 4 by the symbol $\diamond$. This bug is fixed by now.[2]

[1] All REDUCE calculations had been done with `lisp supersparc();` to have access to the Super-Sparc hardware arithmetic.

[2] A corresponding patch is available via anonymous ftp from `ftp.zib-berlin.de` in the directory `pub/redlib/patches`.

n	Maple	Mathematica	REDUCE	MuPAD
10	0.00	0.01	0.05	0.14
100	0.20	0.11	0.83	4.10
500	28.50	3.43[3]	41.3	116.00
1000	347.00	16.10[3]	288.00	506.00
5000	×	647.00[3]	×	×

Table 2: Builtin Chebyshev Polynomials: Calculation of $T_n(x)$

Note that the invocation of the calculation $T_n(x)$ has quite different consequences in the four systems:

Maple calculates all consecutive Chebyshev polynomials $T_k(x)$ $(k = 0, \ldots, n)$ in expanded form if $T_n(x_0)$ is issued for some x_0, and puts these in memory by the `remember` function. Hence the computation times are almost equal in any of the three different situations.

This procedure has the obvious advantage that all computed functions are immediately available afterwards. On the other hand, as a disadvantage the memory is full as soon as one has issued a single computation with high enough $n \in \mathbb{N}$ even if only this particular result is needed.

Mathematica calculates a particular $T_n(x)$ if issued, and uses no remember tables. For numerical computations, both exact and approximate, Mathematica uses a different algorithm which is much faster.

REDUCE calculates a single $T_n(x)$ if issued, and uses no remember tables.

MuPAD also calculates a single $T_n(x)$ if issued, and uses no remember tables.

n	Maple	Mathematica	REDUCE	MuPAD
10	0.02	0.00	0.05	0.12
100	0.28	0.00	0.40	4.34
500	27.90	0.00	5.28	121.00
1000	353.00	0.01	24.90	514.00
5000	×	0.08	×	×
10^4	×	0.13	×	×
10^5	×	1.28	×	×
10^6	×	12.83	×	×
10^7	×	127.00	×	×

Table 3: Builtin Chebyshev Polynomials: Calculation of $T_n(1)$

As a consequence of these considerations, Mathematica seems to have the most efficient builtin implementation of the Chebyshev (and other families of orthogonal) polynomials. On the other hand, as we will see, appropriate implementations enable Maple, REDUCE and MuPAD to calculate $T_n(x)$ for large n much faster than Mathematica.

[3] with the setting `$RecursionLimit=Infinity`. If the user doesn't redefine `$RecursionLimit`, for $n \geq 494$ no results are obtained.

Maple uses the three-term recurrence equation to obtain the polynomial list. Table 8 of § 6

n	Maple	Mathematica	REDUCE	MuPAD
10	0.00	0.01	0.05	0.07
100	0.31	0.03	◇	4.71
500	28.70	0.11	◇	125.00
1000	347.00	0.21	◇	510.00
5000	×	1.03	◇	×
10^4	×	2.05	×	×
10^5	×	21.00[4]	×	×
10^6	×	210.00[4]	×	×

Table 4: Builtin Chebyshev Polynomials: Approximation of $T_n(1)$

gives a fair comparison for this approach between the four systems, which shows that for large $n \in \mathbb{N}$ Mathematica is faster for this approach and can compute a larger list than Maple. However, since the memory and storage requirements are so immense, we think that an efficient computation of a single $T_n(x)$ is the most important task. Hence we are mainly interested to compare the efficiency of the computation of $T_n(x)$ for large n (as large as the computer memory of today's computers allow), and we do not deal with the computation of lists of all $T_k(x)$ $(k = 0, \ldots, n)$, but mainly with the computation of a single $T_n(x)$.
In the following sections, we will consider the efficiency of different approaches to calculate $T_n(x)$.

