1

# Efficient Computation of Chebyshev Polynomials in Computer Algebra

**Wolfram Koepf**
HTWK Leipzig, Dept. Mathematics, P. O. Box 30 00 66,
D-04251 Leipzig, Germany, koepf@imn.htwk-leipzig.de

## 1.1 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 the general purpose computer algebra systems Axiom, Macsyma, Maple, Mathematica, MuPAD and REDUCE. 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 an 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 (this is how the output is given by all the builtin commands), an iterative use of its power series representation is most efficient; the same argument applies to other classical systems of orthogonal polynomials; this is almost trivial because the classical orthogonal polynomials form hypergeometric series, but only Mathematica uses this approach;[1]

- for numerical purposes (mainly rationally exact, but also 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 programs. In each case, we choose the language with the best "asymptotic performance". This code should show that we tried to implement in as straightforward a manner as possible. The other implementations of this article may be obtained from the author.

## 1.2 The Chebyshev Polynomials

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

$$T_n(\cos t) = \cos(nt) ,$$

hence

$$T_n(x) = \cos(n \arccos x) . \tag{1.1}$$

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

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

**Table 1** The Size of $T_n(x)$

| $n$ | Kbytes |
|---|---|
| 10 | 0.04 kB |
| 100 | 1.8 kB |
| 1000 | 153 kB |
| 10000 | 15.2 MB |

$T_n(x)$ form polynomials with integer coefficients whose size grows rapidly with increasing $n$. The leading coefficient of $T_n(x)$ 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.[2]

[1] Note that new versions of MuPAD and REDUCE already contain the best codes presented in this article since a previous version of this article was widely distributed in 1996. Also in Derive's new releases, these functionalities are incorporated.

[2] Saved by Maple, spaces not counted. The space requirements grow quadratic with $n$.

The Chebyshev polynomials have the nice property that $T_n(1) = 1$. This can be used to check the accuracy of the numerical computations (both rationally exact and decimal representation). For further details about these (and other families of orthogonal) polynomials including the algorithms of this article, we refer the reader to [2]§22, [5], [6], [7], and [8].

We think that the user of a computer algebra system is mainly interested in good timings. The memory management is not of such a large interest to him besides the fact that large memory usage might influence the timings, or may even crash the system. By this reason we just compared timings and did not separately check the memory usage. We found an hour waiting time for a result acceptable.

All timings are given in CPU-seconds truncated to three digits, and for Maple, Mathematica, MuPAD and REDUCE, they were originally calculated on a SUN Sparc 10 under SunOS 4.1.3 with the releases Maple V.3, Mathematica 2.2, MuPAD 1.2.2 and REDUCE 3.6[3]. Recently the timings of Maple and Mathematica were repeated with the newest versions: Maple V.5 and Mathematica 3.0. In some instances the different releases behave quite differently, in which case we have included the timings of the new releases in the tables, and we point this out. The timings for Axiom 2.0 were done on an IBM RS 6000–320 H under AIX 3.25, and the timings for Macsyma 419.0 on a HP 9000–730 under HP–UX 9.0. All three computers have a 32 bit architecture.

For calibration purposes we used REDUCE 3.6 to calculate several Chebyshev polynomials with the different types of algorithms in this article. This is the type of calculation (with long integers, etc.) which is of interest for this article. It turned out that the time ratio SUN/HP had an arithmetic mean of 1.0. Hence, we found the timings of HP and SUN comparable. The time ratio SUN/IBM, however, had an arithmetic mean of 0.4. Hence, to make time comparison possible, we multiplied Axiom's timings on the IBM by 0.4. But obviously one should not overestimate the value of the timings, in particular since these platforms seem to perform quite differently for different questions. Rather than giving complete ratings, we were interested in showing trends.

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 (calibrated) CPU-time, or memory overflow occurred. Numerical calculations were done with 50 significant digits to check the quality of the software numerics.

The Chebyshev and other classical families of orthogonal polynomials are accessible in Axiom (`chebyshevT`), Macsyma (`load("specfun"); chebyshev_t`), Maple (`orthopoly[T]`), Mathematica (`ChebyshevT`), MuPAD (`orthpoly::chebyshev1`) and REDUCE (`load specfn; chebyshevt`).

Table 2 shows the calculation times of $T_n(x)$ by the builtin procedures. All six systems give the output as expanded polynomials. Tables 3–4 show the calculation times of $T_n(1)$ in exact and approximate modes, respectively. In Macsyma (and Maple V.5), these computations were of no value since the rewrite rule $T_n(1) = 1$ is automatically applied for $n \in \mathbb{N}$. Note that neither Maple V.3/V.4, nor MuPAD nor

[3] All REDUCE calculations had been done with `lisp supersparc();` to have access to the Super-Sparc hardware arithmetic. This is only necessary on this particular type of computer.

