arXiv:math.CO/0702086v1 5 Feb 2007

EXTENDED RATE, MORE GFUN

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. Thereby we extend and complement Christian Krattenthaler's program `Rate` and the relevant parts of Bruno Salvy and Paul Zimmermann's `GFUN`.

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, `Guess`, that aims at finding such a simple formula, written for the computer algebra system `Axiom`.[1]

Some sequences are very easy to 'guess', like

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

or

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

Others are a little harder, for example

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

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

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

or

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

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 very effective. In particular, we generalise both Christian Krattenthaler's program `Rate` [9], and the guessing functions present in `GFUN` written by Bruno Salvy and Paul Zimmermann [10]. With a little manual aid, we can guess multivariate formulae as well, along the lines of Doron Zeilberger's programs `GuessRat` and `GuessHolo` [13].

We would also like to mention *The online encyclopedia of integer sequences* of Neil Sloane [11]. There, you can enter a sequence of integers and chances are good

[1]Both `Guess` and `Axiom` are freely available and can be tried out online at `http://axiom-developer.org/GuessingFormulas`. Extensive documentation on how to use `Axiom` can be found at `http://axiom-developer.org/AxiomDocumentation`.

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 117,520 sequences, entered more or less manually, and it compares the given sequence with each one of those. Besides that, it tries some simple transformations on the given sequence to find a match. Furthermore it tries some simple programs we will describe below 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 a few minutes by your computer.

On the historical side, we remark that already in 1966 Paul W. Abrahams [1] implemented a program to identify sequences given their first few terms. The first edition of 'A Handbook of Integer Sequences' by Neil Sloane appeared in 1973. In Physics, Anthony Guttmann, Richard Brak and G. S. Joyce [4] used algebraic and holonomic functions to fit series data starting from the early seventies. Finally, François Bergeron and Simon Plouffe's paper [3] from 1992 explores the idea of applying various transformations to the given sequence, to make it rational.

2. Safety and Speed

A formula for Sequence (1), is almost trivial to guess: it seems obvious that it is n^2. More generally, if we believe that the sequence in question is generated by a polynomial, we can simply apply interpolation. However, how can we 'know' that a polynomial formula is appropriate? 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. We call the number of terms used for checking the formula the *safety* of the result.

Apart from safety, the main problem we have to solve is about efficiency. For example, maybe we would like to test whether the n^{th} term of the sequence is given by a formula of the form

$$n \mapsto (a+bn)^n \frac{r(n)}{s(n)} \tag{6}$$

for some a and b and polynomials r and s. Of course, we could set up an appropriate system of polynomial equations. However, it would usually take a very long time to solve this system.

Thus, we need to find *efficient* algorithms that test for large classes of formulae. Obviously, such algorithms exist for interpolation and Padé approximation. For the present package, we implemented an efficient algorithm for a far reaching generalisation of interpolation, proposed by Bernhard Beckermann and George Labahn [2]. Furthermore, we show that there is also a way to guess sequences generated by Formula 6.

Using these algorithms our package clearly outperforms both `Rate` and `GFUN` in terms of speed as well as in the range of formulae that can be guessed.

In the following section we outline the capabilities of our package and provide comparisons of the performance of our package with `Rate` and `GFUN` where appropriate. In Section 4 we describe the most important options that modify the behaviour

of the functions. Finally, in Section 7 we outline the algorithm we use for guessing sequences generated by Formula 6.

3. Function Classes Suitable for Guessing

In this section we briefly present the function classes which are covered by our package. Throughout this section, $n \mapsto f(n)$ is the function we would like to guess, and $F(z) = \sum_{n\geq 0} f(n)z^n$ 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 $f(n)$.

`guessRec`: finds recurrences of the form

$$p\,(1, f(n), f(n+1), \ldots, f(n+k)) = 0, \tag{7}$$

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]
```

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.

Concerning q-analogues, `guessRec(q)` finds recurrences of the form (7), where p is a polynomial with coefficients in $\mathbb{K}[q, q^n]$. Similarly, we provide q-analogues

for `guessPRec` and `guessRat`. Finally, `guessExpRat(q)` 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 $\mathbb{K}[q]$. This includes, for example, Nicholas Loehr's q-analogue $[n+1]_q^{n-1}$ of Cayley's formula, where $[n]_q = 1 + q + \cdots + q^{n-1}$.

3.2. **Guessing $F(z)$.**

`guessADE`**:** finds an algebraic differential equation for $F(z)$, i.e., an equation of the form

$$p\left(1, F(z), F'(z), \ldots, F^k(z)\right) = 0, \tag{8}$$

where p is a polynomial with coefficients in $\mathbb{K}[z]$. 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]
```

returns

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

`guessHolo`**:** only looks for equations of the form (8) 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.