3 Determinants

The Chebyshev polynomials have the representation

$$T_n(x) = \begin{vmatrix} x & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2x & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2x & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2x & -1 \\ 0 & 0 & \cdots & 0 & -1 & 2x \end{vmatrix}$$

as the determinant of an $n \times n$ (almost) band-matrix. In Maple, this is given as

```
with(linalg):

ChebyshevT:=proc(n,x)
local f,A;
  A:=band([-1,2*x,-1],n);
  A[1,1]:=x;
  RETURN(det(A));
end:
```

The codes in Mathematica, REDUCE and MuPAD can be defined analogously.

[4] Mathematica computes the wrong result 0.0.

All classical families of orthogonal polynomials have similar representations. Expanding the above determinant yields the well-known three-term recurrence equation for $T_n(x)$ which we consider in § 6.
To calculate $T_n(x)$ via the above determinant is inherently ineffective since the computation of determinants of large matrices is very expensive. Obviously the special structure of the Chebyshev polynomials is not sufficiently utilized by this approach.

n	Maple	Mathematica	REDUCE[5]	MuPAD
10	0.45	0.16	0.03	21.00[6]
50	230.00	×	3.07	×
100	×	×	47.00	×
150	×	×	208.00	×
200	×	×	646.00	×

Table 5: Determinant Computation of $T_n(x)$

For the sake of completeness, we give the timings for the determinant approach in Table 5. Determinant computations are very slow in Maple, Mathematica, and MuPAD, whereas REDUCE calculates $T_{1000}(1)$ in 78 seconds by this approach.[7] $T_n(x)$ cannot be computed for generic x with any of the four systems for $n \geq 300$.

4 Generating Functions

The function

$$F(z) = \frac{1}{2}\left(\frac{1-z^2}{1-2xz+z^2}+1\right) = \sum_{n=0}^{\infty} T_n(x)\, z^n$$

is the generating function of the Chebyshev polynomials. By Taylor's theorem, one can therefore compute $T_n(x)$ as

$$T_n(x) = \frac{F^{(n)}(0)}{n!}\ .$$

In Maple this is given as

```
ChebyshevT:=proc(n,x)
local F,z,Dn;
  F:=(1-x*z)/(1-2*x*z+z^2);
  Dn:=diff(F,z$n);
  RETURN(subs(z=0,Dn)/n!)
end:
```

Note that other than the determinant approach the generating functions approach in principle is capable to calculate the polynomial system iteratively.
Table 6 gives the timings for the calculation of a single $T_n(x)$ with this approach. REDUCE brings each iterated derivative of $F(z)$ to a rational normal representation which is quite expensive. Maple and Mathematica do not use such normal representations, hence they are much faster. On the other hand, Maple fails very soon because of memory overflow: the

[5] with `on cramer;`.

[6] `MuPAD`'s output is not in normalized polynomial form. This normalization can be done by `normal`, but needs extra time.

[7] this time with `off cramer;`.

iterated derivatives are too large objects. This is even worse if one defines
`F:=((1-z^2)/(1-2*x*z+z^2)+1)/2`,
and hence in this case the timings are worse, too.
The generating functions approach is little better than the determinant approach in computer algebra systems without rational normal representation, but still is quite inefficient.

n	Maple	Mathematica	REDUCE[8]	MuPAD[9]
10	0.03	0.38	0.22	1.88
50	0.93	9.70	111.00	×
100	4.38	38.30	×	×
200	25.20	160.00	×	×
300	×	371.00	×	×
400	×	682.00	×	×
500	×	1101.00	×	×

Table 6: Generating Function Computation of $T_n(x)$

5 Rodrigues Formulas

The Chebyshev polynomials have the Rodrigues representation

$$T_n(x) = \frac{(-1)^n\, 2^n\, n!}{(2n)!}\, \sqrt{1-x^2}\, \frac{d^n}{dx^n}(1-x^2)^{n-1/2}\ .$$

In Maple this is given as

```
ChebyshevT:=proc(n,x)
  normal((-2)^n*n!/(2*n)!*sqrt(1-x^2)*diff((1-x^2)^(n-1/2),x$n))
end:
```