REDUCE could calculate accurate approximations for large $n$, indicated in Table 4 by the symbol $\diamond$.[4] This is due to the bad condition (subtractive cancellation) of the series representation utilized. In all these systems, this bug is fixed by now since a previous version of this article was widely distributed in 1996. In particular, for all computations Maple V Release 5 now uses the divide and conquer approach that we investigate in § 1.10. However, since the polynomials are still given in expanded form, the timings of Table 2 are only slightly better, see the right-most column of Table 2, but Maple gives now correct results in Tables 3–4 (also for arguments different from $x_0 = 1$). With Macsyma, one cannot compute decimal approximations for $n \geq 70$.[5]

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

| $n$ | Axiom | Macsyma | Maple V.3 | Mathematica | MuPAD | REDUCE | Maple V.5 |
|---|---|---|---|---|---|---|---|
| 10 | 0.01 | 0.06 | 0.00 | 0.01 | 0.14 | 0.05 | 0.01 |
| 100 | 0.23 | 0.63 | 0.20 | 0.11 | 4.10 | 0.83 | 0.08 |
| 500 | 6.04 | 26.30 | 28.50 | 2.60[6] | 116.00 | 41.3 | 7.92 |
| 1000 | 23.30 | 165.00 | 347.00 | 12.30[6] | 506.00 | 288.00 | 81.70 |
| 5000 | $\times$ | $\times$ | $\times$ | 418.00[6] | $\times$ | $\times$ | $\times$ |

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

| $n$ | Axiom | Maple V.3 | Mathematica | MuPAD | REDUCE |
|---|---|---|---|---|---|
| 10 | 0.00 | 0.02 | 0.00 | 0.12 | 0.05 |
| 100 | 0.01 | 0.28 | 0.00 | 4.34 | 0.40 |
| 500 | 0.02 | 27.90 | 0.00 | 121.00 | 5.28 |
| 1000 | 0.02 | 353.00 | 0.01 | 514.00 | 24.90 |
| 5000 | 0.10 | $\times$ | 0.08 | $\times$ | $\times$ |
| $10^4$ | 0.20 | $\times$ | 0.13 | $\times$ | $\times$ |
| $10^5$ | 2.12 | $\times$ | 1.28 | $\times$ | $\times$ |
| $10^6$ | 21.20 | $\times$ | 12.83 | $\times$ | $\times$ |
| $10^7$ | 205.00 | $\times$ | 127.00 | $\times$ | $\times$ |
| $10^8$ | 2059.00 | $\times$ | 1090.00 | $\times$ | $\times$ |

The invocation of the calculation $T_n(x)$ has quite different consequences in the six systems:

**Macsyma, MuPAD** and **REDUCE** calculate a single $T_n(x)$ if issued, and use no remember tables.

**Maple** V.3/V.4 calculates *all* consecutive Chebyshev polynomials $T_k(x)$ $(k =$

[4] For $n = 500$, the incorrect results have the magnitude $10^{140}$!

[5] The command `float(chebyshev_t(70,0.25));` (without using even bigfloats) creates the error message `Out of bignum stack space, (si::MULTIPLY-BIGNUM-STACK n) to grow`, whereas the command `bfloat(chebyshev_t(69,0.25));` generates a completely wrong result. Hence, Macsyma also falls in the trap of the subtractive cancellation problem.

[6] Release 2.2 was a bit slower, and one needed the setting `$RecursionLimit=Infinity`.

[7] In release 2.2, Mathematica gave the wrong result 0.0.

**Table 4** Builtin Polynomials: 50-Digits Approximation of $T_n(1.0)$

| $n$ | Axiom | Maple V.3 | Mathematica | MuPAD | REDUCE |
|---|---|---|---|---|---|
| 10 | 0.04 | 0.01 | 0.01 | 0.15 | 0.06 |
| 100 | 0.04 | 0.33 | 0.03 | 4.38 | 0.49 |
| 500 | 0.26 | ◇ | 0.11 | ◇ | ◇ |
| 1000 | 0.46 | ◇ | 0.21 | ◇ | ◇ |
| 5000 | 2.37 | ◇ | 0.98 | ◇ | ◇ |
| $10^4$ | 4.74 | ◇ | 1.96 | ◇ | ◇ |
| $10^5$ | 44.30 | ◇ | 19.60[7] | ◇ | ◇ |
| $10^6$ | 440.00 | ◇ | 196.00[7] | ◇ | ◇ |

$0,\ldots,n)$ in expanded form if $T_n(x_0)$ is issued for some $x_0$, and puts these in memory by the `remember` option. 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.

**Axiom** and **Mathematica** calculate a particular $T_n(x)$ if issued, and use no remember tables. For numerical computations, both exact and approximate, they use different algorithms that are faster, and better conditioned.

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

Maple V.3/V.4 uses the three-term recurrence equation to obtain the collection of polynomials $T_k(x)$ $(k = 0,\ldots,n)$. Table 9 of § 1.7 gives a fair comparison for this approach between the six systems, which shows that for large $n \in \mathbb{N}$, Mathematica is faster in this case 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 for this task.

### 1.3 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 Axiom, this is given as

```
ChebyshevT(n:NonNegativeInteger,x:Expression Integer):Expression Integer == _
  determinant( matrix([[ (if (i=1 and j=1) then x else if i=j then 2*x _
                 else if abs(i-j)=1 then -1 else 0) _
                 for i in 1..n] for j in 1..n]))
```

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

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 § 1.7.

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.

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

| $n$ | Axiom | Macsyma[8] | Maple | Mathematica[9] | MuPAD[10] | REDUCE[11] |
|---|---|---|---|---|---|---|
| 10 | 0.18 | 0.30 | 0.45 | 0.11 | 21.00 | 0.03 |
| 50 | 3.79 | 13.70 | 230.00 | 5.20 | × | 3.07 |
| 100 | 15.60 | 76.80 | × | 24.70 | × | 47.00 |
| 150 | 42.60 | 224.00 | × | 66.20 | × | 208.00 |
| 200 | 68.60 | 473.00 | × | 141.00 | × | 646.00 |
| 300 | 194.00 | 1566.00 | × | 464.00 | × | × |
| 500 | 637.00 | × | × | 2576.00 | × | × |
| 700 | 1278.00 | × | × | × | × | × |

[8] with `ratmx:true;`.

