EXTENDED RATE, MORE GFUN

WALDEMAR HEBISCH AND MARTIN RUBEY

ABSTRACT. We present a software package that guesses formulae for sequences of, for example, rational numbers or rational functions, given the first few terms. We implement an algorithm due to Bernhard Beckermann and George Labahn, together with some enhancements to render our package efficient. Thus we extend and complement Christian Krattenthaler's program `Rate.m`, the parts concerned with guessing of Bruno Salvy and Paul Zimmermann's `GFUN`, the univariate case of Manuel Kauers' `Guess.m` and Manuel Kauers' and Christoph Koutschan's `qGeneratingFunctions.m`.

1. INTRODUCTION

For some a brain-teaser, for others one step in proving their next theorem: given the first few terms of a sequence of, say, integers, what is the next term, what is the general formula? Of course, no unique solution exists, but, by Occam's razor, we will prefer a 'simple' formula over a more 'complicated' one. In this article we present a new package that aims at finding such a simple formula, written for the computer algebra system `FriCAS`.[1]

Some sequences are very easy to 'guess', like

$$\tag{1} 0, 1, 4, 9, \ldots,$$

or

$$\tag{2} 1, 1, 2, 3, 5, \ldots.$$

Others are a little harder, for example

$$\tag{3} 0, 1, 3, 9, 33, \ldots.$$

Of course, at times we might want to guess a formula for a sequence of polynomials or rational functions, too:

$$\tag{4} 1, 1 + q + q^2, (1 + q + q^2)(1 + q^2), (1 + q^2)(1 + q + q^2 + q^3 + q^4) \ldots,$$

or

$$\tag{5} \frac{1 - 2q}{1 - q}, 1 - 2q, (1 - q)(1 - 2q)^3, (1 - q)^2(1 - 2q)(1 - 2q - 2q^2)^3, \ldots$$

Fortunately, with the right tool, it is a matter of a moment to figure out formulae for all of these sequences. In this article we describe a computer program that encompasses well known techniques and adds new ideas that we hope to be effective. In particular, we generalise both Christian Krattenthaler's program `Rate.m` [20], and the guessing functions present in `GFUN` written by Bruno Salvy and

[1]`FriCAS` is freely available at `http://fricas.sourceforge.net`.

Paul Zimmermann [25], and in `qGeneratingFunctions.m` by Manuel Kauers and Christoph Koutschan [16, 17]. With a little manual aid, we can guess multivariate formulae as well, along the lines of Doron Zeilberger's programs `GuessRat` and `GuessHolo` [35, 36], or Manuel Kauers' program `Guess.m` [15]. All these programs, as well as the one presented here, try to *compute* a function that yields the terms when evaluated at 0, 1, 2, and so on. We describe this computational approach in more detail beginning in Section 5.

A completely different idea is pursued by *The online encyclopedia of integer sequences* of Neil Sloane [28]. There, you can enter a sequence of integers and chances are good that the website will respond with one or more likely matches. However, the approach taken is quite different from ours: the encyclopedia keeps a list of currently roughly 160,000 sequences, entered more or less manually, and it compares the given sequence with each one of those. Besides that, there is an email service called `SuperSeeker` that tries some transformations on the given sequence to find a match in the database. Furthermore it tries some programs in the spirit of `Rate.m` and `GFUN` to find a formula, although with a time limit, i.e., it gives up when too much time has elapsed.

Thus, the two approaches complement each other: For example, there are sequences where no simple formula is likely to exist, and which can thus be found only in the encyclopedia. On the other hand, there are many sequences that have not yet found their way into the encyclopedia, but can be guessed in easily by your computer.

In Section 3 we outline the capabilities of our package. In Section 4 we describe the most important options that modify the behaviour of the functions. A very brief description of the algorithms used and the efficiency problems encountered is given in Section 5 and thereafter.

2. History

On the historical side, we remark that already in 1964, Malcolm Pivar and Mark Finkelstein [23] implemented a program to identify sequences given their first few terms, see also Paul W. Abrahams [1]. One interesting feature of their program was the ability to deal with exceptions to a rule: their program would apply, for example, the differencing operator, until most of the terms would be equal. In a second step, it would then locate the exceptions to the rule and try to guess formulae for the positions and for the values of the exceptions.

The first edition [27] of 'A Handbook of Integer Sequences' by Neil Sloane appeared in 1973. In 1992, François Bergeron and Simon Plouffe [5] explored the idea of applying various transformations to the given sequence, for example series reversion. They then used Padé approximation to see whether the result might be rational. In the same article an experimental program to check for 'constructible differentially finite' series is also briefly described, but it seems that it was not so successful.

In Physics, Richard Brak, G. S. Joyce, Michael E. Fisher, Helen Au-Yang, Anthony Guttmann [8, 13] developed methods using algebraic and hypergeometric or holonomic functions to fit series data starting from the early seventies. Of course, they named their techniques differently, and, more importantly, they were primarily interested in estimating 'critical exponents' and 'critical points' of the function whose first few Taylor coefficients are given.

3. Some Function Classes Suitable for Guessing

In this section we briefly present the function classes which are currently explicitly covered by our package. We want to stress however, that in many cases it is easy to add other function classes, should the need arise. (This will become clear in Section 5.)

Throughout this article, $n \mapsto f(n)$ is the function we would like to guess, and $f(x) = \sum_{n \geq 0} f(n)x^n$ is its generating function. The values $f(n)$ are supposed to be elements of some field $\mathbb{K}$, usually the field of rationals or rational functions. We alert the reader that the first value in the given sequence always corresponds to the value $f(0)$.

3.1. Guessing sequences $f(n)$.

guessRec: finds recurrences of the form

(6) $$p\big(f(n), f(n+1), \ldots, f(n+k)\big) = 0,$$

where p is a polynomial with coefficients in $\mathbb{K}[n]$. For example,

guessRec [1,1,0,1,- 1,2,- 1,5,- 4,29,- 13,854,- 685]

yields

$$[f(n) : -f(n+2) - f(n+1) + f(n)^2 = 0, f(0) = 1, f(1) = 1].$$

Note that, at least in the current implementation, we do not exclude solutions that do not determine the function f completely. For example, given a list containing only zeros and ones, one result will be

$$[f(n) : f(n)^2 - f(n) = 0, f(0) = \ldots].$$

guessPRec: only looks for recurrences with linear p, i.e., it recognises P-recursive sequences. As an example,

guessPRec [0, 1, 0, -1/6, 0, 1/120, 0, -1/5040, 0, 1/362880,
           0, -1/39916800, 0, 1/6227020800]

returns

$$[f(n) : (-n^2 - 3n - 2)f(n+2) - f(n) = 0, f(0) = 0, f(1) = 1].$$

guessRat: finds rational functions. For the sequence given in (1), we find n^2 as likely solution.

guessExpRat: finds rational functions with an Abelian term, i.e.,

$$f(n) = (a + bn)^n \frac{r(n)}{s(n)}$$

where r and s are polynomials.

guessExpRat [0,3,32,375,5184]

yields

$$n(n + 2)^n,$$

which could be interpreted, for example, as the number of labelled trees with one edge selected.

guessBinRat: finds rational functions with a binomial term, i.e.,

$$f(n) = \binom{a + bn}{n} \frac{r(n)}{s(n)}$$

where r and s are polynomials.

Concerning q-analogues, `guessRec(q)` finds recurrences of the form (6), where p is a polynomial with coefficients in $\mathbb{K}[q, q^n]$. Similarly, we provide q-analogues for `guessPRec` and `guessRat`. For example, to guess a formula for Sequence (4), we enter[2]

```
guessRat(q)([1,1+q+q^2,(1+q+q^2)*(1+q^2),(1+q^2)*(1+q+q^2+q^3+q^4)],
            [])
```

and obtain as function

$$\frac{q^3 q^{2n} + (-q^2 - q)q^n + 1}{q^3 - q^2 - q + 1}.$$

Unfortunately the simplifying capabilities of `FriCAS` are rather weak, so it takes some extra work to simplify the above expression to

$$\frac{(1 - q^{n+1})(1 - q^{n+2})}{(1 - q)(1 - q^2)},$$

i.e., the q-binomial coefficient $\begin{bmatrix} n+2 \\ 2 \end{bmatrix}_q := \frac{[n+2]_q[n+1]_q}{[2]_q[1]_q}$, where $[n]_q := \frac{1-q^n}{1-q} = 1 + q + \cdots + q^{n-1}$.

Moreover, it is also possible to guess 'mixed' recurrences, i.e., where p has coefficients in $\mathbb{K}[q, n, q^n]$, see the description of the option `maxMixedDegree` in Section 4. For example,

```
guessPRec(q)([1,1,2*q^2,6*q^6,24*q^12,120*q^20,720*q^30,5040*q^42],
              maxMixedDegree==2, homogeneous==true)
```

returns

$$[f(n) : (n + 1)f(n + 1)q^{2n} - f(n + 1) = 0, f(0) = 1].$$

The q-version of `guessExpRat` recognises functions of the form

$$f(n) = (a + bq^n)^n \frac{r(q^n)}{s(q^n)},$$

a and b being in $\mathbb{K}[q]$ and r and s polynomials with coefficients in $\mathbb{K}[q]$. For Sequence (5), we enter

```
guessExpRat(q)([(1-2*q)/(1-q),1-2*q,(1-q)*(1-2*q)^3,
                (1-q)^2*(1-2*q)*(1-2*q-2*q^2)^3], [])
```

to obtain

$$\frac{2q - 1}{q - 1}(2q^n - 3q + 1)^n.$$

Another example would be Nicholas Loehr's q-analogue $[n + 1]_q^{n-1}$ of Cayley's formula.

Finally, `guessBinRat(q)` tries to fit the given terms to

$$f(n) = \begin{bmatrix} a + bn \\ n \end{bmatrix}_q \frac{r(q^n)}{s(q^n)},$$

where $\begin{bmatrix} n \\ m \end{bmatrix}_q = \prod_{i=1}^m \frac{1-q^{n-i+1}}{1-q^i}$.

---

[2]Because of a flaw in `FriCAS`, one has to explicitly specify a list of options when using the q-versions of the guessing functions. In the example above, we simply gave an empty list of options, and did thus not override any of the default options.