All classical families of orthogonal polynomials have a similar Rodrigues representation. The complexity is comparable to the one of the last section.
The iterated derivatives of $(1-x^2)^{n-1/2}$, however, are simpler functions than the derivatives of $F(z)$ so that the timings are better. In particular, this time the rational normal representation in REDUCE is useful since it keeps the memory size small, see Table 7.

n	Maple	Mathematica	REDUCE	MuPAD
10	0.05	0.15	0.05	2.12
100	3.70	13.60	3.85	24.90
200	23.98	60.10	19.60	127.00
300	85.60	138.00	49.80	409.00
400	×	254.00	103.00	838.00
500	×	431.00	190.00	×
1000	×	2000.00	1375.00[10]	×

Table 7: Rodrigues Formula Computation of $T_n(x)$

[8] with `off exp;`.
[9] `MuPAD`'s output is not in normalized polynomial form. This normalization can be done by `normal`, but needs extra time.
[10] with `set_heap_size 3000000;`.

6 Recurrence Equations

Now, we start to discuss the methods that are more efficient. The first such method is the use of the recurrence equation

$$T_n(x) = 2x\,T_{n-1}(x) - T_{n-2}(x) \tag{2}$$

with the initial functions

$$T_0(x) = 1 \qquad \text{and} \qquad T_1(x) = x\ .$$

Note that via (1) this recurrence equation is equivalent to the trigonometric identity

$$\cos(nt) = 2\cos t\,\cos((n-1)t) - \cos((n-2)t)\ .$$

n	Maple	Mathematica	REDUCE	MuPAD
10	0.01	0.05	0.02	0.05
100	0.31	2.31	1.17	3.55
500	29.10	53.60	28.20[11]	90.30
1000	344.00	173.00	×	×
2000	×	1246.00	×	×

Table 8: Recursive Computation of $T_n(x)$

With `remember`, we can use (2) recursively by the Maple procedure

```
ChebyshevT:=proc(n,x)
option remember;
  if n=0 then 1 elif n=1 then x
    else expand(2*x*ChebyshevT(n-1,x)-ChebyshevT(n-2,x))
  fi
end:
```

The `remember` option gives recursive programs linear complexity since all calculations are done exactly once.
Table 8 shows the timings for this approach. REDUCE generates variable stack overflow.[12]
The timings for Maple are comparable to those in Table 2, since this is Maple's builtin strategy. As already mentioned, the remember feature has the disadvantage that all previously calculated $T_k(x)$ have to be stored. Therefore the memory requirements are immense.
One might have the idea to use the recurrence equation without expanding intermediate results. Indeed, this decreases the cost by the cost of the expansion, but it generates so huge expressions that it turns out not to be a good idea at all, and the resulting expression is difficult to handle even for small n. Already T_{20} needs more than 80kB of storage space with this approach, compare Table 1. Their complicated nested structure makes any evaluation of these objects very time consuming.
The following iterative approach

```
ChebyshevT:=proc(n,x)
local T,i;
  if n=0 then 1 elif n=1 then x else
    T[-2]:=1; T[-1]:=x;
```

[11] with `set_bndstk_size(100000); lisp setq(simplimit!*,100000);`.
[12] A forthcoming version of REDUCE will include a `remember` option like the other systems.

```
    for i from 2 to n do
      T[0]:=expand(2*x*T[-1]-T[-2]);
      T[-2]:=T[-1]; T[-1]:=T[0];
    od;
  fi;
  RETURN(T[0]);
end:
```

remembers only the last two polynomials and does therefore not generate memory overflow. Hence the timings are much better.

n	Maple	Mathematica	REDUCE	MuPAD
10	0.01	0.05	0.00	0.07
100	0.26	2.16	0.44	4.74
1000	189.00	216.00	39.30	544.00
2000	1246.00	1087.00	207.00	2814.00
3000	×	2442.00	554.00	×
4000	×	×	1177.00	×

Table 9: Iterative Computation of $T_n(x)$

This is until now by far the most successful approach. All the systems do rather well, with REDUCE being most successful. On the other hand, with none of the systems one can calculate $T_{10000}(x)$ using this approach. In the following sections, we consider methods with which this is possible.