[9] These are the timings of Mathematica 3.0. The previous release 2.2 was *much* slower and could not compute $T_{50}(x)$ within one hour!

[10] `MuPAD`'s output is not in normalized polynomial form. This normalization can be done by `normal`, but needs extra time. A more sophisticated programming technique makes MuPAD a little faster.

[11] with `on cramer;`.

The timings for the determinant approach are given in Table 5. Determinant computations are very slow in Maple, Mathematica 2.2, and MuPAD, whereas Macsyma, Mathematica 3.0 and REDUCE are not bad. Axiom is astonishingly good, and leaves the other systems far behind. $T_n(x)$ cannot be computed for generic $x$ with any of the systems besides Axiom for $n \geq 600$ within one hour. Note that the computer algebra system Derive which is available only for IBM compatible PCs is almost as fast as Axiom (checked with an INTEL 486-100 CPU under DOS/Windows 95).[12]

## 1.4 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 Mathematica this is given as

```
ChebyshevT[n_,x_]:=Module[{F,z,Dn},
        F=((1-z^2)/(1-2*x*z+z^2)+1)/2;
        Dn=D[F,{z,n}];
        Expand[Dn/n!/.z->0]
]
```

Table 6 gives the timings for the calculation of a single $T_n(x)$ with this approach. MuPAD's derivatives of $F(z)$ are unnecessarily complicated[13], which makes their computation for high $n$ inaccessible in reasonable time and space. Axiom and REDUCE bring 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 iterated derivatives are large objects, and Maple remembers and stores all of them in memory. Remembering everything is a typical Maple feature which frequently causes problems. In the current situation, this effect can be avoided by clearing the memory ourselves with the implementation

[12] The computation of $T_{200}(x)$ took 146 sec. with Derive. Derive can calculate $T_{700}(x)$ within one hour.
[13] In Mupad 1.2.2 this defect starts already with $n = 2$, whereas in Release 1.4 it starts with $n = 3$.
[14] As always these are only the CPU times. The waiting times for the results are *much* higher. Maple seems to do nothing but garbage collection.
[15] `MuPAD`'s output is not in normalized polynomial form. This normalization can be done by `normal`, but needs extra time.
[16] with `off exp;`.
[17] These are the results of Maple V Release 5. In Release V.3 the timings were about 25% larger.

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

| $n$ | Axiom | Macsyma | Maple[14] | Mathematica | MuPAD[15] | REDUCE[16] | Maple (forget)[17] |
|---|---|---|---|---|---|---|---|
| 10 | 7.66 | 1.05 | 0.03 | 0.38 | 1.88 | 0.22 | 0.34 |
| 50 | $\times$ | 34.50 | 0.93 | 9.70 | $\times$ | 111.00 | 2.67 |
| 100 | $\times$ | 124.00 | 4.38 | 38.30 | $\times$ | $\times$ | 8.41 |
| 200 | $\times$ | 633.00 | 25.20 | 160.00 | $\times$ | $\times$ | 48.80 |
| 300 | $\times$ | 1821.00 | $\times$ | 371.00 | $\times$ | $\times$ | 153.40 |
| 400 | $\times$ | $\times$ | $\times$ | 682.00 | $\times$ | $\times$ | 306.00 |
| 500 | $\times$ | $\times$ | $\times$ | 1101.00 | $\times$ | $\times$ | 571.00 |
| 600 | $\times$ | $\times$ | $\times$ | 1627.00 | $\times$ | $\times$ | 971.00 |
| 700 | $\times$ | $\times$ | $\times$ | 2253.00 | $\times$ | $\times$ | 1658.00 |

```
ChebyshevT:=proc(n,x)
local j,F,z;
  readlib(forget);
  F:=((1-z^2)/(1-2*x*z+z^2)+1)/2;
  for j from 1 to n do
    F:=diff(F,z);
    forget(diff);
  od;
  RETURN(subs(z=0,F)/n!)
end:
```

which generates the right-most timings in Table 6: These are worse than the original ones for small $n$, but much better for large $n$, and still better than Mathematica's. This example gives a clue how much a small trick can influence the overall behavior of such an implementation.

The generating functions approach is little better than the determinant approach in computer algebra systems without rational normal representation, but still is quite inefficient.

## 1.5 Rodrigues Formulas

The Chebyshev polynomials have the Rodrigues representation

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

In REDUCE, this is given as

```
procedure ChebyshevT(n,x);
(-2)^n*factorial(n)/factorial(2*n)*sqrt(1-x^2)*df((1-x^2)^(n-1/2),x,n)$
```

All classical families of orthogonal polynomials have similar Rodrigues representations. 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 Axiom and REDUCE is useful since it keeps the memory size small, see Table 7.

Again, Maple has better behavior for large $n$ with `forget`, see the right-most column in Table 7.

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

| $n$ | Axiom | Macsyma[18] | Maple | Mathematica | MuPAD | REDUCE | Maple (forget)[19] |
|---|---|---|---|---|---|---|---|
| 10 | 0.91 | 0.35 | 0.05 | 0.15 | 2.12 | 0.05 | 0.36 |
| 100 | 16.20 | 20.00 | 3.70 | 13.60 | 24.90 | 3.85 | 6.98 |
| 200 | 78.60 | 138.00 | 23.90 | 60.10 | 127.00 | 19.60 | 27.10 |
| 300 | 224.00 | 454.00 | 85.60 | 138.00 | 409.00 | 49.80 | 63.50 |
| 400 | 511.00 | 881.00 | × | 254.00 | 838.00 | 103.00 | 126.00 |
| 500 | 1039.00 | 1631.00 | × | 431.00 | × | 190.00 | 226.00 |
| 1000 | × | × | × | 2000.00 | × | 1375.00[20] | × |