3.2. **Guessing Series $f(x)$.**

guessADE: finds an algebraic differential equation for $f(x)$, i.e., an equation of the form

$$(7) \qquad p\Big(f(x), f'(x), \ldots, f^{(k)}(x)\Big) = 0,$$

where p is a polynomial with coefficients in $\mathbb{K}[x]$. A typical example is $\sum n^n \frac{x^n}{n!}$:

```
guessADE [1,1,2,9/2,32/3,625/24,324/5,117649/720,131072/315,
          4782969/4480]
```

returns

$$[[x^n]f(x): -xf'(x) + f(x)^3 - f(x)^2 = 0, f(0) = 1, f'(0) = 1].$$

Maybe more interesting, we obtain also a differential equation for the exponential generating function with coefficients of the form covered by guessExpRat:

```
guessADE([(a*n+b)^n/factorial(n) for n in 0..32],
         maxPower==3, maxDerivative==3, homogeneous==true)
```

However, this equation is already quite big:

$$4b^2(a+b)^2 f(x)^2 f''(x) + 3ab^2(a+b)xf(x)^2 f'''(x) - 4b^2(2a+b)^2 f(x)f'(x)^2$$
$$- a(a^3 + a^2 b + 19ab^2 + 3b^3)xf(x)f'(x)f''(x) - 3a^3(a-3b)f(x)f'(x)f'''(x)$$
$$+ 5a^3(a-3b)x^2 f(x)f''(x)^2 + 4a^4 xf'(x)^3 = 0.$$

We stress that we did not try to prove this equation – it remains a guess, even though we checked the first few hundred terms.

Another interesting example is given by the generating function for the chromatic polynomials of rooted triangulations, as found by William Tutte [31]. Or, as a test case, to guess a differential equation for Jacobi's θ-function $1 + 2\sum z^{n^2}$ a list of the first 3600 terms,

```
guessADE(l, maxPower==14, maxDerivative==3, maxDegree==6)
```

and a little patience (roughly ten minutes on an AMD Opteron processor) suffice. In fact, according to Don Zagier [34, Section 5.1, Proposition 15] already Ramanujan knew that every modular and every quasi-modular form on Γ_1 satisfies a third order algebraic differential equation.

guessHolo: only looks for equations of the form (7) with linear p, that is, it recognises holonomic or differentially-finite functions. It is well known that the class of holonomic functions coincides with the class of functions having P-recursive Taylor coefficients. However, the number of terms necessary to find the differential equation often differs greatly from the number of terms necessary to find the recurrence. Returning to the example given for guessPRec, we find that already

```
guessHolo [0,1,0,-1/6,0,1/120]
```

returns

$$[[x^n]f(x): -f''(x) - f(x) = 0, f(0) = 0, f'(0) = 1].$$

Moreover, now we immediately recognise the coefficients as being those of the sine function.

guessAlg: looks for an algebraic equation satisfied by $f(x)$, i.e., an equation of the form
$$p\left(f(x)\right) = 0,$$
the prime example being given by the Catalan numbers

`guessAlg [1,1,2,5,14,42]`

which yields
$$[[x^n]f(x) : xf(x)^2 - f(x) + 1 = 0, f(0) = 1].$$

guessPade: recognises rational generating functions or, equivalently, recurrences with constant coefficients. For the Fibonacci sequence given in (2), we find as likely solution
$$[[x^n]f(x) : (x^2 + x - 1)f(x) + 1 = 0].$$

guessFE: finds 'Mahler-type' functional equations for $f(x)$ (see for example [21]), i.e., equations of the form
$$(8) \qquad p\left(f(x), f(x^2), \ldots, f(x^k)\right) = 0,$$
where p is a polynomial with coefficients in $\mathbb{K}[x]$. A typical example is the number of unlabelled rooted binary trees:

`guessFE [0,1,1,1,2,3,6,11,23]`

which returns
$$[[x^n]f(x) : f(x^2) + f(x)^2 - 2f(x) + 2x = 0, f(x) = x + x^2 + x^3 + 2x^4 + O(x^5)].$$

Browsing *the online encyclopedia of integer sequences*, we discovered another rather surprising functional equation: consider the sequence `A118006` of binary words w_n defined by $w_1 = "01"$ and $w_{n+1} = concat[w_n, w_n, reverse(w_n)]$. Then

`guessFE w 4`

indicates that the limiting word w_∞ satisfies
$$(x - 1)(x^2 - x + 1)(x^2 + x + 1)((x^2 + x + 1)\left(f(x) - x^2 f(x^3)\right) + x(x^2 + 1)^2 = 0.$$

Again, we did not try to prove this equation but only checked the first few hundred terms.

For `guessADE` and `guessHolo` we provide q-analogues, replacing differentiation with q-dilation: `guessADE(q)` finds differential equations of the form
$$(9) \qquad p\left(f(x), f(qx), \ldots, f(q^k x)\right) = 0,$$
where p is a polynomial with coefficients in $\mathbb{K}[q, x]$. Generating functions satisfying such q-equations frequently occur in the enumeration of polyominoes and the study of orthogonal polynomials. As an example, we can recover the q-algebraic differential equation for the generating function of bar polyominoes by horizontal perimeter – marked by x, vertical perimeter – marked by y and area – marked by q, as given by Richard Brak and Thomas Prellberg [24]. We enter

`guessADE(q)(l, maxDerivative==1, maxPower==2, maxDegree==1)`

where `l` are the first eleven coefficients of the series in x:

`l := [0, q*y/(1-q*y), q^2*y*(1+q*y)/(1-q*y)/(1-q^2*y), ...]`

The solver immediately finds the solution

$$[x^n]f(x) : (qxf(x) + (qx + 1)y)f(qx) + (qx - 1)f(x) + qxy = 0,$$
$$f(0) = 0, f'(0) = \frac{qy}{1 - qy}, \dots],$$

it then takes a few seconds to verify it.

3.3. **Operators.** The observation made by Christian Krattenthaler before writing his program `Rate.m` [20] is the following: it occurs frequently that although a sequence of numbers is not generated by a rational function, the sequence of successive quotients is.

We slightly extend upon this idea, and apply recursively one or both of the two following operators:

> `guessSum` - Δ_n: the differencing operator, transforming $f(n)$ into $f(n) - f(n-1)$.
>
> `guessProduct` - Q_n: the operator that transforms $f(n)$ into $f(n)/f(n-1)$.

For example, to guess a formula for Sequence (3), we enter

`guess([0, 1, 3, 9, 33], [guessRat], [guessSum, guessProduct]).`

The second argument to `guess` indicates which of the functions of the previous section to apply to each of the generated sequence, while the third argument indicates which operators to use to generate new sequences.

The package will then respond with

$$\sum_{s=0}^{n-1} \prod_{p=0}^{s-1} (p + 2),$$

i.e., the sum of the first factorials.

In the case where only the operator Q_n is applied, our package is directly comparable to `Rate.m`. In this case the standard example is the number of alternating sign matrices

`guess [1, 1, 2, 7, 42, 429, 7436, 218348]`

which yields

$$\prod_{k=0}^{n-1} \prod_{l=0}^{k-1} \frac{27l^2 + 54l + 24}{16l^2 + 32l + 12} = \prod_{k=0}^{n-1} \prod_{l=0}^{k-1} \frac{3(3l + 2)(3l + 4)}{4(2l + 1)(2l + 3)}.$$

3.4. **Closure properties and zero test.** Part of what makes a class of functions interesting are its closure properties, summarised in the table below for some classes of functions. Apart from the theoretical point of view, it is also good to know that the computer can guess an equation for $f(n)$ if it can do so for $f(n) + 1$.

However, one has to keep in mind that even simple transformations may increase the number of terms necessary to successfully guess an equation dramatically. For example, consider the (exponential) generating function for the Bell numbers B_n, counting the number of partitions of $\{1, 2, \dots, n\}$, which is

$$B(x) = \sum_{n \geq 0} B_n \frac{x^n}{n!} = e^{e^x - 1}.$$

This series is not holonomic, but it satisfies the simple algebraic differential equation $B''B - (B')^2 - B'B = 0$, and the first thirteen terms suffice to find it. By contrast, it takes 36 terms to guess the shifted series $(e^{e^x-1} - 1)/x$.

In the same spirit, note that without specifying the search space any further, already six terms are enough to guess a functional equation for the number of unlabelled rooted binary trees. On the other hand, we need at least 42 terms to guess an equation for the square of their generating function.

This phenomenon also explains why Christian Krattenthaler's program `Rate.m` is so useful: of course there is also an algebraic recurrence for the number of alternating sign matrices, namely

$$(-16n^2 - 32n - 12)f(n)f(n+2) + (27n^2 + 54n + 24)f(n+1)^2 = 0,$$

but we need 35 terms to guess it instead of eight. (Instead of looking for a formula having k nested products, we could also use the options `Somos==true, maxShift==k, homogeneous==2^(k-1)`, but this only works well for k less than 4.)

type of equation	$+$	$\cdot$	$(.)^{-1}$	$\circ$	$(.)^{(-1)}$	D	$\int$	$\odot$	S
Padé	✓	✓	✓	✓	-	✓	-	✓	✓
algebraic	✓	✓	✓	✓	✓	✓	-	-	✓
linear differential	✓	✓	-	alg.	-	✓	✓	✓	✓
algebraic differential	✓	✓	✓	✓	✓	✓	✓	-	✓

TABLE 1. closure properties. $((.)^{-1}$: multiplicative inverse, $\circ$: composition, $(.)^{(-1)}$: compositional inverse, $\int$: definite integration, $\odot$: Hadamard product, S: shift, alg.: algebraic substitution; inverse and substitution only apply when the result is again a formal power series.

Proofs for the closure properties of rational, algebraic and linear differential equations can be found, for example, in Richard Stanley's book Enumerative Combinatorics 2 [30] or his article on differentiably finite series [29]. For algebraic differential equations, proofs were given by Alexander Ostrowski in [22], see also [19]. Slightly weaker closure properties hold for the q case. In particular, q-holonomic series are only closed under the substitution $x \mapsto x^k, k \in \mathbb{N}$, see for example [16].

Algebraic recurrence relations seem to satisfy no interesting closure properties: for example, take any sequence that does not satisfy an algebraic recurrence relation, and write it as the sum of two sequences, one with odd terms zero, the other with even terms zero. Both summands are solutions of $f(n)f(n+1) = 0$. However, a related class, so called 'admissible recurrences' was studied by Manuel Kauers [14] and has been shown to enjoy many closure properties.

Similarly, we are not aware of any results concerning closure properties of Mahler-type functional equations as defined in our paper. However, linear equations $p\left(f(x), f(x^r), f(x^{r^2}) \ldots, f(x^{r^k})\right) = 0$ for fixed r were shown by Phillippe Dumas [11] to give rise to nice closure properties. This extends to the non-linear case.

One very important property of these classes is the availability of a *zero-test*, i.e., an algorithm that will decide whether any given equation (together with sufficiently many initial values) has only the zero solution. For linear differential equations

this is folklore, for algebraic differential equations an algorithm was proposed for example by Joris van der Hoeven [32]. In many cases, such a test allows to verify conjectured identities automatically, as exercised for example by GFUN.