7 Differential Equations

The Chebyshev polynomial $T_n(x)$ is the unique polynomial solution of the differential equation

$$(1-x^2)\,f''(x) - x\,f'(x) + n^2\,f(x) = 0 \tag{3}$$

with the initial value

$$T_n(0) = \begin{cases} 0 & \text{if } n \text{ is odd} \\ (-1)^{n/2} & \text{if } n \text{ is even} \end{cases}.$$

In [1] a very efficient algorithm to calculate the polynomial and rational solutions of certain operator equations was published, in particular for linear ordinary differential equations with polynomial coefficients like (3).
Using the Maple implementation `ratlode` of this algorithm, written by M. Bronstein, and available in the Maple share library [3], one gets the timings of Table 10.

n	Maple
10	0.50
100	0.60
1000	7.36
10000	612.00

Table 10: Differential Equations Computation of $T_n(x)$

The results are again given as expanded polynomials.[13]
Note that this algorithm is the first one to break the complexity barrier in calculating $T_n(x)$ for $n \geq 10000$. Moreover $T_{1000}(x)$ is calculated in no more than a few seconds!
In the next section, we will see that with a more direct approach even better timings are possible.

8 Series Representations

Since $T_n(x)$ for fixed $n \in \mathbb{N}$ is a polynomial, any closed form series representation might be helpful to calculate it. Several closed form series representations for $T_n(x)$ are known of which we only utilize the Taylor expansion at $x = 0$

$$T_n(x) = \frac{n}{2} \sum_{k=0}^{[n/2]} (-1)^k \frac{(n-k-1)!}{k!\,(n-2\,k)!} (2x)^{n-2k} \; . \tag{4}$$

This representation has the advantage over others that is contains only $[n/2]$ summands rather than n. It corresponds exactly to the expanded polynomial which was the output of the preceding algorithms anyway.
Representation (4) can be calculated by the Maple procedure[14]

```
ChebyshevT:=proc(n,x)
local k,result;
  if n=0 then RETURN(1) fi;
  if n=1 then RETURN(x) fi;
  result:=0;
  for k from 0 to n/2 do
    result:=result+n/2*(-1)^k*(n-k-1)!/k!/(n-2*k)!*(2*x)^(n-2*k)
  od;
  RETURN(result);
end:
```

This implementation yields the timings of Table 11.

n	Maple	Mathematica	REDUCE	MuPAD
10	0.03	0.01	0.03	0.08
100	0.28	0.33	0.37	0.38
1000	703.00	36.60	40.00	71.80
2000	×	335.00	251.00	626.00
3000	×	1348.00	789.00	2340.00
4000	×	3406.00	1791.00	×

Table 11: Series Computation of $T_n(x)$

The timings are worse than the timings of the last section. This behaviour is due to the fact that the calculation of the coefficients

$$a_k = \frac{n}{2} \frac{(n-k-1)!}{k!\,(n-2\,k)!} (-1)^k (2x)^{n-2k}$$

[13]The algorithm expands in powers of $x - a$ for a certain a. It turns out that in the current situation $a = 0$ is chosen.

[14]If one uses the `sum` command for large n, then Maple tries to find a closed form for the sum, without success, hence we use a `for` loop. Maple's timings are much better with the `add` procedure of Maple V.4.

of $T_n(x) = \sum_{k=0}^{[n/2]} a_k$ is rather expensive: For any $k = 0, \ldots, [n/2]$ large factorials have to be calculated in both numerator and denominator, and finally the fraction has to be converted to lowest terms. Since the coefficients a_k are integers, this procedure has a large overhead which can be omitted if one calculates a_k iteratively. Since the term ratio is given by

$$\frac{a_k}{a_{k-1}} = -\frac{(n-2\,k+2)\,(n-2\,k+1)}{4\,k\,x^2\,(n-k)}\,, \tag{5}$$

the series computation (4) can be done alternatively by the Maple procedure

```
ChebyshevT:=proc(n,x)
local k,tmp,result;
  if n=0 then RETURN(1) fi;
  if n=1 then RETURN(x) fi;
  tmp:=(2*x)^n/2;
  result:=tmp;
  for k from 1 to n/2 do
    tmp:=-tmp/4/k*(n-2*k+2)*(n-2*k+1)/x^2/(n-k);
    result:=result+tmp
  od;
  RETURN(result);
end:
```