## 1.6 Matrix Powers

Now, we start to discuss methods that are more efficient. One such method was introduced in [5]. Here Richard Fateman considered the representation of the Chebyshev polynomials

$$\begin{pmatrix} T_n(x) \\ T_{n-1}(x) \end{pmatrix} = \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} T_{n-1}(x) \\ T_{n-2}(x) \end{pmatrix} = \cdots = \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} x \\ 1 \end{pmatrix}$$

by matrix powers. In REDUCE, this is given as

```
procedure ChebyshevT(n,x);
begin
  A:=mat((2*x,-1),(1,0));
  b:=mat((x),(1));
  A:=A^(n-1);
  A:=A*b;
  return(A(1,1));
end$
```

Whereas Maple Release V.3 was rather slow and could not compute $T_{2000}(x)$ within one hour, Release V.5 beats REDUCE:

```
with(linalg);
ChebyshevT:=proc(n,x)
local b,c,A;
```

[18] Here we use of the rational normal form `rat`. This is most efficient.
[19] These are the results of Maple V Release 5. In Release V.3, the timings were about 25% larger.
[20] with `set_heap_size 3000000;`.

```
  A:=array([[2*x,-1],[1,0]]);
  b:=vector([x,1]);
  A:=evalm(A^(n-1));
  c:=linalg[multiply](A,b);
  RETURN(c[1]);
end:
```

**Table 8** Calculation of $T_n(x)$ by Matrix Powers

| $n$ | Axiom | Macsyma | Maple[21] | Mathematica | MuPAD | REDUCE | Macs. (`matrixpower`) |
|---|---|---|---|---|---|---|---|
| 10 | 0.22 | 0.12 | 0.14 | 0.06 | 7.06 | 0.01 | 0.09 |
| 100 | 0.70 | × | 0.72 | 4.01 | 24.10 | 0.37 | 14.40 |
| 500 | 19.90 | × | 10.80 | 111.20 | 151.00 | 12.20 | 577.00 |
| 1000 | 153.00 | × | 38.20 | 607.00 | 538.00 | 62.20 | × |
| 2000 | 1722.00 | × | 135.00 | 4211.00 | × | 336.00 | × |
| 3000 | × | × | 291.00 | × | × | 982.00 | × |
| 4000 | × | × | 479.00 | × | × | 2201.00 | × |

Since matrix powers can be calculated by iterative squaring, a typical divide and conquer approach, it is interesting to check which of the systems provide this type of implementation. It turns out that most systems calculate matrix powers by this approach. Only Macsyma does not use this technique, hence it fails very soon, see Table 8. Using the implementation

```
matrixpower(A,n):=block([B],
  if n=1 then return(A),
  if floor(n/2)=n/2 then
    (B:matrixpower(A,n/2),
    return(B.B))
  else return(matrixpower(A,n-1).A)
)$
```

for the computation of matrix powers makes Macsyma much faster, although not competitive with the other systems, see the right-most column in Table 8.

Note that the approach of this section cannot be generalized to the other systems of orthogonal polynomials (besides the Chebyshev polynomials $U_n(x)$ of the second type). Its availability depends heavily on the fact that the coefficients of the recurrence equation of the Chebyshev polynomials, which will be considered next, do not depend on $n$ [5].

## 1.7 Recurrence Equations

In this section, we discuss the use of the recurrence equation

$$T_n(x) = 2x\,T_{n-1}(x) - T_{n-2}(x) \tag{1.2}$$

[21] These are the timings of Maple V Release 5. Release V.3 was *much* slower and could not compute $T_{2000}(x)$ within one hour!

with the initial functions

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

Note that via (1.1) this recurrence equation is equivalent to the trigonometric identity

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

Using a remember table, we can use (1.2) recursively by the Mathematica procedure

```
ChebyshevT[n_,x_]:=ChebyshevT[n,x]=
If[n==0,1,If[n==1,x,Expand[2*x*ChebyshevT[n-1,x]-ChebyshevT[n-2,x]]]]
```

The use of remember tables gives recursive programs linear complexity since all calculations are done exactly once.

**Table 9** Recursive Computation of $T_n(x)$

| $n$ | Axiom | Macsyma | Maple | Mathematica | MuPAD[22] | REDUCE |
|---|---|---|---|---|---|---|
| 10 | 0.51 | 0.06 | 0.01 | 0.05 | 0.03 | 0.02 |
| 100 | $\times$ | $\times$[23] | 0.31 | 2.31 | 0.97 | 1.17 |
| 500 | $\times$ | $\times$ | 29.10 | 53.60 | 18.60 | 28.20[24] |
| 1000 | $\times$ | $\times$ | 344.00 | 173.00 | 86.80 | $\times$ |
| 2000 | $\times$ | $\times$ | $\times$ | 1246.00 | $\times$ | $\times$ |

Table 9 shows the timings for this approach. REDUCE generates variable stack overflow since it does not have a `remember` feature.

The timings for Maple are comparable to those in Table 2, since this is Maple V.3'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. If the user needs the complete list $T_k(x)$ $(k = 0, \ldots, n)$, then this recursive approach using `remember` is most efficient.

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 (in input format) with this approach, compare Table 1. Their complicated nested structure makes any evaluation of these objects very time consuming.