4. Options

To give you the maximum flexibility in guessing a formula for your favourite sequence, we provide options that modify the behaviour of the functions as described in Section 3. The options are appended, separated by commas, to the guessing function in the form `option==value`. See below for some examples.

maxDerivative, maxShift: specify the maximum derivative in an algebraic differential equation, or, in a recurrence relation, the maximum shift. Setting the option to `arbitrary` specifies that the maximum derivative – the maximum shift – may be arbitrary, which is the default.

maxPower: specifies the maximum total degree in an algebraic differential equation or recurrence: for example, the degree of $(f'')^3 f'$ is 4. Setting the option to `arbitrary` specifies that the maximum total degree may be arbitrary, which is the default.

homogeneous: specifies whether the search space should be restricted to homogeneous algebraic differential equations or homogeneous recurrences, i.e., the case where the polynomial p in Equation (6) and Equation (7) is homogeneous. By default, it is set to `false`. Setting it to a positive integer, only homogeneous polynomials p of this degree are tried. Setting it to `true`, all homogeneous polynomials p up to total degree `maxPower` are tried.

Somos: specifies whether the search space should be restricted to algebraic differential equations where the sum of differentials is constant. Similarly, when guessing recurrences, Somos insists that the sum of shifts is constant. By default it is set to `false`. Setting it to a positive integer, the sum of differentials or shifts must be equal to this number. Setting it to `true` is equivalent to invoking the guesser with `Somos==2`, `Somos==3`, ..., `Somos==d`, where `d` is the specified `maxDerivative` (or `maxShift`) times `maxPower` or `homogeneous`.

maxDegree: specifies the maximum degree of the coefficient polynomials in an algebraic differential equation, a Mahler-type functional equation or a recurrence with polynomial coefficients. For rational functions with an exponential term, `maxDegree` bounds the degree of the denominator polynomial. The default value of `maxDegree` is `arbitrary`.

allDegrees: specifies whether all possibilities of the degree vector – taking into account `maxDegree` – should be tried. The default is `true` for `guessPade` and `guessRat` and `false` for all other functions.

maxMixedDegree: allows guessing of mixed q-recurrences. Its value determines the maximum degree of q^n in the coefficients, default being zero.

maxLevel: specifies how many levels of recursion are tried when applying operators as described in Section 3.3. Note that, applying either of the two operators results in a sequence which is by one shorter than the original sequence. Therefore, in case both `guessSum` and `guessProduct` are specified, the number of times a guessing algorithm from the given list of functions

is applied is roughly 2^n, where n is the number of terms in the given sequence. Thus, especially when the list of terms is long, it is important to set `maxLevel` to a low value.

Still, the default value is `arbitrary`, which means that the number of levels is only restricted by the number of terms given in the sequence.

`safety:` specifies, as explained in detail in Section 5 and Section 6 the number of additional equations a solution has to satisfy. The default setting is 1.

Experiments indicate that, the larger the class of functions covered, the larger one should set `safety`. Moreover, when the sequence contains many zeros, higher settings of safety are appropriate. For all algorithms we recommend to set `safety` higher than the number of trailing zeros. The reason is best illustrated by an example:

`guessPade([a,b,c,0])`

returns

$$[[x^n]cx^2 + bx + a].$$

In other words, if the sequence has a trailing zero, `guessPade` trivially finds a solution. A few experiments and a moment's thought will reveal that the other algorithms behave similarly.

`check:` determines whether we want to check the solutions returned by the modular solver using a deterministic check, or whether we content ourselves with a (rather weak) Monte-Carlo type check, or skip checking entirely, the default value being `deterministic`.

`checkExtraValues:` specifies whether we want to return only those solutions that fit the given data perfectly. With `checkExtraValues==false`, the complete basis of the solution space is returned, see Section 6. The default value is `true`.

`one:` specifies whether the guessing function should return as soon as at least one solution is found. By default, this option is set to `true`.

`indexName, variableName, functionName:` specify symbols to be used for the output. The defaults are `n`, `x` and `f` respectively.

`debug:` specifies whether information about progress should be reported.

5. Rational Interpolation

The underlying idea of all guessing software is to fit the given data to a *model*. For example, a formula for Sequence (1), is almost trivial to guess: it seems obvious that it is n^2. A natural model to check is that the sequence in question is generated by a polynomial – we simply apply polynomial interpolation. Given a list of four terms – $0, 1, 4, 9$ in our example – we should expect that we need a polynomial of degree three to interpolate. Since the actual degree is lower, that is, the interpolating polynomial is overdetermined by the data, it is reasonable to accept n^2 as a good guess.

Generalising to *Hermite-Padé interpolation*, we can cover most models described in Section 3.1:

Rational Interpolation Problem, Sequence Variant. *Let* $\mathbf{f} = [f^{(1)}(n), \ldots, f^{(m)}(n)]$ *be a vector of (truncated) sequences over some integral domain, and* $\mathbf{n} = [n^{(1)}, \ldots, n^{(m)}]$ *a vector of non-negative integers, serving as degree bounds. Let* $\sigma \geq 0$. *Determine*

a polynomial vector $\mathbf{p} = [p^{(1)}(n), \ldots, p^{(m)}(n)]$ *with* $\deg p^{(l)}(n) < n^{(l)}$ *such that*

$$(10) \qquad p^{(1)}(n) \cdot f^{(1)}(n) + \cdots + p^{(m)}(n) \cdot f^{(m)}(n) = 0 \text{ for } 0 \leq n < \sigma.$$

Note that, by equating coefficients, this problems can be reduced to solving an appropriate *linear* system of equations with $n^{(1)} + \cdots + n^{(m)} - 1$ unknowns, namely the coefficients of the polynomials $p^{(1)}(n), \ldots, p^{(m)}(n)$, up to normalisation. Thus, we will in fact determine a basis of the space of solutions. However, instead of using, for example, naive Gaussian elimination, we will take advantage of the special structure of these linear systems to achieve better performance. To illustrate, we would like to be able to solve systems where $n^{(1)} + \cdots + n^{(m)}$ is as large as ten thousand.

Setting $\sigma = n^{(1)} + n^{(2)} - 1$ and $\mathbf{f} = [(1, 1, \ldots, 1), (f_0, f_1, \ldots, f_{\sigma-1})]$ we would recover ordinary rational interpolation. However, to have more confidence in the 'guessed' formula, we use $\sigma = n^{(1)} + \cdots + n^{(m)} - 1 + \texttt{safety}$ instead.

More generally, to guess algebraic recurrences we consider the (infinite) sequence of monomials in the 'variables' $f(n), f(n+1), f(n+2), \ldots$

$$\Big(\prod_i f(n + \lambda_i - 1) \Big)_\lambda,$$

where $\lambda = (\lambda_1, \lambda_2, \ldots)$ runs over the integer partitions in lexicographic order:

$$1, f(n), f(n)^2, f(n+1), f(n)^3, f(n)f(n+1), f(n+2), f(n)^4, f(n)^2 f(n+1), \ldots$$

Then, for each $m \geq 2$ we solve the rational interpolation problem with $\mathbf{f}$ given by the first m entries of this sequence, and $\mathbf{n}$ such that the number of unknowns $n^{(1)} + \cdots + n^{(m)} - 1$ in the corresponding linear system plus the specified value of $\texttt{safety}$ equals the number of equations σ.

In the formulation of the rational interpolation problem above, the sequence of evaluation points was chosen as $0, 1, 2, \ldots$, but it is straightforward to generalise to arbitrary evaluation points. Doing so, we can also find q-recurrences, by pretending that f is given at the points $q^0, q^1, q^2, \ldots$ instead.

To deal with the models described in Section 3.2, we need to solve another variant of the rational interpolation problem:

Rational Interpolation Problem, Series Variant. *Let* $\mathbf{f} = [f^{(1)}(x), \ldots, f^{(m)}(x)]$ *be a vector of (truncated) power series over some integral domain, and* $\mathbf{n} = [n^{(1)}, \ldots, n^{(m)}]$ *a vector of non-negative integers, serving as degree bounds. Let* $\sigma \geq 0$. *Determine a polynomial vector* $\mathbf{p} = [p^{(1)}(x), \ldots, p^{(m)}(x)]$ *with* $\deg p^{(l)}(x) < n^{(l)}$ *such that*

$$(11) \qquad \operatorname{ord}(\mathbf{p} \cdot \mathbf{f}) = \operatorname{ord}\Big(p^{(1)}(x) \cdot f^{(1)}(x) + \cdots + p^{(m)}(x) \cdot f^{(m)}(x) \Big) \geq \sigma.$$

In this case, setting $\sigma = n^{(1)} + n^{(2)} - 1$ and $\mathbf{f} = [1, f(x)]$, where $f(x)$ is the truncated power series with the given values as Taylor coefficients, we recover Padé approximation. This allows us to 'guess' sequences that are Taylor coefficients of rational generating functions.

To guess algebraic differential equations, we consider the sequence of monomials $\Big(\prod_i D^{\lambda_i - 1} f(x) \Big)_\lambda$, where D is the differentiation operator and $\lambda = (\lambda_1, \lambda_2, \ldots)$ runs over the integer partitions in lexicographic order as before:

$$1, f(x), f(x)^2, f'(x), f(x)^3, f(x)f'(x), f''(x), f(x)^4, f(x)^2 f'(x), \ldots$$

To guess q-algebraic differential equations, we just replace the usual differentiation operator with q-dilation: $\mathrm{D}\,f(x) := f(qx)$. Finally, `guessFE` uses the sequence of monomials $\left(\prod_i f(x^{\lambda_i})\right)_\lambda$.

For the present package, we originally implemented a fraction free algorithm proposed in 2000 by Bernhard Beckermann and George Labahn [4], which at the time proved much faster than what `GFUN` had. However, during the refereeing process it became clear that a modular approach would be even more efficient. This was first pointed out by Manuel Kauers, and independently by Alin Bostan and Bruno Salvy in private communication. Consequently, we decided to follow this approach and implemented a modular version of an *older* algorithm from 1994, also by Bernhard Beckermann and George Labahn [2], when the coefficients are rational numbers or rational functions with integer coefficients. This turned out to be very fruitful, although quite labour-some. For other coefficient domains we still use the fraction free algorithm, although we plan to extend the modular approach to allow algebraic numbers as coefficients as soon as possible.