Note that this approach can always be used if polynomials are given as hypergeometric series, which applies to all classical orthogonal polynomials.
It turns out that this implementation by far is the most efficient way to calculate the expanded polynomial $T_n(x)$ for large $n \in \mathbb{N}$. Maple, REDUCE as well as MuPAD are very efficient in doing so, and leave Mathematica far behind them.
On the other hand, the timings of Tables 2 and 12 suggest that this is exactly the way how Mathematica's builtin implementation calculates the Chebyshev polynomials.[15]

n	Maple	Mathematica	REDUCE	MuPAD
10	0.00	0.01	0.03	0.02
100	0.05	0.25	0.18	0.13
1000	3.00	16.50	3.38	3.91
10000	304.00	3027.00	203.00	500.00
20000	1761.00	×	816.00	2282.00
25000	2851.00	×	1278.00[16]	×

Table 12: Iterative Series Computation of $T_n(x)$

The given iterative algorithm gives a clue why the algorithm of Abramov, Bronstein and Petkovšek [1] presented in the last section is so fast: Their algorithm is based on the iterative computation of series representations, and it calculates $T_n(x)$ similarly as considered here.

9 Divide and Conquer Approach

In this section, we leave the road of trying to find the polynomials in expanded form. Since (4) forms an alternating series with huge integer coefficients, by cancellation it cannot be used

[15] Actually, Mathematica seems to calculate a_k recursively rather than iteratively by means of (5) since `$RecursionLimit` is involved.

for numerical purposes when using decimal representations of fixed precision, and it is rather inefficient when using exact integer arithmetic.
We will find a way to calculate $T_n(x)$ very efficiently in a non-expanded form which furthermore yields also an efficient representation for numerical purposes. Therefore we utilize the formula (see e.g. [2], (22.7.24))

$$2\,T_n(x)\,T_m(x) = T_{n+m}(x) + T_{n-m}(x) \qquad (n > m)\ . \tag{6}$$

Using (6) for $m = n$ and $m = n-1$, we get the Maple implementation

```
ChebyshevT:=proc(n,x)
option remember;
  if n=0 then 1
    elif n=1 then x
    elif type(n,even) then 2*ChebyshevT(n/2,x)^2-1
    else 2*ChebyshevT((n-1)/2,x)*ChebyshevT((n+1)/2,x)-x
  fi
end:
```

This is a typical divide and conquer approach since the problem of size n is carried out by the computation of (at most) 2 subproblems of size $n/2$. With this approach it is necessary to use the remember feature since otherwise intermediate computations have to be carried out several times, resulting in exponential complexity. On the other hand for $n = 10^{15}$, e.g., only 50 iterations are necessary.
Table 13 shows the timings for this approach.

n	Maple	Mathematica	REDUCE[17]	MuPAD
1000	0.00	0.05	21.40	0.04
10^6	0.03	0.10	×	0.07
10^9	0.03	0.16	×	0.11
10^{12}	0.06	0.21	×	0.15
10^{15}	0.05	0.25	×	0.20

Table 13: Divide and Conquer Computation of $T_n(x)$

The efficiency of the method is due to the fact that it yields very small representations of $T_n(x)$ for large n. For $T_{1000}(x)$, we have for example

$$\begin{aligned} T_{1000}(\,x\,) = 2\Big(2\Big(2\Big(2\Big(2\Big(2\,&\\ &\Big(2\,\Big(2\,(\,2\,x\,(\,2\,x^2-1\,)-x\,)\,\Big(2\,(\,2\,x^2-1\,)^2-1\Big)-x\Big)\,y-x\Big)\\ &(\,2\,y^2-1\,)-x\Big)^2-1\Big)\Big(2\Big(2\\ &\Big(2\,\Big(2\,(\,2\,x\,(\,2\,x^2-1\,)-x\,)\,\Big(2\,(\,2\,x^2-1\,)^2-1\Big)-x\Big)\,y-x\Big)\\ &(\,2\,y^2-1\,)-x\Big)\Big(2\,(\,2\,y^2-1\,)^2-1\Big)\,-x\Big)\,-x\Big)^2-1\Big)^2-1\Big)^2-1 \end{aligned}$$