The following iterative approach

```
ChebyshevT(n:NonNegativeInteger,x:Variable x):Polynomial Integer == _
( _
```

[22] with MuPAD's type `poly`.
[23] Macsyma generates the error message `Bind stack overflow`.
[24] with `set_bndstk_size(100000); lisp setq(simplimit!*,100000);`.

```
  if n=0 then return(1) else _
    if n=1 then return(x) else ( _
      T2:=1; T1:=x; _
      for i in 2..n repeat ( _
        T0:=2*x*T1-T2; _
        T2:=T1; T1:=T0 ); _
        return T0 ) _
)
```

in Axiom remembers only the last two polynomials and does therefore not generate memory overflow, see Table 10. Since Axiom and REDUCE have a polynomial normal representation, there is no need to use a high level language procedure like `Expand`, hence the timings are much better than in the other systems.

This is until now the most successful approach for the calculation of a single $T_n(x)$. All the systems do rather well. On the other hand, with none of the systems can one calculate $T_{10000}(x)$ (within the proposed one hour of computing time) using this approach. In the following sections, we consider methods with which this is possible.

**Table 10** Iterative Computation of $T_n(x)$

| $n$ | Axiom | Macsyma[25] | Maple | Mathematica | MuPAD | REDUCE |
|---|---|---|---|---|---|---|
| 10 | 0.19 | 0.04 | 0.01 | 0.05 | 0.04 | 0.00 |
| 100 | 1.66 | 2.18 | 0.26 | 2.16 | 0.87 | 0.44 |
| 1000 | 37.90 | 395.00 | 189.00 | 216.00 | 84.00 | 39.30 |
| 2000 | 137.00 | 1578.00 | 1246.00 | 1087.00 | 798.00 | 207.00 |
| 3000 | 362.00 | × | × | 2442.00 | × | 554.00 |
| 4000 | 671.00 | × | × | × | × | 1177.00 |
| 5000 | 1201.00 | × | × | × | × | 1523.00 |

## 1.8 Differential Equations

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

$$(1-x^2)\,f''(x) - x\,f'(x) + n^2\,f(x) = 0 \tag{1.3}$$

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 (1.3).

[25] Here we use of the rational normal form `rat`. This is most efficient.

Using the Maple implementation `ratlode` of this algorithm, written by M. Bronstein [3], one gets the timings of Table 11.

**Table 11** Differential Equations Computation of $T_n(x)$

| $n$ | Maple |
|---:|---:|
| 10 | 0.50 |
| 100 | 0.60 |
| 1000 | 7.36 |
| 10000 | 612.00 |

The results are again given as expanded polynomials.[26]

Note that this algorithm is the first one to enable the calculation of $T_n(x)$ for $n \geq 10000$ within an hour. 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.

## 1.9 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 $x = 0$

$$T_n(x) = \frac{n}{2} \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \frac{(n-k-1)!}{k!\,(n-2\,k)!}\,(2x)^{n-2k} \,. \tag{1.4}$$

This representation has the advantage over other series representations that it requires only $n/2$ additions rather than $n$. Hence other series representations are less efficient.

Representation (1.4) corresponds exactly to the expanded polynomial which was the output of the preceding algorithms anyway. It can be calculated by the REDUCE procedure

```
procedure ChebyshevT(n,x);
begin
scalar k;
return(for k:=0:floor(n/2)
sum n/2*(-1)^k*factorial(n-k-1)/factorial(k)/factorial(n-2*k)*(2*x)^(n-2*k))
end$
```

This implementation yields the timings of Table 12.
Axiom and REDUCE have the most efficient factorial calculation. This is why they

[26] The algorithm expands in powers of $x - a$ for a certain $a$. It turns out that in the current situation $a = 0$ is chosen.
[27] These are the results of Maple V Release 5. Release V.3 was slower.

**Table 12** Series Computation of $T_n(x)$

| $n$ | Axiom | Macsyma | Maple[27] | Mathematica | MuPAD | REDUCE |
|---|---|---|---|---|---|---|
| 10 | 0.94 | 0.04 | 0.00 | 0.01 | 0.09 | 0.03 |
| 100 | 4.78 | 0.92 | 0.15 | 0.33 | 0.38 | 0.37 |
| 1000 | 60.50 | 143.00 | 253.00 | 36.60 | 70.90 | 40.00 |
| 2000 | 316.00 | 1282.00 | 2549.00 | 335.00 | 602.00 | 251.00 |
| 3000 | 823.00 | × | × | 1348.00 | 2150.00 | 789.00 |
| 4000 | 1868.00 | × | × | 3406.00 | × | 1791.00 |
| 5000 | 3756.00 | × | × | × | × | 3696.00 |

succeed in Table 12. This time, Maple's problem is not the memory, but its factorial computation is rather inefficient.

The timings are much worse than the timings of the last section. This behavior is due to the fact that the calculation of the summands

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

of $T_n(x) = \sum\limits_{k=0}^{\lfloor n/2 \rfloor} a_k$ is rather expensive: For any $k = 0, \ldots, \lfloor n/2 \rfloor$, 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

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

are integers, this procedure has a large overhead. To get better timings, we can replace the factorials by a binomial coefficient

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

In Macsyma, this yields the implementation

```
ChebyshevT(n,x):=block([k,result],
        if n=0 then return(1) else
        if n=1 then return(x) else
        (result:0,
        for k:0 thru (n-1)/2 do
        result:result+n/2*(-1)^k/(n-2*k)*binomial(n-k-1,k)*(2*x)^(n-2*k),
        if floor(n/2)=n/2 then result:result+(-1)^(n/2),
        return(result))
)$
```

In Axiom, MuPAD and REDUCE, this is unfortunately less efficient than the factorial approach. Maple has an improved timing, but still severe problems, see Table 13. However, Macsyma's and Mathematica's binomial coefficient implementations are rather efficient and generate an impressive speed-up.