We would like to stress that meanwhile most of the packages mentioned in the introduction use modular techniques, however using other algorithms for solving over a prime field. According to Bruno Salvy, `GFUN` now uses an algorithm introduced in 1997, again by Bernhard Beckermann and George Labahn [3]. Manuel Kauers package `Guess.m` uses the solver provided by `Mathematica`, it is thus unclear which algorithm is used.

Still, our package outperforms the other freely available packages, for many configurations of degree bounds and size of the vector $\mathbf{f}$, (see Section 10), as well as – for univariate sequences – the range of formulae that can be guessed.

We also implemented specialised algorithms to test whether the nth term of the sequence is given by a formula of the form

$$(12) \qquad n \mapsto (a + bn)^n \frac{r(n)}{s(n)} \quad \text{or} \quad n \mapsto \binom{a + bn}{n} \frac{r(n)}{s(n)},$$

for some a and b and polynomials r and s. Unfortunately, we could not avoid solving *non-linear* equations in this case. Even after exploiting some surprising coincidences that reduce the size of the arising equations the performance of this algorithm is disappointing: already eight or nine terms, i.e., degree two in r and s pose a challenge, even over a finite field.[3]

6. Safety

How can we 'know' that a formula discovered via interpolation is appropriate? At first glance, the answer is quite simple: we use all but the last few terms of the sequence to derive the formula. After this, the last terms are compared with the values predicted by the polynomial. If they coincide, we can be confident that the guessed formula is correct.

In the case of the rational interpolation problem we get the same set of accepted solutions when we use all values, but keep lower degree bounds. We use this approach as it is more efficient than actually computing 'bad' solutions and rejecting them later, although there is a subtle interaction with an extra check that we perform.

---

[3]Meanwhile, it seems that we have found a suitable approach, but due to time constraints we cannot describe it in this article.

Very recently, Alin Bostan and Manuel Kauers [6, Section 2.4] described in some detail various other possibilities of checking whether a guessed formula is likely to be 'correct', the method we just outlined being clearly the most practical. Unfortunately, it turns out that this method is problematic in certain situations. In this section we explain why.

First of all, we cannot expect that all elements of the solution space of the rational interpolation problem 'interpolate' the given data in the following sense: consider the truncated power series $f(x) = 1 + x^6 + O(x^7)$, and let $\mathbf{f} = [1, f(x), f'(x)]$. Setting the vector of degree bounds $\mathbf{n} = [2, 2, 2]$ and $\sigma = 6$ (note that we 'loose' one term because of differentiation, so we have 6 equations in our linear system), rational interpolation yields the basis $[(1, -1, 0), (0, 0, x)]$. Thus, the general solution to the rational interpolation problem with the given constraints is

$$(\alpha + \beta x)(1 - f(x)) + \gamma x f'(x) = 0,$$

α, β and γ being elements from the coefficient field.

Apparently, none of the two basis vectors actually interpolates all given values: $1 - f(x) = -x^6 + O(x^7)$, and $x f'(x) = 6x^6 + O(x^7)$. One might be tempted to simply discard non-interpolating basis vectors (which we do when `checkExtraValues` is true), but doing so we risk loosing 'good' solutions, too:

$$(6\gamma + \beta x)(1 - f(x)) + \gamma x f'(x) = O(x^7)$$

interpolates just fine for any β and γ. In particular, the set of interpolating solutions is not a vector space.

An uncomfortable consequence of the above is as follows: we provide an option `maxDegree` that allows the user to specify the maximum degree of the coefficient polynomials, see Section 4. When set to some integer value d, we (essentially) do not compute solution spaces of configurations $\mathbf{f}$ with $(d+1)|\mathbf{f}|$ being less than the number of values provided. Suppose now that we find an interpolating solution without setting `maxDegree`, and that the maximal coefficient degree of this solution happens to be d. Then it may be the case that setting `maxDegree==d` instead yields no result, because all basis elements are discarded. Similarly, one might expect that increasing both `safety` and the number of values by one does not yield more solutions. But at lower `safety` our check may reject all basis elements, while at higher `safety` the basis may contain an interpolating solution.

A possible way to resolve this dilemma might be to reject solution spaces that are not one-dimensional. However, when pursuing this idea, another difficulty surfaces: namely, it is not completely trivial to decide whether two solutions are *really* different. For example, consider $\mathbf{f} = [1, f(x), f'(x)]$, and suppose that $f(x)$ is in fact a polynomial $p(x)$. Then the interpolation routine will not only find the solution $[p(x), 1, 0]$, but also $[p'(x), 0, 1]$. More generally, it is well known that one often needs more coefficients to determine the minimal order equation than to find a solution of higher order. Thus, if we have enough values to guess the minimal order equation then the problem is easy. But otherwise we will either find multiples of the minimal equation, or some parasitic solutions.

This problem can be remedied, at least for linear and also algebraic differential equations: in the linear case, we could simply compute a greatest common right divisor of the given equations, whereas in the algebraic case we could apply Ritt elimination.

Still, there is again some danger that 'good' solutions are lost: for example, if a sequence is non-zero only at very few indices $n_1, n_2, \ldots, n_k$, then the interpolation algorithm will not only find the 'good' solution, but also $(n - n_1)(n - n_2) \ldots (n - n_k)f(n) = 0$, and the greatest common right divisor of the two will be trivial.

We admit that so far we were unable to find a completely satisfying solution to this problem. In the meantime, we provide options (in particular `checkExtraValues`, and `one`, see Section 4) that let the user decide.

7. Rational Reconstruction

As already mentioned in Section 5, our solver uses a modular technique: instead of solving the rational interpolation problem over the integers, we solve the problem over several machine size prime fields and use Chinese remaindering to obtain integer solutions. In the same spirit, given coefficients that are rational functions, we evaluate them at several random points, solve the simpler problems and use rational reconstruction to obtain polynomial solutions.

There are two different ways in which this plan can fail: it may happen that the solution of the problem in the prime field is not an image of the solution of the problem in the original ring. In Corollary 9.6 we will see that there are ways to discover such 'bad reductions', provided we have at least one 'good reduction'. However, we cannot a priori exclude the possibility that all reductions are bad.

Moreover, it may occur that due to an unfortunate choice of evaluation points we obtain wrong solutions – usually, when we have too few evaluation points we get no solution, but it may happen that we construct one that is actually wrong.

Therefore, the solution returned from the core solver is only probably correct, and we need to check it before returning it. Thus, the main loop of the solver in pseudocode is:

```
repeat
    sol := do_solve(data, inner_call? == false)
    if check(sol) then return sol
```

where `do_solve` produces a probably correct solution and `check` verifies correctness. So far we did not encounter a case where the check failed – the solver is designed in such a way that probability of wrong answer is very low. Therefore, instead of looping, we print an error message and fail.

`do_solve` is a Brown-style [9] routine similar to Subroutine M and Subroutine P in the gcd algorithm of Mark van Hoeij and Michael Monagan [33], which in pseudocode looks as follows:

```
do_solve(data) ==
    if R = Z_p then return solve_over_Z_p(data)
    bad_count := 0
    good_count := 0
    sol := empty()
    repeat
        modulus := choose_modulus()
        if inner_call? then
            new_data := eval(data, modulus)
        else
            new_data := reduce(data, modulus)
        new_sol := do_solve(new_data, inner_call? == true)
```

```
if new_sol = "no_solution" then return "no_solution"
reduction_status := check_reduction(new_sol)
if new_sol = "failed" or reduction_status = "bad" then
    bad_count := bad_count + 1
    if inner_call? and bad_count > good_count + 2 then
        return "failed"
else
    good_count := good_count + 1
    if reduction_status = "all_bad" then sol := empty()
    sol := chinese_remainder(sol, new_sol, modulus)
    rr := rational_reconstruction(sol)
    if not rr = "failed" then return rr
```

In contrast to Mark van Hoeij and Michael Monagan we present this algorithm as a single routine, to stress that the processing is generic: in the outer level, when `inner_call?` is false, `choose_modulus` chooses machine size primes, in the inner level it chooses random evaluation points.

The routine `check_reduction` applies Corollary 9.6 to `new_sol` – we keep the necessary information about previously obtained solutions in a global variable, namely the minimal dimension of the solution space and the minimal values of the critical indices. (Of course, when there are no previous solutions, `new_sol` is automatically treated as 'good'.) Also, we discard solutions with leading exponent being smaller than the leading exponent of previous solutions – this is necessary to ensure correct normalisation after rational reconstruction and also avoids the problem of 'bad content', see [33].

The counters `bad_count` and `good_count` are used to detect cases where we encounter bad reduction already at the outer level – we copied the method used in [33]. Without this test the solver would spend a lot of time reconstructing solutions which would then be discarded at outer level.

The variable `sol` contains homomorphic images of the bases of the solution spaces constructed in the current stage, and `rational_reconstruction(sol)` tries to find the basis in the original ring. In the following, we indicate which tricks we decided to implement to make the procedure efficient enough. Namely, for polynomials we use naive quadratic multiplication, gcd and Chinese remaindering (Lagrange interpolation) routines. Also our rational reconstruction implementation uses a simple quadratic method. However, we save time by trying rational reconstruction not in every step but only after an interval: for polynomials we use a quadratically growing sequence of points, while for integers we switch to a big step (currently 100) after passing a threshold (currently 200).

More precisely, to reconstruct a solution in characteristic zero given modular images $m_1, m_2, \ldots, m_n$ and moduli $p_1, p_2, \ldots, p_n$ we need to compute M such that $M = m_i \mod p_i$, and then apply rational reconstruction to M. However, instead of computing M directly we first compute intermediate solutions M_j such that $M_j = m_i \mod p_i$ for $i \in \{100j, \ldots, 100j + 99\}$, updating incrementally. Whenever we have finished such a block of 100 primes and computed M_j, we update M such that $M = M_j \mod P_j$ where $P_j = p_{100j}p_{100j+1} \ldots p_{100j+99}$, and then apply rational reconstruction to M. This scheme makes more efficient use of bignum routines: we

perform most operations on relatively small bignums, and fewer operations on big bignums.

There is one more improvement to rational reconstruction which first appeared in NTL [26] (see also Section 3.1 of the description of the IML [10], which is where we learned from the trick): when reconstructing the vector of rational coefficients, we incrementally compute the common denominator of the coefficients already reconstructed, and impose it on subsequent terms. Since we are looking for an integer (or polynomial) solution and fractions appear only due to normalisation, it is natural to expect all terms to have the same or very similar denominators. Often, multiplying the next term by the common denominator computed so far, it turns out that the product is already acceptable as rational reconstruction of this coefficient.