`guessHolo` is also the function provided by `GFUN`. Here is a comparison of average running times in seconds over several runs on the same machine for a list of n elements:

n:	50	75	100	125
`GFUN`:	1.9	5.2	22.1	63.0
`Guess`:	0.1	0.3	0.6	1.2

`guessAlg`**:** looks for an algebraic equation satisfied by $F(z)$, i.e., an equation of the form

$$p\left(1, f(x)\right) = 0,$$

the prime example being given by the Catalan numbers

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

which yields

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

`guessPade`: recognises rational generating functions. For the Fibonacci sequence given in (2), we find as likely solution

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

We provide q-analogues, replacing differentiation with q-dilation: `guessADE(q)` finds differential equations of the form

$$p\left(1, F(z), F(qz), \ldots, F(q^k z)\right) = 0, \tag{9}$$

where p is a polynomial with coefficients in $\mathbb{K}[q, z]$. Similarly, there are q-analogues for `guessHolo`, `guessAlg`, and `guessPade`.

3.3. **Operators.** The main observation made by Christian Krattenthaler in designing his program `Rate` [9] 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`. In this case the standard example is the number of alternating sign matrices

```
guess([1, 1, 2, 7, 42, 429, 7436, 218348], [guessRat],
      [guessProduct])
```

which yields

$$\prod_{q=0}^{n-1} \prod_{p=0}^{q-1} \frac{27p^2 + 54p + 24}{16p^2 + 32p + 12}.$$

Here are the average running times in seconds for our package and `Rate` over several runs on the same machine for a list of n elements:

n:	14	15	16	17	18
`Rate`:	1.0	3.3	29.7	44.9	398
`Guess`:	0.9	2.3	6.6	22.4	74

3.4. **Remarks on closure properties.**

- All of the presented classes of functions are closed under the shift operator. I.e., if a formula for the sequence $(s_1, s_2, \ldots, s_m)$ can be guessed, then the corresponding formula for $(s_2, s_3, \ldots, s_{m+1})$ can be found, too, at least in principle. Of course, it may well be that the number of terms necessary for guessing the transformed sequence is so much higher that the necessary computing power is not available.
- Except of the class of functions covered by `guessExpRat` and possibly `guessRec`, all classes presented in Sections 3.1 and 3.2 are closed under addition. I.e., if formulae for $(s_1, s_2, \ldots, s_m)$ and $(t_1, t_2, \ldots, t_m)$ can be guessed, then a formula for $(s_1+t_1, s_2+t_2, \ldots, s_m+t_m)$ can also be found.

  Of course, when including operators, as described in Section 3.3, the picture changes: for example, $n! + 1$ is not covered by the class of rational functions together with the operator Q_n.
- There are various other closure properties that some of the classes in Sections 3.1 and 3.2 enjoy. For example, all classes of generating functions in Section 3.2 are closed under the Cauchy product, while all classes of sequences in Section 3.1 except `guessExpRat` and possibly `guessRec` are closed under the Hadamard product, i.e., the element-wise multiplication.

  Furthermore, differentially finite functions are closed under substitution of an algebraic function, and the class of P-recursive sequences is closed under the operator Δ_n.

  The class of functions covered by `guessExpRat` however does not seem to satisfy any interesting closure properties. It is not even closed under addition of a constant or the Hadamard product! It would be good to have a remedy for this, i.e., a more general class of functions, which is still suitable for guessing. A natural extension would be the class of functions of the form $p(n)^n \frac{r(n)}{s(n)}$ for polynomial p, which would indeed be closed under the Hadamard product.
- The class of differentially algebraic functions has received interest in combinatorics only very recently, although it comprises several important generating functions. For example, the (exponential) generating function for the Bell numbers B_n, counting the number of partitions of $\{1, 2, \ldots, n\}$, is

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

  This function is not holonomic, but it satisfies the simple algebraic differential equation $B''B - (B')^2 - B'B = 0$.

  Furthermore, this class satisfies very interesting closure properties: it is closed under addition, multiplication, inverse, differentiation and compositional inverse. Furthermore, substituting a rational function for the main variable again yields a function satisfying an ADE. Finally, it should be noted that a zero-test for this class is available [12].
- A similar remark holds for the class of nonlinear recurrence relations. It seems that very little is known here, but a larger class, so called 'admissible recurrences' has been shown to enjoy many closure properties by Manuel Kauers [7].

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.

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

`safety`: specifies, as explained at the beginning of Section 2 the number of values reserved for testing any solutions found. The default setting is 1.

Experiments seem to indicate that for `guessADE` higher settings are appropriate than for `guessRat`. I.e., if a rational function interpolates the given list of terms, where the final term is used for testing, we can be pretty sure that the formula found is correct. By contrast, we recommend setting `safety` to 3 or 4 when using `guessADE`. For all algorithms except `guessExpRat` we recommend to omit trailing zeros.