**Table 13** Series Computation of $T_n(x)$ with Binomial Coefficients

| $n$ | Macsyma | Maple | Mathematica |
|---|---|---|---|
| 10 | 0.03 | 0.01 | 0.01 |
| 100 | 0.71 | 0.20 | 0.20 |
| 1000 | 23.30 | 64.50 | 11.00 |
| 2000 | 116.00 | 1049.00 | 66.80 |
| 3000 | 279.00 | × | 214.00 |
| 4000 | 555.00 | × | 489.00 |
| 5000 | 996.00 | × | 1060.00 |
| 6000 | 1546.00 | × | 1772.00 |

However, much more efficient is the following approach avoiding the computation of factorials or binomial coefficients by calculating $a_k$ iteratively. Since the term ratio is given by

$$\frac{a_k}{a_{k-1}} = -\frac{(n-2\,k+2)\,(n-2\,k+1)}{4\,k\,x^2\,(n-k)}\,, \tag{1.5}$$

the series computation (1.4) can be done alternatively by the MuPAD procedure

```
ChebyshevT:=proc(n,x)
local k,tmp,result;
begin
  if n=0 then return(poly(1,[x])) end_if;
  if n=1 then return(poly(x,[x])) end_if;
  tmp:=poly((2*x)^n/2,[x]);
  result:=tmp;
  for k from 1 to n/2 do
    tmp:=tmp*poly(-(n-2*k+2)*(n-2*k+1),[x]);
    tmp:=divide(tmp,poly(4*k*(n-k)*x^2,[x]),Exact);
    result:=result+tmp
  end_for;
  return(result);
end_proc:
```

using only polynomial arithmetic. Note that this approach can always be used if polynomials are given as hypergeometric series, which applies to all classical orthogonal polynomial systems.
It turns out that this is by far the most efficient way to calculate the expanded polynomial $T_n(x)$ for large $n \in \mathbb{N}$. Maple, MuPAD as well as REDUCE are very efficient in doing so, and leave Mathematica far behind them. MuPAD is most efficient only if one uses the type `poly` (and does not work with expressions) since then its fast polynomial arithmetic is used.

The timings of Tables 2 and 14 suggest that the present method is exactly how Mathematica's builtin implementation calculates the Chebyshev polynomials. Derive

[28] with MuPAD's type `poly`. Without using this type, the calculation times are ten times slower for large $n$.
[29] with `set_heap_size 10000000;`.

**Table 14** Iterative Series Computation of $T_n(x)$

| $n$ | Axiom | Macsyma | Maple | Mathematica | MuPAD[28] | REDUCE | Math. (`Apply`) |
|---|---|---|---|---|---|---|---|
| 10 | 0.59 | 0.02 | 0.00 | 0.01 | 0.03 | 0.03 | 0.01 |
| 100 | 4.16 | 0.54 | 0.05 | 0.25 | 0.17 | 0.18 | 0.11 |
| 1000 | 39.70 | 9.77 | 3.00 | 16.50 | 2.24 | 3.38 | 2.88 |
| 10000 | 551.00 | 361.00 | 304.00 | 3027.00 | 62.30 | 210.00 | 1046.00 |
| 20000 | 1703.00 | $\times$ | 1761.00 | $\times$ | 210.00 | 816.00 | $\times$ |
| 25000 | $\times$ | $\times$ | 2851.00 | $\times$ | 326.00 | 1278.00[29] | $\times$ |
| 30000 | $\times$ | $\times$ | $\times$ | $\times$ | 470.00 | $\times$ | $\times$ |

again turns out to be as fast as the fastest systems here.[30]

Note that the implementations of this section using `for` loops are not optimal since then the sum has to be restructured iteratively. This effect can be avoided using lists as in the Mathematica code

```
ChebyshevT[n_,x_]:=Module[{k,tmp,tab},
        If[n==0,Return[1]];
        If[n==1,Return[x]];
        tmp=(2*x)^n/2;
        tab=Table[tmp=-tmp/4/k*(n-2*k+2)*(n-2*k+1)/x^2/(n-k),{k,1,Floor[n/2]}];
        (2*x)^n/2+Apply[Plus,tab]
]
```

In Maple V.3, this measure did not increase the efficiency significantly despite the message in Maple's help page of the `seq` command.[31] On the other hand, Mathematica's code can be significantly accelerated by the above code, see Table 14, right column. This shows that Mathematica's `Do` construct is quite inefficient. It turns out, however, that this new code in Mathematica is really fast only together with a 64 bit word size, for example on a DEC Alpha workstation, generating $T_{30000}(x)$ in less than 100 seconds!

## 1.10 Divide and Conquer Approach

In this section, we leave the road of trying to find the polynomials in expanded form. Since (1.4) forms an alternating series with huge integer coefficients, by cancellation it cannot be used 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

[30] The computation of $T_{4000}(x)$ took 12.4 sec. with Derive. Derive can calculate $T_{10000}(x)$ in less than a minute.

[31] Maple's message is: "In either form, the seq version is more efficient than the for-loop version because the for-loop version constructs many intermediate sequences. Specifically, the cost of the seq version is linear in the length of the sequence generated but the for-loop version is quadratic." In Maple V.5, the `seq` command indeed gives better timings.

furthermore yields also an efficient representation for numerical purposes.[32] Therefore, we utilize the formula (see e.g., [2](22.7.24), or also [5])

$$2\,T_n(x)\,T_m(x) = T_{n+m}(x) + T_{n-m}(x) \qquad (n \geq m)\ . \tag{1.6}$$

Using (1.6) 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 makes sense to use the remember feature since otherwise intermediate computations have to be carried out several times, resulting in exponential complexity.[33] On the other hand, for $n = 10^{15}$, e.g., only 50 iterations are necessary, hence the use of the remember option does not cause memory problems. Table 15 shows the timings for this approach.