8. Implementation aspects

In this section we give some details of our implementation. More precisely, we explain some choices we made when implementing the subroutines `solve_over_Z_p`, `eval_or_reduce` and `check` mentioned in the algorithm in Section 7.[4]

8.1. **solving over $\mathbb{Z}_p$.** As already mentioned, our solver over $\mathbb{Z}_p$ closely follows [2]. It returns a matrix of polynomials, every column constitutes one solution of the Hermite-Padé interpolation problem. Solutions, and also the residuals that occur within the algorithm, are packed in vectors of machine size integers. This is possible, since we perform the same operations on each component polynomial of the solution. However, it lowers control and memory management overhead.

Computations are performed in place – otherwise memory management would dominate the run time. As basic operation we use 'multiply and add', that is we compute $v_1 + cv_2$ for two vectors v_1 and v_2 and a scalar c, and assign the result to v_1. Compared to separate addition and multiplication with a scalar, this approach halves the cost of remainder computations needed for modular arithmetic and also halves the loop overhead.

Currently the 'multiply and add routine' is written in Spad (`FriCAS`' high level implementation language) and via Common Lisp compiled to machine code.[5] On 64-bit machines 32-bit times 32-bit multiplication and 64-bit by 32-bit remainder are compiled directly to machine instructions. On a 64-bit Core 2 this leads to about 20 clocks per multiply and add step – it seems that this is the same as the cost of the machine instruction to compute a remainder. On 32-bit machines our compilation scheme performs operations involving 64-bit numbers (either as a result or as an argument) via calls to bignum routines, which causes much higher execution times.

In principle we could speed up the solver over $\mathbb{Z}_p$ replacing remainder operations by multiplications. Moreover, the 'multiply and add' routine is quite small so it would make sense to replace it by an assembler routine. We estimate that this would make our inner loop run 5-10 times faster. However, after some initial work our solver over $\mathbb{Z}_p$ turned out to be the fastest part of the package, so we concentrated on removing bottlenecks in other parts. Also, despite the quadratic complexity

---

[4]The actual code can be found in the files `modhpsol.spad.pamphlet` and `mantepse.spad.pamphlet` in the `FriCAS` distribution.

[5]`FriCAS` can compile code using a variety of Common Lisp implementations. Currently, best performance is obtained with `SBCL`, `http://www.sbcl.org`, thus in the following we assume this implementation.

and the non-optimal inner loop our solver seems to compare favourably with other programs, like `GFUN`, `Guess.m` and `qGeneratingFunctions.m`.

8.2. **computing modular images.** Initially we used a very naive evaluation scheme to obtain the vector of truncated power series (or sequences) over $\mathbb{Z}_p$: we did the computation in characteristic 0 and then computed remainders of division by p. Measurements showed that this approach used more than 98% of the execution time. Therefore, we switched to a faster scheme.

For polynomials with integer coefficients we use a specialised routine to reduce coefficients modulo p. Also, for univariate polynomials with coefficients in $\mathbb{Z}_p$ we use a specialised evaluation routine. However, for multivariate polynomials with modular coefficients we still use the naive method: we substitute for the variables in characteristic 0 and then use the routine just mentioned to reduce coefficients modulo p. Of course this needlessly uses bignum arithmetic, but since multivariate evaluation is typically followed by several univariate evaluations the cost seem to be acceptable.

Besides making the evaluation routines fast enough, it is important to generate the vector $\mathbf{f}$ of sequences or truncated power series efficiently. Initially we computed $\mathbf{f}$ in characteristic 0 and then performed modular reduction. It turned out that doing the modular reduction only on the original sequence and computing the derived sequences only over Z_p is much faster.

Moreover, we remark that for truncated power series computing $\mathbf{f}$ involves computing many Cauchy products, which can be expensive. Therefore we implemented a simple optimiser, that takes a vector of monomials and detects common factors that can be cached. This reduced the number of Cauchy products that have to be computed significantly.

With these improvements, the time needed for computing the modular images typically is comparable to the time needed for solving over Z_p.

8.3. **checking solutions.** For large problems, checking the solutions may be dominant factor. For sequences operations are performed pointwise, but for truncated power series we need polynomial multiplication which is much more expensive – the current polynomial routines of `FriCAS` are quadratic. Also, memory use may be a problem: we can solve the Hermite-Padé problem without actually computing the series or sequences forming it (we only compute modular images), but we need the actual system to check the solutions. For example, we can guess an equation for the generating function of Gessel walks (see the article by Manuel Kauers, Christoph Koutschan, and Doron Zeilberger about its holonomicity [18] and the article by Alin Bostan and Manuel Kauers about its algebraicity [7]) using a few hundred megabytes of memory, but explicitly storing the Hermite-Padé problem needs several gigabytes and we run out of memory on an 8 GB machine.

We therefore introduced options, see Section 4, that allow skipping the checks or to use a Monte-Carlo check. Although on small problems the Monte-Carlo check may be slower than the deterministic check, for the problem of Gessel walks it only takes about 30 sec. on a 2.4 GHz Core 2 and has moderate memory usage.

There is an additional problem: our initial data are fractions. `FriCAS` fraction arithmetic simplifies fractions after each operation, which is quite costly (but also avoids intermediate expression swell which would be even more costly). In practise in many cases denominator is 1, and using fractions causes almost no extra cost.

But some problems really make use of fractional coefficients and for such systems checking is more costly. For linear recurrences we implemented a special purpose checking routine which computes the common denominator using smaller number of gcd operations and performs the rest of operations in fraction-free way. This routine also avoids storing the vector $\mathbf{f}$ of the Hermite-Padé problem and thus avoids problems with excessive memory usage. Unfortunately, for other problems checking is much more complicated so currently for them we use general purpose checking routine which uses standard fraction arithmetic.

We also considered using a modular method for checking. Such a check would avoid problems with memory use. However, our a priori bounds on the number of evaluation points needed are so large, that the modular check is likely to be slower then our current version. Thus we only implemented a Monte-Carlo version of a modular check (currently only for the case of truncated power series).

9. Normalisation

The output of the algorithm proposed by Bernhard Beckermann and George Labahn in 1994 [2] is a so called σ-basis (also known as *order basis*) for the solution space of the rational interpolation problem.

In this section, we will call a polynomial vector a *solution* of the rational interpolation problem if it satisfies the order condition (10) or (11). The degree constraints will be taken into account in a different manner, namely through the notion of *defect*:

Definition 9.1. Let $\mathbf{n} = [n^{(1)}, \ldots, n^{(m)}]$ a vector of degree bounds and $\mathbf{p} = [p^{(1)}(x), \ldots, p^{(m)}(x)]$ be a vector of polynomials.

Then the *defect* of $\mathbf{p}$ is the (possibly negative) difference between degree bound and degree of $\mathbf{p}$. More precisely:

$$\operatorname{defect} \mathbf{p} = \min \left\{ n^{(i)} - \deg p^{(i)}(x) : i \in \{1, \ldots, m\} \right\}.$$

For any $\delta \in \mathbb{Z}$, we denote by $\mathcal{L}_\delta^\sigma$ the space of solutions with defect strictly larger than $-\delta$.

A σ-*basis* $\{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$ is a set of solutions of the rational interpolation problem, such that every solution $\mathbf{q} = [q^{(1)}(x), \ldots, q^{(m)}(x)]$ can be written as a linear combination $\mathbf{q} = \sum_{r=1}^m \alpha_r(x)\mathbf{p}_r$, with defect $\mathbf{q} \leq \operatorname{defect} \mathbf{p}_r - \deg \alpha_r(x)$, in a unique way. Note that the elements of the σ-basis do not need to satisfy the degree bounds, i.e., their defect can be negative.

An alternative description of σ-bases is given as follows, see [2, Equation (6) and (7)]:

(13) $\mathcal{L}_\delta^\sigma = \operatorname{span}\{x^j \mathbf{p}_r : 1 \leq r \leq m, j < \operatorname{defect} \mathbf{p}_r + \delta\},$

and

(14) $\dim \mathcal{L}_\delta^\sigma = \sum_{r=1}^m \max(\operatorname{defect} \mathbf{p}_r - d, 0).$

It follows that the set of defects is an invariant of the rational interpolation problem.

We want to reconstruct a σ-basis over an integral domain R (usually the integers or polynomials with integer coefficients) from several σ-bases over quotient rings R/I, where I is a prime ideal in R. For this purpose it is crucial to know when

a σ-basis over the field of fractions $Frac(R/I)$ of R/I is an image of the σ-basis over R. The difficulty is that for a given Hermite-Padé problem, the σ-basis is not uniquely determined. Fortunately, over a field we may obtain uniqueness using an appropriate normalisation.

In the following, we define such a normalisation and show the existence and the uniqueness of a normalised σ-basis. We then prove a statement that can be applied to detect 'bad reductions', i.e., to decide whether a given normalised σ-basis over $Frac(R/I)$ is a modular image of a normalised σ-basis over $Frac(R)$.

Definition 9.2. Let $\mathbf{p} = [p^{(1)}(x), \ldots, p^{(m)}(x)]$ be a polynomial vector, and $\mathbf{n} = [n^{(1)}, \ldots, n^{(m)}]$ a vector of degree bounds. The *critical index* of $\mathbf{p}$ is the minimal index of the vector where the defect is attained:

$$\text{critical}\,\mathbf{p} = \min\left\{i : n^{(i)} - \deg p^{(i)}(x) = \text{defect}\,\mathbf{p}, i \in \{1, \ldots, m\}\right\}.$$

$\mathbf{p}$ is *normalised* if the leading coefficient of $p^{(i)}(x)$ is 1, when i is the critical index of $\mathbf{p}$.

$\mathbf{p}$ is *reduced* with respect to another polynomial vector $\mathbf{q} = [q^{(1)}(x), \ldots, q^{(m)}(x)]$, if at the critical index i of $\mathbf{q}$ we have $\deg p^{(i)}(x) < \deg q^{(i)}(x)$. A sequence of polynomial vectors $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ is *reduced* if $\mathbf{q}_r$ is reduced by $\mathbf{q}_s$ for all $r \neq s$. Note that this implies that the critical indices of the vectors must be all different.

A sequence of polynomial vectors $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ is *sorted*, if for $r < s$

$$\text{defect}\,\mathbf{q}_r > \text{defect}\,\mathbf{q}_s \quad \text{or}$$
$$\text{defect}\,\mathbf{q}_r = \text{defect}\,\mathbf{q}_s \quad \text{and}\,\text{critical}\,\mathbf{q}_r < \text{critical}\,\mathbf{q}_s.$$

A sequence of polynomial vectors is *normalised*, if it is sorted, reduced, and all its elements are normalised.

