Chapter 34

Numerical evaluation of power series

We give algorithms for the numerical evaluation of power series. If the series coefficients are rational, the binary splitting (binsplit) algorithm can be applied for rational arguments and the rectangular schemes for real (full-precision) arguments. As a special case of the binary splitting algorithm, a method for fast radix conversion is described. Finally we describe a technique for the summation of series with alternating coefficients.

34.1 The binary splitting algorithm for rational series

The straightforward computation of a series for which each term adds a constant amount of precision (for example, the arc-cotangent series with arguments \(>1\)) to a precision of \(N\) digits involves the summation of \(O(N)\) terms. To get \(N\) bits of precision one has to add \(O(N^2)\) terms of the sum, each term involves one (length-\(N\)) short division (and one addition). Therefore the total work is \(O(N^3)\), which makes it impossible to compute billions of digits from linearly convergent series even if they are as ‘good’ as Chudnovsky’s famous series for \(\pi\) (given in \([102]\)):

\[
\frac{1}{\pi} = \frac{6546168160}{\sqrt{640320}} \sum_{k=0}^{\infty} \left( \frac{13591409}{545140134} + k \right) \left( \frac{(6k)!}{(k!)^3 (3k)!} \frac{(-1)^k}{640320^{3k}} \right) \quad (34.1-1a)
\]

\[
= \frac{12}{\sqrt{640320}} \sum_{k=0}^{\infty} (-1)^k \frac{(6k)!}{(k!)^3 (3k)!} \frac{13591409 + 545140134 \cdot k}{640320^{3k}} \quad (34.1-1b)
\]

34.1.1 Binary splitting scheme for products

34.1.1.1 Computation of the factorial

We motivate the binsplit algorithm by giving the analogue for the fast computation of the factorial. Define \(f_{m,n} := m \cdot (m+1) \cdot (m+2) \cdots (n-1) \cdot n\), then \(n! = f_{1,n}\). We compute \(n!\) by recursively using the relation \(f_{m,n} = f_{m,x} \cdot f_{x+1,n}\) where \(x = \lfloor (m+n)/2 \rfloor\):

```plaintext
1 indent(i)=for(k=1,8*i,print1(" ")); \ aux: print 8*i spaces
2 3 F(m, n, i=0)=
4 { /* Factorial, self-documenting */
5  local(x, ret);
6   indent(i); print("F("m", "n", ");
7     if (m==n, /* then: */
8       ret = m; \ \ == F(m,m)
9       , /* else: */
10       x = floor( (m+n)/2 );
11       ret = F(m, x, i+1) * F(x+1, n, i+1);
12     );
13   indent(i); print("^== ", ret);
14   return( ret );
15 }
```

The function prints the intermediate values occurring in the computation. The additional parameter \(i\) keeps track of the calling depth, used with the auxiliary function \(indent()\). Figure 34.1-A shows the output with the computation of \(8!\) = \(F(1,8)\). A fragment like
Figure 34.1-A: Quantities with the computation of $8!$. 

\begin{verbatim}
F(5, 6)  
F(5, 5)  ^= 5  
F(6, 6)  ^= 6  
-= 30
\end{verbatim}

says "F(5,6) called F(5,5) [which returned 5], then called F(6,6) [which returned 6]. Then F(5,6) returned 30." For the computation of other products modify the line \texttt{ret=m;} as indicated in the code.

Note that we compute the product in a depth-first fashion to obtain a localized memory access. An implementation of the scheme by computing products of pairs, pairs of pairs, etc., gives the identical result but is likely to suffer from cache problems.

### 34.1.1.2 Computation of a polynomial from its roots

Given the $n$ roots $a_i$ of a polynomial $C = \sum_{j=0}^{n} c_j x^j$ we can compute $C$ by a trivial modification of the routine above:

```c
1 F(m, n, i=0)=
2 { /* Polynomial by roots, self-documenting */
3  \[--snip--\]
4    \[--snip--\]
5    \[--snip--\]
6    \[--snip--\]
7 }
```

Here we choose the roots to be $a_i = i$. The quantities with the computation of $C = \prod_{i=1}^{8} (x - i)$ are shown in figure 34.1-B. The coefficient of this particular polynomial are the (signed) Stirling numbers of the first kind, see figure 11.1-A on page 277.

### 34.1.2 Binary splitting scheme for sums

For the evaluation of a sum $\sum_{k=0}^{N-1} a_k$ we use the ratios $R_k$ of consecutive terms:

$$R_k := \frac{a_k}{a_{k-1}}$$ (34.1-2)