**Table 15** Divide and Conquer Computation of $T_n(x)$

| $n$ | Axiom | Macsyma | Maple | Mathematica | MuPAD | REDUCE[34] |
|---|---|---|---|---|---|---|
| 1000 | 19.10 | 0.07 | 0.00 | 0.05 | 0.04 | 21.40 |
| $10^6$ | × | 0.13 | 0.03 | 0.10 | 0.07 | × |
| $10^9$ | × | 0.45 | 0.03 | 0.16 | 0.11 | × |
| $10^{12}$ | × | × | 0.06 | 0.21 | 0.15 | × |
| $10^{15}$ | × | × | 0.05 | 0.25 | 0.20 | × |

To get more detailed information about the handling of these expressions by the different systems, we substituted $x = 1$ in the results, giving Table 16.

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

$$T_{1000}(x) = 2\Big(2\Big(2\Big(2\Big( \\ 2\,\big(2\,\big(2\,\big(2\,(\,2\,x\,(\,2\,x^2-1\,)-x\,)\,\big(2\,(\,2\,x^2-1\,)^2-1\big)-x\big)\;y-x\big)\;(\,2\,y^2-1\,)-x\big)^2-1$$

[32] For purely numerical calculations, there may be more efficient methods. These cannot be used to compute rationally exact results, though. This type of numerical calculations are not our primary concern. Still, the efficiency of our approach is not bad.

[33] With little effort, one can rewrite the procedure iteratively to avoid the `remember` option.

[34] with `off exp;`.

**Table 16** Substitution of $x=1$ in $T_n(x)$

| $n$ | Axiom | Macsyma | Maple | Mathematica | MuPAD | REDUCE |
|---|---|---|---|---|---|---|
| 1000 | 0.14 | 0.01 | 0.00 | 0.01 | 0.01 | 0.05 |
| $10^6$ | × | 0.45 | 0.05 | 0.20 | 0.24 | × |
| $10^9$ | × | 14.10 | 1.43 | 7.51 | 7.62 | × |
| $10^{12}$ | × | × | 27.60 | 145.00 | 157.00 | × |
| $10^{15}$ | × | × | 255.00 | × | 1216.00 | × |

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

where $y$ is an abbreviation for

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

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 in the direct approach that we considered in the preceding section.[35]

For large enough $n$, Macsyma generates the error message `Bind stack overflow`. REDUCE's failure has two reasons: on the one hand it misses the remember option, but more decisively even with `off exp;` and `off factor;`, it iteratively generates "normal forms" making many evaluations of the expressions computed necessary, and being very costly. The same comment applies to Axiom. In such cases, it should be possible to turn off the rational normal representation.

**Table 17** Divide and Conquer Computation of $T_n(1)$

| $n$ | Axiom | Macsyma | Maple | Mathematica[36] | MuPAD | REDUCE |
|---|---|---|---|---|---|---|
| $10^{10}$ | 0.18 | × | 0.05 | 0.18 | 0.14 | × |
| $10^{100}$ | ×[37] | × | 0.50 | 1.90 | 1.39 | × |
| $10^{1000}$ | × | × | 24.40 | 25.05 | × | × |
| $10^{2000}$ | × | × | 91.90 | × | × | × |

Tables 17–18 give the timings of the exact and approximative calculations (50 digits) of $T_n(1)$ with the current approach. Mathematica computes the wrong approximation 0.0, indicated by the $\diamond$, although Mathematica 3.0 claims to keep track of error bounds.

These computations show that this is a very efficient way to calculate the Chebyshev polynomials accurately, in particular with rationally exact results. On the other hand,

[35] Maple V.5 computes $T_n(x)$ by expanding the intermediate results in the divide and conquer computation; compare the timings given in Table 2.
[36] with `$RecursionLimit=Infinity`. For $n=10^{2000}$, a segmentation fault occurs.
[37] Axiom generates the error message `Invocation history stack overflow`.

**Table 18** 50 Digits Divide and Conquer Approximation of $T_n(1.0)$

| $n$ | Axiom | Macsyma | Maple | Mathematica | MuPAD | REDUCE |
|---:|---:|---:|---:|---:|---:|---:|
| $10^{10}$ | 0.18 | $\times$ | 0.03 | 0.23 | 0.12 | 0.66 |
| $10^{100}$ | $\times$ | $\times$ | 0.95 | $\diamond$ | 1.66 | $\times$ |
| $10^{1000}$ | $\times$ | $\times$ | 30.70 | $\diamond$ | $\times$ | $\times$ |
| $10^{2000}$ | $\times$ | $\times$ | 109.00 | $\times$ | $\times$ | $\times$ |

the complexity of the calculation depends heavily on the complexity of the output. Since $T_n(1) = 1$ is very simple, the calculation is 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.

Nevertheless, the given implementations 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 19, e.g.[38]

$$T_{100}\left(\frac{1}{4}\right) = \frac{2512136227142750476878317151377}{2535301200456458802993406410752} .$$

In Table 19, we present the timings for the calculation of $T_n(1/4)$, and in Table 20, the number of digits of both numerators and denominators of the corresponding results are given.