Lemma 9.3. *Let R be an integral domain. If there is a normalised σ-basis over R for the solution space of a given rational interpolation problem, then it is uniquely determined.*

Proof. Suppose we have two normalised bases $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$ and $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$, and suppose that $\mathbf{p}_r = \mathbf{q}_r$ for $r < t$. Assume without loss of generality that $d := \text{defect}\,\mathbf{q}_t \geq \text{defect}\,\mathbf{p}_t$. We will first show that the free module generated by the vectors in $\mathbf{P}$ with defect d coincides with the free module generated by the vectors in $\mathbf{Q}$ with defect d.

Consider any $\mathbf{q}_r$ with defect $\mathbf{q}_r = d$, and expand it in the σ-basis $\mathbf{P}$: $\mathbf{q}_r = \sum_{s=1}^{m} \alpha_s(x)\mathbf{p}_s$. We want to show that $\deg \alpha_s(x) = 0$, i.e., $\alpha_s(x) \in R$ for all s.

Consider any s with $\alpha_s \neq 0$. Since $\mathbf{P}$ is a σ-basis, we have $d = \text{defect}\,\mathbf{q}_r \leq \text{defect}\,\mathbf{p}_s$. Suppose that $\text{defect}\,\mathbf{q}_r < \text{defect}\,\mathbf{p}_s$. If $s \geq t$ then $\text{defect}\,\mathbf{p}_s \leq \text{defect}\,\mathbf{p}_t$, since $\mathbf{P}$ is sorted, contradicting the assumption that $d \geq \text{defect}\,\mathbf{p}_t$. However, if $s < t$ we have $\mathbf{q}_s = \mathbf{p}_s$, and therefore that $\mathbf{q}_r$ is reduced with respect to $\mathbf{p}_s$. Thus, at the critical index i of $\mathbf{p}_s$ the degree of $q_r^{(i)}(x)$ is smaller than the degree of $p_s^{(i)}(x)$, which contradicts $\alpha_s \neq 0$. So we must have $\text{defect}\,\mathbf{q}_r = \text{defect}\,\mathbf{p}_s$, and therefore $\deg \alpha_s(x) = 0$.

Similarly, expanding any $\mathbf{p}_r$ with defect d in the σ-basis $\mathbf{Q}$ as $\mathbf{p}_r = \sum_{s=1}^{m} \beta_s(x)\mathbf{q}_s$ we obtain $\beta_s(x) \in R$: $\mathbf{Q}$ is a σ-basis, so we have $d = \text{defect}\,\mathbf{p}_r \leq \text{defect}\,\mathbf{q}_s$. Suppose that $\text{defect}\,\mathbf{p}_r < \text{defect}\,\mathbf{q}_s$. Since $\mathbf{Q}$ is sorted we must have $s < t$ and thus $\mathbf{p}_s = \mathbf{q}_s$. We then conclude as above that $\deg \beta_s(x) = 0$.

This shows that the free modules over R spanned by $\{\mathbf{p}_{r_0}, \ldots, \mathbf{p}_{r_1}\} := \{\mathbf{p}_r :$ defect $\mathbf{p}_r = d\}$ and by $\{\mathbf{q}_{s_0}, \ldots, \mathbf{q}_{s_1}\} := \{\mathbf{q}_s : \text{defect } \mathbf{q}_s = d\}$ coincide. Note that $r_0 = s_0$: because of defect $\mathbf{p}_{r_0} = d \geq$ defect $\mathbf{p}_t$ we have $r_0 \leq t$. Thus, if we had $s_0 < r_0$, then $s_0 < t$ which implies $\mathbf{p}_{s_0} = \mathbf{q}_{s_0}$, and finally defect $\mathbf{p}_{s_0} = d$, a contradiction. Suppose now $r_0 < s_0$, then $r_0 < t$, so $\mathbf{q}_{r_0} = \mathbf{p}_{r_0}$ and therefore defect $\mathbf{q}_{r_0} = d$, again a contradiction. The number of vectors in both sequences must be the same, too, so $r_1 = s_1$.

It remains to prove that the two polynomial sequences are in fact identical. If $r_0 = r_1$, i.e., the module is one-dimensional, then the condition that $\mathbf{p}_{r_0}$ and $\mathbf{q}_{r_0}$ are normalised implies that they are identical.

If $r_0 < r_1$, we first show that the critical index i of $\mathbf{p}_{r_0}$ is the same as the critical index j of $\mathbf{q}_{r_0}$. Since $\mathbf{p}_{r_0}$ is in the span of $\{\mathbf{q}_{r_0}, \ldots, \mathbf{q}_{r_1}\}$, there must also be a $\mathbf{q}_r \in \{\mathbf{q}_{r_0}, \ldots, \mathbf{q}_{r_1}\}$ with $\deg q_r^{(i)} \geq n^{(i)} - d = \deg p_{r_0}^{(i)}$. In fact, we have $\deg q_r^{(i)} = n^{(i)} - d$, since the defect of $\mathbf{q}_r$ is d. It follows that the critical index of $\mathbf{q}_r$ is at most i, and, since $\{\mathbf{q}_{r_0}, \ldots, \mathbf{q}_{r_1}\}$ is sorted, $j \leq i$. Interchanging the rôles of $\{\mathbf{p}_{r_0}, \ldots, \mathbf{p}_{r_1}\}$ and $\{\mathbf{q}_{r_0}, \ldots, \mathbf{q}_{r_1}\}$, we obtain $i \leq j$ and therefore $i = j$.

Thus also the modules spanned by $\{\mathbf{p}_{r_0+1}, \ldots, \mathbf{p}_{r_1}\}$ and $\{\mathbf{q}_{r_0+1}, \ldots, \mathbf{q}_{r_1}\}$ coincide: we can obtain both from the module spanned by $\{\mathbf{p}_{r_0}, \ldots, \mathbf{p}_{r_1}\}$ by restricting to the set of polynomial vectors $\mathbf{v}$ with $\deg v^{(i)} < n^{(i)} - d$.

Iterating the argument of the previous two paragraphs, we find that the two modules generated by $\mathbf{p}_{r_1}$ and $\mathbf{q}_{r_1}$ coincide, and since these vectors are normalised, they must be identical. In particular, their critical indices coincide. Because the sequences $\{\mathbf{p}_{r_0}, \ldots, \mathbf{p}_{r_1}\}$ and $\{\mathbf{q}_{r_0}, \ldots, \mathbf{q}_{r_1}\}$ are reduced, we can reuse the argument of the previous paragraph with i being this critical index, to obtain that also the modules spanned by $\{\mathbf{p}_{r_0}, \ldots, \mathbf{p}_{r_1-1}\}$ and $\{\mathbf{q}_{r_0}, \ldots, \mathbf{q}_{r_1-1}\}$ coincide. Now induction shows that $\mathbf{p}_r = \mathbf{q}_r$ for $r \in \{r_0, \ldots, r_1\}$, which concludes the proof of uniqueness. □

Lemma 9.4. *The solution space of every rational interpolation problem has a normalised σ-basis over a field.*

Proof. The existence of σ-bases over a field is meanwhile well known, for example Bernhard Beckermann and George Labahn prove that their algorithm produces a σ-basis in [2]. Let $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$ be a σ-basis. We will show that for any d we can replace $\{\mathbf{p}_r : \text{defect } \mathbf{p}_r \geq d\}$ by an equivalent normalised sequence. We proceed by induction: assume that $S_0 = \{\mathbf{p}_1, \ldots \mathbf{p}_k\}$ is a normalised sequence of vectors with defect $\mathbf{p}_r > d$, and let $S_1 = \{\mathbf{p}_{k+1}, \ldots, \mathbf{p}_l\}$ is a sequence of vectors with defect d, which we will successively add to S_0.

First of all we remark that replacing any vector $\mathbf{q}$ of a σ-basis $\mathbf{P}$ by $\tilde{\mathbf{q}} = \mathbf{q} + \alpha \mathbf{p}$, where $\mathbf{p} \in \mathbf{P}$ and defect $\mathbf{p} \geq$ defect $\mathbf{q} + \deg \alpha$, again yields a σ-basis. Moreover, since by Equation (14) the set of defects is an invariant of the solution space, we have defect $\tilde{\mathbf{q}} =$ defect $\mathbf{q}$.

Thus, by subtracting appropriate polynomial multiples of previous elements we may assume that each $\mathbf{p}_t \in S_1$ is reduced with respect to $\mathbf{p}_r \in S_0$. Namely, let $\mathbf{q} \in S_1$, let Q_0 be set of critical indices of elements of S_0 and let $d_i = \text{defect } \mathbf{p}_r$ with r such that i is critical index of $\mathbf{p}_r$. Consider $c_i = n^{(i)} - \deg q^{(i)}$. If $c_i > d_i$ for all $i \in Q_0$, then $\mathbf{q}$ is reduced with respect to all $\mathbf{p}_r \in S_0$, $r < t_0$. Otherwise consider $Q_1 = \{i : c_i \leq d_i\}$ and let Q_2 be set of $i \in Q_1$ such that c_i is minimal. Select the first element $\mathbf{p} \in S_0$, with critical index $i_0 \in Q_2$. Then choose α, such that $\deg(q^{(i_0)} -$

$\alpha p^{(i_0)}) < \deg q^{(i_0)}$ and replace $\mathbf{q}$ in S_1 by $\tilde{\mathbf{q}} = \mathbf{q} - \alpha\mathbf{p}$. Let $b_i = n^{(i)} - \deg \tilde{q}^{(i)}$. Note that $\deg \alpha = d_{i_0} - c_{i_0}$ and (by the definition of defect) $n^{(i)} - \deg p^{(i)} \geq d_{i_0}$ for all i, so $\deg(\alpha p^{(i)}) \leq n^{(i)} - c_{i_0}$. Moreover, if i is the critical index of $\mathbf{p}_r$ which is smaller than $\mathbf{p}$, then the defect of $\mathbf{p}_r$ is bigger or equal to the defect of $\mathbf{p}$, and, since $\mathbf{p}$ is reduced with respect to $\mathbf{p}_r$, we have $n^{(i)} - \deg p^{(i)} > d_{i_0}$ and $\deg(\alpha p^{(i)}) < n^{(i)} - c_{i_0}$. Next $\deg \tilde{q})^{(i)} \leq \max\left(\deg q^{(i)}, \deg(\alpha p^{(i)})\right)$. Consequently, we have for all i

$$
\begin{aligned}
b_i &= n^{(i)} - \deg \tilde{q}^{(i)} \\
&\geq n^{(i)} - \max\left(\deg q^{(i)}, \deg(\alpha p^{(i)})\right) \\
&= \min\left(n^{(i)} - \deg q^{(i)}, n^{(i)} - \deg(\alpha p^{(i)})\right) \\
&= \min\left(c_i, n^{(i)} - \deg(\alpha p^{(i)})\right) \\
&\geq \min(c_i, c_0),
\end{aligned}
$$

and for i corresponding to $\mathbf{p}_r$ less or equal to $\mathbf{q}$ we have $b_i \geq \min(c_i, c_0 + 1)$. This means that after a finite number of reductions passing trough $\mathbf{q}_0 = \mathbf{q}$, $\mathbf{q}_1 = \tilde{\mathbf{q}}$, $\mathbf{q}_2$, etc. we will increase $\min\{n^{(i)} - \deg q_j^{(i)}) : i \in Q_1\}$. Continuing the reduction process we will increase $\left|\{i \in Q_0 : n^{(i)} - \deg q_j^{(i)} > d_i\}\right|$ and eventually $\mathbf{q}_j$ will be reduced with respect to all elements of S_0.