where y is an abbreviation for

$$y = 2\,\Big(2\,(\,2\,x^2-1\,)^2-1\Big)^2-1\ .$$

[17] with `off exp;`.

This obviously is a very compact way to write $T_{1000}(x)$, compare with Table 1. Note that expansion of these expressions cannot be done with similar efficiency as the direct approach that we considered in the preceding section.
REDUCE's internal representation makes many evaluations of the expressions computed necessary, hence the timings are bad.
Note that the given representations furthermore enable the fast rationally exact calculation of $T_n(x_0)$ for $x_0 \in \mathbb{Q}$, and not too large $n \in \mathbb{N}$, compare Table 16, e.g.[18]

$$T_{100}\left(\frac{1}{4}\right) = \frac{251213622714275047687831715137 7}{253530120045645880299340641075 2}.$$

Tables 14–15 give the timings of the exact and approximative calculations of $T_n(1)$ with the current approach.
These show that this is a very efficient way to calculate the Chebyshev polynomials accurately, in particular with rationally exact results. On the other hand, the complexity of the calculation depends heavily on the complexity of the output. Since $T_n(1) = 1$ was very simple, the calculation was done almost instantly. If we calculate $T_n(x_0)$ for rational $x_0 \neq 1$, then the result typically is a rational number with huge numerators and denominators. Hence the timings are much slower in these cases, the reason of which is the complexity of the result and not of the algorithm, though.

n	Maple	Mathematica	REDUCE	MuPAD
1000	0.03	0.05	0.06	0.04
10^6	0.03	0.10	0.20	0.08
10^9	0.03	0.18	0.40	0.11
10^{12}	0.05	0.21	0.71	0.17
10^{15}	0.06	0.25	0.96	0.19

Table 14: Divide and Conquer Computation of $T_n(1)$

n	Maple	Mathematica[19]	REDUCE	MuPAD
1000	0.03	0.06	0.08	0.03
10^6	0.01	0.16	0.27	0.08
10^9	0.05	0.20	0.56	0.10
10^{12}	0.05	0.28	1.12	0.17
10^{15}	0.06	0.31	1.61	0.20

Table 15: Divide and Conquer Approximation of $T_n(1)$

In Table 16, we present the timings for the calculation of $T_n(1/4)$, and in Table 17, the number of digits of both numerators and denominators of the corresponding results are given.
Furthermore, the method gives a very fast algorithm to compute high precision approximations for high n, e.g.[20]

$$T_{10^{15}}(0.25) = 0.72080797822908764055052380948925341839879949 68000\ldots\ .$$

Note that the algorithm is much faster than Mathematica's builtin approach, see Tables 3–4.

[18] The numerators and denominators of $T_{1000}(x_0)$ are too large to be presented here, compare Table 17.
[19] Mathematica returns the wrong result 0.0 for $n \geq 10^9$.
[20] Try to calculate this with any other method!

n	Maple	Mathematica	REDUCE	MuPAD
1000	0.03	0.05	0.27	0.07
10^4	0.48	0.13	10.10	1.85
10^5	39.70	2.08	×	179.00
10^6	×	76.10	×	×

Table 16: Divide and Conquer Computation of $T_n(1/4)$

n	numer. digits	denom. digits
1000	300	301
10^4	3010	3010
10^5	30103	30103
10^6	301029	301030

Table 17: Numerator and Denominator Size of $T_n(1/4)$

How accurate are these computations? Table 18 gives the number of correct digits of the calculations of $T_n(0.25)$, done with a precision of 16 digits, and the system specific approximate modes (`evalf` in Maple, `N` in Mathematica, `on rounded` in REDUCE, and `float` in MuPAD).