**Table 19** Divide and Conquer Computation of $T_n(1/4)$

| $n$ | Axiom | Macsyma | Maple[39] | Mathematica | MuPAD | REDUCE |
|---:|---:|---:|---:|---:|---:|---:|
| 1000 | 0.10 | 0.12 | 0.02 | 0.05 | 0.07 | 0.27 |
| $10^4$ | 1.64 | 0.63 | 0.13 | 0.13 | 1.85 | 10.10 |
| $10^5$ | 106.00 | $\times$ | 3.71 | 2.08 | 179.00 | $\times$ |
| $10^6$ | $\times$ | $\times$ | 128.00 | 76.10 | $\times$ | $\times$ |
| $10^7$ | $\times$ | $\times$ | $\times$ | 2882.00 | $\times$ | $\times$ |

**Table 20** Numerator and Denominator Size of $T_n(1/4)$

| $n$ | numer. digits | denom. digits |
|---:|---:|---:|
| 1000 | 300 | 301 |
| $10^4$ | 3 010 | 3 010 |
| $10^5$ | 30 103 | 30 103 |
| $10^6$ | 301 029 | 301 030 |
| $10^7$ | 3 010 300 | 3 010 300 |

[38] The numerators and denominators of $T_{1000}(x_0)$ are too large to be presented here, compare Table 20.

[39] These are the timings of Maple V Release 5. Release V.3 was *much* slower and could not compute $T_{10^6}(x)$ within one hour!

**Table 21** Accuracy of 50-Digit Approximations of $T_n(0.25)$

| $n$ | Axiom | Macsyma | Maple | Mathematica | MuPAD | REDUCE |
|---|---|---|---|---|---|---|
| 1000 | 48 | 50 | 47 | 43 | 50 | 50 |
| $10^6$ | 46 | 47 | 44 | 39 | 50 | 50 |
| $10^9$ | 42 | 44 | 40 | 35 | 49 | 45 |
| $10^{12}$ | $\times$ | $\times$ | 38 | 33 | 47 | 44 |
| $10^{15}$ | $\times$ | $\times$ | 34 | 29 | 44 | 40 |

Furthermore, the method gives a very fast algorithm to compute high precision approximations for high $n$, e.g.[40]

$$T_{10^{15}}(0.25) = 0.72080797822908764055052380948925341839879949680000...$$

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

How accurate are these computations? Table 21 gives the number of correct digits of the calculations of $T_n(0.25)$, done with a precision of 50 digits, and the system specific approximate modes (`Float` in Axiom, `bfloat` in Macsyma, `evalf` in Maple, `N` in Mathematica, `on rounded` in REDUCE, and `float` in MuPAD).

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 $n \in \mathbb{N}$ without further precautions.

Unfortunately, such a divide and conquer approach is not available for all classical orthogonal polynomials. The Chebyshev polynomials $U_n(x)$ of the second type, 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:
```

[40] Try to calculate this with another method of your choice!

## 1.11 Conclusion

Our article presents algorithms for the computation of orthogonal polynomials, especially Chebyshev polynomials, with which one can receive results that are not available with previously implemented algorithms.

Our considerations show:

1. None of the general purpose systems considered had the "best" algorithms implemented. For all of the systems, considerable speed-up could be obtained by the implementation of better algorithms, for symbolic as well as numerical computations.
2. New versions of MuPAD and REDUCE already contain the best codes presented in this article since a previous version of this article was widely distributed in 1996. Also, in Derive's new releases, these functionalities are incorporated.
3. The efficiency 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 (mainly the use of rational normal representations) plays an important role, but also the efficiency of utilized subalgorithms (determinant computation in Table 5, computation of factorials of large integers in Table 12, ...) is an issue.
4. Efficient symbolic and efficient numeric computation often require different algorithms.
5. 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.
6. For the rationally exact computation of numerical values of the Chebyshev polynomials, the presented divide and conquer algorithm is most efficient. It it also well-conditioned and obtains decimal approximations rather fast. If the expanded form is not required, this algorithm also efficiently computes $T_n(x)$ and $U_n(x)$ generically.
7. If the expanded form of an orthogonal polynomial is needed, then the iterative use of the closed form series representation (Table 14) is most efficient, and all the systems can compute $T_{10000}(x)$ by this approach. The same technique applies also to the computation of the other classical families of orthogonal polynomials.

**Acknowledgments**

I would like to thank Peter Deuflhard who initiated my studies on the given topic for his encouragement and support, Winfried Neun for his help with REDUCE, and Richard Fateman who called my attention to his paper [5], and gave important remarks on Macsyma. Furthermore, the comments and suggestions of Harald Böing and Jochen Fröhlich were very helpful. Finally, I would like to thank Michael Wester for his encouragement and comments.

# References

Abramov, Sergei A., Bronstein, Manuel, Petkovšek, Marko: On polynomial solutions of linear operator equations. Proc. of ISSAC 95, ACM Press, New York, 1995, 290–296.

Abramowitz, Milton and Stegun, Irene A. (1964). *Handbook of Mathematical Functions.* Dover Publ., New York.

Bronstein, Manuel: Maple package `ratlode`, private communication.

Deuflhard, Peter, Homann, Andreas: *Numerical Analysis. A First Course in Scientific Computation.* Walter de Gruyter, Berlin–New York, 1995.

Fateman, Richard: Lookup tables, recurrences and complexity. Proc. of ISSAC 89, ACM Press, New York, 1989, 68–73.

Rivlin, Theodore J.: *The Chebyshev Polynomials.* Pure & Applied Mathematics. John Wiley & Sons, New York–London–Sydney–Toronto, 1974.

Szegő, Gábor: *Orthogonal Polynomials.* Amer. Math. Soc. Coll. Publ. Vol. **23**, New York City, 1939.

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