Note that every element $\mathbf{p}_r \in S_0$ is automatically reduced with respect to every element $\mathbf{q} \in S_1$, because the defect of $\mathbf{p}_r$ is strictly smaller than the defect of $\mathbf{q}$: let i be the critical index of $\mathbf{p}_r \in S_0$, then $n^{(i)} - \deg q^{(i)} \geq \operatorname{defect}\mathbf{q} > \operatorname{defect}\mathbf{p}_r = n^{(i)} - \deg p_r^{(i)}$.

It remains to show how to ensure that $\mathbf{p}_r \in S_1$ is reduced with respect to $\mathbf{p}_s \in S_1$. This can be achieved by applying a simpler version of process described above: in a first step reduce $\mathbf{p}_{k+2}, \ldots, \mathbf{p}_l$ with respect to $\mathbf{p}_{k+1}$, then $\mathbf{p}_{k+3}, \ldots, \mathbf{p}_l$ with respect to the sequence $\{\mathbf{p}_{k+1}, \mathbf{p}_{k+2}\}$ and so on. Note that, since the defects of these vectors are all equal, the degree of α will always be zero. This implies that the vector replacing $\mathbf{p}_s$ will remain reduced with respect to $\mathbf{p}_r$, $r < s$.

After this step, all $\mathbf{p}_s \in S_1$ are reduced with respect to $\mathbf{p}_r \in S_1$ with $r < s$. Similarly, but going backwards, we ensure that $\mathbf{p}_s \in S_1$ are reduced with respect to $\mathbf{p}_r \in S_1$ with $r > s$. $\qquad\square$

To continue we need a lemma about linear systems.

Lemma 9.5. *Let $n_1 = \dim\ker(A)_{Frac(R)}$ and $n_2 = \dim\ker(A)_{Frac(R/I)}$. Then $n_1 \leq n_2$. Moreover, if $n_1 = n_2$, then $\ker(A)_{Frac(R/I)} = \pi\left(\ker(A)_{R_I}\right)$, where π is the quotient map and R_I is the localisation of R at I, i.e., the ring of fractions $\frac{r}{s}$ with $r \in R$ and $s \in R \setminus I$.*

Proof. For vector spaces we have $\dim\ker(A) = \dim\operatorname{dom}(A) - \operatorname{rank}(A)$, so $n_1 \leq n_2$ is equivalent to $\operatorname{rank}(A)_{Frac(R)} \geq \operatorname{rank}(A)_{Frac(R/I)}$. This inequality follows easily by considering minors of A. So it remains to prove that $n_1 = n_2$ implies $\ker(A)_{Frac(R/I)} = \pi(\ker(A)_{R_I}$. Let $m = \operatorname{rank}(A)_{Frac(R/I)}$. It is enough to prove this equality for surjective A over $Frac(R)$. Namely permuting rows of A we can write A in block form:

$$
\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}
$$

such that A_1 has m rows and $\operatorname{rank}(A_1)_{Frac(R/I)} = m$. Since $n_1 = n_2$ this means that also $\operatorname{rank}(A_1)_{Frac(R/I)} = \operatorname{rank}(A_1)_{Frac(R)}$. Next, $\ker(A)_{Frac(R)} \subset \ker(A_1)_{Frac(R)}$,

and $\dim \ker(A)_{Frac(R)} = \dim \ker(A_1)_{Frac(R)}$, so $\ker(A)_{Frac(R)} = \ker(A_1)_{Frac(R)}$. Consequently, $A_2 = 0$ on $\ker(A_1)_{Frac(R)}$, so also $\ker(A)_{R_I} = \ker(A_1)_{R_I}$. Similarly, $\ker(A)_{Frac(R/I)} = \ker(A_1)_{Frac(R/I)}$. So, indeed it is enough to prove $\pi\big(\ker(A_1)_{R_I}\big) = \ker(A_1)_{Frac(R/I)}$ and replacing A by A_1 we may assume that A is surjective. By permuting columns of A we may assume that A has block form:

$$\begin{pmatrix} A_1 & A_2 \end{pmatrix}$$

where A_1 is invertible over $Frac(R/I)$. This means that determinant of $A_1 \notin I$, so A_1 is invertible over R_I. Multiplying from the left by A_1^{-1} we may assume that A_1 is identity matrix. But then it is easy to compute the kernel over R_I: it consists of the vectors of the form $(-A_2 v, v)$, where v is an arbitrary vector in the domain of A_2. This implies that $\dim \pi(\ker(A)_{R_I} = \dim \ker(A)_{Frac(R/I)}$, which gives the claim. $\qquad\square$

Corollary 9.6. *Let $\mathbf{P}$ be a σ-basis over $Frac(R)$ and let $\mathbf{Q}$ be a σ-basis over $Frac(R/I)$. Assume that $\operatorname{defect} \mathbf{p}_r \geq \operatorname{defect} \mathbf{p}_{r+1}$ and $\operatorname{defect} \mathbf{q}_r \geq \operatorname{defect} \mathbf{q}_{r+1}$ for all r. Then $\operatorname{defect} \mathbf{p}_r \leq \operatorname{defect} \mathbf{q}_r$ for all r. If both $\mathbf{P}$ and $\mathbf{Q}$ are normalised and the defects and critical indices of $\mathbf{p}_r$ and $\mathbf{q}_r$ are the same for all r, then $\mathbf{P}$ is defined over R_I and $\mathbf{Q}$ is an image of $\mathbf{P}$ via the quotient map π.*

Proof. According to Equation (14), the dimensions of the solution spaces $\mathcal{L}_\delta^\sigma$ of the rational interpolation problem over $Frac(R)$ and $Frac(R/I)$ are given by $\sum_{r=1}^m \max(\operatorname{defect} \mathbf{p}_r - \delta, 0)$, and $\sum_{r=1}^m \max(\operatorname{defect} \mathbf{q}_r - \delta, 0)$ respectively. By Lemma 9.5, and choosing $\delta \in \mathbb{Z}$ small enough, we obtain that $\operatorname{defect} \mathbf{p}_r \leq \operatorname{defect} \mathbf{q}_r$.

When all defects are equal, the dimensions of the solution spaces coincide, so by the second part of the lemma, the solutions over $Frac(R/I)$ are images of solutions over R_I. In particular, for each r the solution $\mathbf{q}_r$ is an image of a solution $\mathbf{h}_r$ over R_I. By Equations (13) and (14), the sequence $\mathbf{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_m\}$ is a σ-basis over $Frac(R)$. It is easy to see that normalising $\mathbf{H}$ gives a σ-basis which is equal to $\mathbf{H} \mod I$ (here we need the assumption on the critical indices). Since there is a unique normalised σ-basis, $\mathbf{H} \mod I$ and $\mathbf{Q}$ coincide. $\qquad\square$

Corollary 9.7. *The previous corollary remains valid if we only assume that I is an intersection of prime ideals, and that σ-basis computation and normalisation worked over $Frac(R/I)$ without encountering division by non-invertible element.*

Proof. If $I = I_1 \cap \cdots \cap I_n$ then $Frac(R/I)$ is isomorphic to a subring of product $\Pi_{i=1}^n Frac(R/I_i)$ and the claim easily follows. $\qquad\square$

Let us stress that the corollary above means that either all modular images are 'bad reductions' (that is the dimension of the space of modular solutions is bigger than the dimension of the original space, or not all critical indices are equal) which is highly improbable, or, by normalising and rejecting solutions with larger defect or different critical indices we obtain a consistent normalisation.

10. Performance

To test the performance of the package, we ran a few examples with our package, `GFUN` (version 3.5 on Maple 11), and `Guess.m` (version 0.32 on Mathematica 7.0).

Timings are in seconds, best of three runs. `Guess.m` was run on a `Intel Core 2 E8400 @ 3 GHz` with 6MB cache and 1.8GB RAM but running a 32-bit operating system, the other two on a `Intel Pentium 4 @ 3 GHz`, 2MB cache, 1 GB RAM.

Both `Guess` and `GFUN` tried all configurations of order and degree, only `Guess.m` was run with specified order and degree of the recurrence. Since both `GFUN` and `Guess.m` look for homogeneous recurrences by default, we invoked `guessPRec` with `homogeneous==true`. We believe that neither `GFUN` not `Guess.m` check the recurrence found, thus `guessPRec` was invoked with `check=='skip`.

On the one hand, we recovered randomly generated homogeneous polynomial recurrences over $\mathbb{Q}$ from data, see Table 2. On the other hand, we computed homogeneous polynomial recurrences over $\mathbb{Q}[t]$ for the first few integer powers of the Hermite polynomials, see Table 3. (This second test was only run against `GFUN`. For comparison, we also indicate order and degree of the recurrence discovered.)

Readers should be cautious interpreting the data. Theoretically, the algorithm used by `GFUN` has lower complexity for large degrees, while `Guess.m` seems best adapted to very low degrees. However, as we explained performance depends very much on implementation detais and we lack sufficient information about the other packages to make a more general and precise statement.

11. Further work

To conclude, we would like to point out possible future directions:

- It would be very important to generalise to the multidimensional case, as already implemented by Manuel Kauers in his package. Of course, we can employ 'diagonal guessing', see [36]. I.e., we could first guess formulas for each row, and then guess formulas for the coefficients of these. However, this approach is rather slow and, more importantly, depends on the availability of many terms.
- The performance of `guessExpRat` and `guessBinRat` is very disappointing, making the two procedures nearly useless. Moreover, these two are but a toy example for real world applications, where one would like to guess formulas like

$$\det\left(\binom{3(i+j)+1}{i+j}\right) = \prod_{i=1}^{n} \frac{(6i+4)!(2i+1)!}{2(4i+2)!(4i+3)!} \sum_{i=0}^{n} \frac{n!(4n+3)!!(3n+i+2)!}{(3n+2)!i!(n-i)!(4n+2i+3)!!},$$

  as found by Ömer Eğecioğlu, Timothy Redmond and Charles Ryavec [12].
- Maybe there are other interesting operators besides Δ_n and Q_n that could be applied recursively to the sequence. Furthermore, there is a list of transformations used in *The online encyclopedia of integer sequences*, it might be rewarding to check which of those extend the class of functions already covered significantly.