n	Maple	Mathematica	REDUCE	MuPAD
1000	14	15	18	18
10^6	11	11	15	15
10^9	7	8	11	11
10^{12}	5	6	9	10
10^{15}	1	3	6	5

Table 18: Accuracy of Approximations of $T_n(0.25)$

The table shows that the presented divide and conquer algorithm is rather well-conditioned (see e.g. [4]), hence the algorithm can be applied for quite large $n \in \mathbb{N}$, up to $n = 10^6$, say, without any further precautions.
Unfortunately, such a divide and conquer approach is not available for all classical orthogonal polynomials. The Chebyshev polynomials of the second type $U_n(x)$, however, can be calculated in a similar way by the identities (see e.g. [2], (22.6.26), (22.6.28))

$$2\,T_n(x)\,U_{m-1}(x) = U_{n+m-1}(x) + U_{m-n-1}(x) \qquad (m > n)$$

for $m = n + 1$ and

$$2\,T_n(x)\,U_{n-1}(x) = U_{2n-1}(x)\ .$$

These give the Maple implementation

```
ChebyshevU:=proc(n,x)
option remember;
  if n=0 then 1
    elif n=1 then 2*x
    elif type(n,even) then 2*ChebyshevT(n/2,x)*ChebyshevU(n/2,x)-1
    else 2*ChebyshevU((n-1)/2,x)*ChebyshevT((n+1)/2,x)
  fi
end:
```

10 Conclusion

Our article presents algorithms for the computation of orthogonal polynomials, especially Chebyshev polynomials, with which one can break the complexity barrier, and receive results that are not available with previously implemented algorithms.
Our considerations show:

1. The efficieny of a specific method does not only depend on the underlying algorithm, but also heavily on the specifics of the computer algebra system used. Here in particular the internal representation plays an important role, but also the efficiency of utilized subalgorithms (determinant computation in Table 5, computation of factorials of large integers in Table 11, ...) is an issue.

2. Efficient symbolic and efficient numeric computation mostly require different algorithms.

3. Remember options can enhance efficiency in specific situations, but often iterative programs are more adequate and faster since memory should be used carefully in computer algebra to avoid overflow.

4. For the computation of numerical values of the Chebyshev polynomials, both rationally exact, and decimal approximation, the presented divide and conquer algorithm is most efficient. The same applies to the computation of $T_n(x)$ and $U_n(x)$ if the expanded form is not required.

5. If the expanded form of an orthogonal polynomial is needed, then the iterative use of the closed form series representation is most efficient. This applies also to the computation of orthogonal polynomials for which no divide and conquer approach is available.

Acknowledgments

I would like to thank Peter Deuflhard who initiated my studies on the given topic for his encouragement and support, and Winfried Neun for his help with REDUCE.

References

[1] Abramow, S. A., Bronstein, M., Petkovšek, M.: On polynomial solutions of linear operator equations. Proc. of ISSAC 95, ACM Press, New York, 1995, 290–296.

[2] Abramowitz, M., Stegun, I. A. (1964). Handbook of Mathematical Functions. Dover Publ., New York.

[3] Bronstein, M.: `ratlode`. Maple share library, 1995.

[4] Deuflhard, P., Homann, A.: Numerical Analysis. A First Course in Scientific Computation. Walter de Gruyter, Berlin–New York, 1995.

[5] Melenk, H.: The complexity barrier in REDUCE – a case study. Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Technical Report TR 94–06, 1994.

[6] Rivlin, Th. J.: The Chebyshev Polynomials. Pure & Applied Mathematics. John Wiley & Sons, New York–London–Sydney–Toronto, 1974.

[7] Szegö, G.: Orthogonal Polynomials. Amer. Math. Soc. Coll. Publ. Vol. **23**, New York City, 1939.

[8] Tricomi, F. G.: Vorlesungen über Orthogonalreihen. Grundlehren der Mathematischen Wissenschaften **76**, Springer-Verlag, Berlin–Göttingen–Heidelberg, 1955.