References

[1] Paul W. Abrahams, *Application of LISP to sequence prediction*, Proceedings of the first ACM symposium on Symbolic and algebraic manipulation, ACM, 1966.

[2] Bernhard Beckermann and George Labahn, *A uniform approach for the fast computation of matrix-type Padé approximants*, SIAM Journal on Matrix Analysis and Applications **15** (1994), no. 3, 804–823. MR MR1282696 (95f:65030)

[3] ――――, *Recursiveness in matrix rational interpolation problems*, J. Comput. Appl. Math. **77** (1997), no. 1-2, 5–34, ROLLS Symposium (Leipzig, 1996). MR MR1440002 (97k:65025)

[4] ――――, *Fraction-free computation of matrix rational interpolants and matrix GCDs*, SIAM Journal on Matrix Analysis and Applications **22** (2000), no. 1, 114–144 (electronic). MR MR1779720 (2001e:65024)

order\degree	10	20	30	40	50
5	0.0	0.1	0.3	0.6	1.1
	0.3	1.1	1.7	4.3	5.7
	0.1	0.5	1.8	5.0	11.3
10	0.1	0.6	1.6	3.6	7.4
	1.7	5.6	8.8	20.9	27.8
	0.3	2.4	10.8	29.2	65.0
15	0.4	2.0	5.4	12.5	24.5
	5.6	16.0	42.6	59.4	77.3
	0.9	7.5	33.3	87.1	201.6
20	1.0	4.8	13.8	31.6	115.6
	19.2	39.4	99.0	137.1	179.0
	1.9	15.3	69.4	196.5	447.5
25	2.2	10.2			
	40.3	85.3			
	3.3	30.8			
30	4.2	18.7			
	75.8	162.8			
	5.2	50.1			
35	7.3	32.6			
	132.3	278.6			
	7.7	75.8			
40	11.6	51.8			
	221.0	604.9			
	11.0	120.0			
45	17.8	78.1			
	353.6	906.5			
	15.1	164.9			
50	26.1	116.5			
	536.1	309.4			
	20.0	219.7			
55	37.0	163.1			
	787.3	838.7			
	26.0	285.6			
60	52.3	222.2			
	1094.6	2516.6			
	38.6	362.3			
65	70.3				
	1509.0				
	48.0				
70	92.5				
	1927.2				
	58.5				

TABLE 2. Guessing random homogeneous recurrences with polynomial coefficients over $\mathbb{Q}$. The first line is Guess, the second GFUN, the third Guess.m.

	$H(.,t)^1$	$H(.,t)^2$	$H(.,t)^3$	$H(.,t)^4$	$H(.,t)^5$	$H(.,t)^6$
order	3	4	5	6	7	8
degree	1	3	7	13	22	34
`Guess`	0.0	0.0	0.0	0.4	5.1	46.2
`GFUN`	0.0	0.1	2.5	20.2	238.3	fail

TABLE 3. Guessing random homogeneous recurrences for powers of the Hermite polynomials. (`GFUN` ran out of memory computing the last entry.)

[5] François Bergeron and Simon Plouffe, *Computing the generating function of a series given its first few terms*, Experimental Mathematics **1** (1992), no. 4, 307–312. MR MR1257287 (95b:05008)

[6] Alin Bostan and Manuel Kauers, *Automatic classification of restricted lattice walks*, Proceedings of the 21st International Conference on Formal Power Series and Algebraic Combinatorics, Discrete Mathematics and Theoretical Computer Science, DMTCS, 2009, pp. 201–215.

[7] Alin Bostan and Manuel Kauers (with an Appendix by Mark van Hoeij), *The Complete Generating Function for Gessel Walks is Algebraic*, Proceedings of the American Mathematical Society (2010).

[8] Richard Brak and Anthony J. Guttmann, *Algebraic approximants: a new method of series analysis*, Journal of Physics. A. Mathematical and General **23** (1990), no. 24, L1331–L1337. MR MR1090002 (92c:82065)

[9] W. S. Brown, *On Euclid's algorithm and the computation of polynomial greatest common divisors*, J. Assoc. Comput. Mach. **18** (1971), 478–504. MR MR0307450 (46 #6570)

[10] Zhuliang Chen and Arne Storjohann, *A BLAS based C library for exact linear algebra on integer matrices*, ISSAC'05, ACM, New York, 2005, pp. 92–99 (electronic). MR MR2280534

[11] Philippe Dumas, *Récurrences mahlériennes, suites automatiques, études asymptotiques*, Institut National de Recherche en Informatique et en Automatique (INRIA), Rocquencourt, 1993, Thèse, Université de Bordeaux I, Talence, 1993. MR MR1346304 (96g:11021)

[12] Ömer Eğecioğlu, Timothy Redmond, and Charles Ryavec, *A multilinear operator for almost product evaluation of Hankel determinants*, Journal of Combinatorial Theory, Series A **117** (2010), no. 1, 77–103. MR MR2557881

[13] Michael E. Fisher and Helen Au-Yang, *Inhomogeneous differential approximants for power series*, Journal of Physics. A. Mathematical and General **12** (1979), no. 10, 1677–1692. MR MR545393 (81d:41005a)

[14] Manuel Kauers, *Algorithms for Nonlinear Higher Order Difference Equations*, Ph.D. thesis, RISC-Linz, Johannes Kepler University, Linz, 2005, http://www.risc.uni-linz.ac.at/people/mkauers/publications/kauers05c.pdf.

[15] _____, *Guessing Handbook*, Tech. Report 2009-07, Johannes Kepler Universitt Linz, 2009.

[16] Manuel Kauers and Christoph Koutschan, *A Mathematica package for q-holonomic sequences and power series*, Tech. Report 2007-16, SFB F013, 2007.

[17] Manuel Kauers and Christoph Koutschan, *A Mathematica package for q-holonomic sequences and power series*, The Ramanujan Journal **19** (2009), no. 2, 137–150. MR MR2511667

[18] Manuel Kauers, Christoph Koutschan, and Doron Zeilberger, *Proof of Ira Gessel's lattice path conjecture*, Proceedings of the National Academy of Sciences of the United States of America **106** (2009), no. 28, 11502–11505. MR MR2538821

[19] Martin Klazar, *Bell numbers, their relatives, and algebraic differential equations*, Journal of Combinatorial Theory, Series A **102** (2003), no. 1, 63–87. MR MR1970977 (2004d:11014)

[20] Christian Krattenthaler, *RATE: A Mathematica guessing machine*, http://mat.univie.ac.at/~kratt/rate/rate.html.

[21] Kurt Mahler, *On a class of non-linear functional equations connected with modular functions*, Journal of the Australian Mathematical Society Series A **22** (1976), no. 1, 65–118. MR MR0441867 (56 #258)

[22] Alexander Ostrowski, *Über Dirichletsche Reihen und algebraische Differentialgleichungen*, Mathematische Zeitschrift **8** (1920), no. 3-4, 241–298. MR MR1544442

[23] Malcolm Pivar and Mark Finkelstein, *Automation, Using LISP, of Inductive Inference on Sequences*, The Programming Language LISP: Its Operation and Applications (Edmund C. Berkeley and Daniel G. Bobrow, eds.), Information International, Inc., 1964, pp. 125–136.

[24] Thomas Prellberg and Richard Brak, *Critical exponents from nonlinear functional equations for partially directed cluster models*, Journal of Statistical Physics **78** (1995), no. 3, 701–730.

[25] Bruno Salvy and Paul Zimmerman, *GFUN: a maple package for the manipulation of generating and holonomic functions in one variable*, Transactions on Mathematical Software **20** (1994), no. 2, 163–177, `http://pauillac.inria.fr/algo/libraries`.

[26] Victor Shoup, *NTL: A Library for Doing Number Theory*, 2005, `http://www.shoup.net/ntl/`.

[27] Neil J. A. Sloane, *A handbook of integer sequences*, Academic Press [A subsidiary of Harcourt Brace Jovanovich, Publishers], New York-London, 1973. MR MR0357292 (50 #9760)

[28] ———, *The on-line encyclopedia of integer sequences*, Notices of the American Mathematical Society **50** (2003), no. 8, 912–915, `http://www.research.att.com/~njas/sequences`. MR MR1992789 (2004f:11151)

[29] Richard P. Stanley, *Differentiably finite power series*, European Journal of Combinatorics **1** (1980), no. 2, 175–188. MR MR587530 (81m:05012)

[30] ———, *Enumerative combinatorics. Vol. 2*, Cambridge Studies in Advanced Mathematics, vol. 62, Cambridge University Press, Cambridge, 1999, With a foreword by Gian-Carlo Rota and appendix 1 by Sergey Fomin. MR MR1676282 (2000k:05026)

[31] William T. Tutte, *Chromatic sums revisited*, Aequationes Mathematicae **50** (1995), no. 1-2, 95–134. MR MR1336864 (96i:05070)

[32] Joris van der Hoeven, *A new zero-test for formal power series*, Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (New York), ACM, 2002, pp. 117–122 (electronic). MR MR2035239

[33] Mark van Hoeij and Michael Monagan, *Algorithms for polynomial GCD computation over algebraic function fields*, ISSAC 2004, ACM, New York, 2004, pp. 297–304. MR MR2126957 (2005j:68136)

[34] Don Zagier, *Elliptic modular forms and their applications*, The 1-2-3 of modular forms, Universitext, Springer, Berlin, 2008, pp. 1–103. MR MR2409678 (2010b:11047)

[35] Doron Zeilberger, *Dave Robbins' art of guessing*, Advances in Applied Mathematics **34** (2005), no. 4, 939–954. MR MR2129005

[36] ———, *The holonomic ansatz. I. Foundations and applications to lattice path counting*, Annals of Combinatorics **11** (2007), no. 2, 227–239. MR MR2336017 (2008g:05010)

Institute of Mathematics, Wrocław University
E-mail address: `waldemar.hebisch@math.uni.wroc.pl`
URL: `http://www.math.uni.wroc.pl/~hebisch/`

Institut für Algebra, Zahlentheorie und Diskrete Mathematik, Leibniz Universität Hannover, Welfengarten 1, D-30167 Hannover, Germany
E-mail address: `martin.rubey@math.uni-hannover.de`
URL: `http://www.iazd.uni-hannover.de/~rubey/`