`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`.

`maxDegree`: specifies the maximum degree of the coefficient polynomials in an algebraic differential equation or a recursion with polynomial coefficients. For rational functions with an exponential term, `maxDegree` bounds the degree of the denominator polynomial. This option is especially interesting if trying rather long sequences where it is unclear whether a solution will be found or not.

Setting `maxDegree` to -1, which is the default, specifies that the maximum degree can be arbitrary.

For example, on a `Mobile Intel(R) Pentium(R) III CPU` with 1000 megahertz, for a list `l` of 50 positive random values,

```
guessRat(l, maxDegree==-1)
```

takes roughly twelve seconds, while

```
guessRat(l, maxDegree==20)
```

returns after 0.2 seconds.

`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.

`homogeneous`: specifies whether the search space should be restricted to homogeneous algebraic differential equations or homogeneous recurrences. By default, it is set to `false`.

`maxDerivative - maxShift`: specify the maximum derivative in an algebraic differential equation, or, in a recurrence relation, the maximum shift. Setting the option to -1 specifies that the maximum derivative – the maximum shift – may be arbitrary.

`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 -1 specifies that the maximum total degree may be arbitrary. For example,

```
guessRec([1, 1, 1, 1, 2, 3, 7, 23, 59, 314, 1529, 8209, 83313,
          620297, 7869898, 126742987, 1687054711, 47301104551,
          1123424582771, 32606721084786, 1662315215971057],
```

```
                maxPower==2)
```

returns the Somos-4 recurrence

$$[f(n) : f(n)f(n+4) - f(n+1)f(n+3) - f(n+2)^2 = 0, \\ f(0) = 1, f(1) = 1, f(2) = 1, f(3) = 1],$$

whereas without limiting the power to 2, we need the first 33 values, and instead of roughly one second half a minute of computing time.

`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 -1, which means that the number of levels is only restricted by the number of terms given in the sequence.

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

5. A note on the output

The output of any function described in Section 3 is a list of formulae which seem to fit, along with an integer that states from which term on the formula is correct. The latter is necessary, because rational interpolation features sometimes *unattainable points*, as the following example shows:

```
guessRat [3, 4, 7/2, 18/5, 11/3, 26/7]
```

produces the output

```
            4n + 6
[[function= ------,order= 2]],
             n + 2
```

where $order = 2$ indicates that the first two terms of the sequence *might* not coincide with the value predicted by the returned function. A similar situation occurs, if the function generating the sequence has a singular point at $n_0 \in \mathbb{N}$, where $0 \leq n_0 < m$ and m is the number of given values. We would like to stress that this is rather a feature than a 'bug': most terms will be correct, just as in the example above, where the value at $n = 0$ is indeed 3.

To select an element from a list, a dot followed by the appropriate integer is used. Similarly, to select only the function of the above output, we write a dot followed by `function`, as in

```
guessRat [3, 4, 7/2, 18/5, 11/3, 26/7].1.function
```

For better readability we only print the – sometimes slightly edited – function in this article.

6. Some more examples

In this section we present some examples to further illustrate our package. We begin with the last two sequences in the introduction, which are taken from the 'q-world'. To guess a formula for Sequence (4), we enter

```
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

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

Note that, because of a flaw in `Axiom`, 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. Furthermore, we have to admit that the simplifying capabilities of `Axiom` 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.

Very similarly, 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

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

An interesting field of research is the enumeration of polyominoes. For example, we can recover the q-algebraic differential equation for the generating function of bar polyominoes by horizontal – marked by x, vertical perimeter – marked by y and area – marked by q, as given by Richard Brak and Thomas Prellberg. We enter

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

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

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

Unfortunately, the program takes a very long time to return the output

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

in fact, it took a little less than five hours on our machine! However, substituting any integer value for y, we obtain the result within a few seconds, thus it is not unlikely that we hit some performance bug in the polynomial library of `Axiom` here.

Let us now return to ordinary sequences. Johan Wästlund reported to the author that in 1985, Marc Mézard and Giorgio Parisi studied the integral equation

$$G(x)=\int_{-x}^{\infty}e^{-G(y)}dy,$$

that is equivalent to the functional equation

$$G'(x)=-e^{-G(-x)}.$$

Somehow, they found that a solution is given by $\log(1+e^x)$, but with the aid of our program, anybody can do it: just compute a Taylor series expansion of the functional equation, and feed the first forty values into `guessADE`:

```
guessADE([1/2, 1/8, 0, -1/192, 0, 1/2880, 0, 17/645120, \dots],
         maxShift==2)
```

After roughly nine seconds we obtain

$$[x^n]f(x) : xf''(x) + x^2f'(x)^2 + (2xf(x) - x + 2)f'(x) + f(x)^2 - f(x) = 0, \dots],$$

which is an ordinary differential equation that can be solved automatically.

Finally, we would like to point out that our package is efficient enough to do the examples of Doron Zeilberger's article[13], too. He considered the number of paths from the origin to (m, n), with north, east, and north-east steps. Using the obvious recurrence

$$H(m,n) = H(m-1,n)+H(m,n-1)+H(m-1,n-1), \text{ with } H(m,0) = H(0,n) = 1,$$

we first guess formulae $H(m,0), H(m,1), \dots$:

```
l := [guessRat([H(i, n) for i in 0..n+1]).1.function::FRAC POLY INT
      for n in 0..8];
```

(The notation `::FRAC POLY INT` instructs `Axiom` to convert the preceding expression into a fraction of polynomials over the integers) Then, in a second step, we can guess a formula for $H(m,n)$. Within a moment,

```
guessPRec(l, indexName==m)
```

returns

$$[f(m) : (m+2)f(m+2) + (-2n-1)f(m+1) + (-m-1)f(m) = 0, \\ f(0) = 1, f(1) = 2n+1],$$

as desired.

It would certainly be worth investigating whether Doron Zeilbergers algorithm's contain improvements that could be merged with our package.

7. Guessing formulae with an Abelian term

In this section we explain how `guessExpRat` works. We first present a rough description, and then show how to make the algorithm reasonably efficient.

7.1. **The basic idea.** The basic idea is to reduce the problem to rational interpolation. Let $s = (s_0, s_1, \dots, s_m)$ be the given list of values, which are elements of a field $\mathbb{K}$. We then follow the following algorithm for each pair (k, l) with $k + l = m - 4$:

(1) Let $y_n = s_n(a + bn)^{-n}$, where a and b are indeterminates. Thus, for every natural number n, y_n is an element of $\mathbb{K}(a, b)$, i.e., a rational function over $\mathbb{K}$ in a and b.
(2) Let $n \mapsto p_n(a, b)$ be the result of interpolating $(y_0, y_1, \dots, y_{m-3})$ using rational interpolation, where the degree of the numerator is k, the degree of the denominator is l. Again, p_n is an element of $\mathbb{K}(a, b)$ for every $n \in \mathbb{N}$.

It remains to determine a and b. Note that so far we have only used the values s_i for $i \leq m - 3$. Thus, in general the remaining three values s_{m-2}, s_{m-1} and s_m will overdetermine a and b, which is exactly what we had in mind. We set up a system of three equation in two variables:

(3) Let $q_i = \text{numerator}\,(p_{m+i-3} - y_{m+i-3})$ for $i \in \{1,2,3\}$. Note that q_1, q_2 and q_3 are polynomials in a and b.

We now have to solve the system of the three equations $q_i = 0$, $i \in \{1,2,3\}$ in the two variables a and b. We proceed as follows:

(4) Eliminate b by calculating the resultant r_1 of q_1 and q_3, and the resultant r_2 of q_2 and q_3. Thus, r_1 and r_2 are univariate polynomials in a.
(5) Compute the greatest common divisor r_3 of r_1 and r_2 and calculate the solutions $\tilde{a}$ of $r_3 = 0$. For simplicity, we consider only those that are elements of the field.

Note that using the greatest common divisor to determine r_3 instead of computing yet another resultant may produce extraneous solutions. After all, having computed the resultants r_1 and r_2 we only know that there exists a $\tilde{b}_1$, such that $q_1(\tilde{a},\tilde{b}_1) = q_3(\tilde{a},\tilde{b}_1) = 0$ and a $\tilde{b}_2$ such that $q_2(\tilde{a},\tilde{b}_2) = q_3(\tilde{a},\tilde{b}_2) = 0$, but it might happen that b_1 and b_2 do not coincide.

However, this is easily worked around by checking all zeros of r_3 on the polynomials q_i for $i \in \{1,2,3\}$. Since we only consider solutions $\tilde{a} \in \mathbb{K}$, this approach is much faster. Unfortunately, bivariate resultants and Gröbner base methods are – at least currently – too slow for our purposes.

(6) If the system $q_i = 0$, $i \in \{1,2,3\}$ has a solution $(\tilde{a},\tilde{b})$, return the formula $n \mapsto (\tilde{a} + \tilde{b}n)^n p_n(\tilde{a},\tilde{b})$. Otherwise, no formula of this form that generates the given sequence exists for the pair of degrees (k,l).

7.2. **Improving the algorithm.**

- Before applying the algorithm as described in the previous section, we filter out any zeros occurring in s. More precisely, let $(x_0, x_1, \ldots, x_M)$ be the list of indices such that $s_{x_i} \neq 0$, i.e., $(s_{x_0}, s_{x_1}, \ldots, s_{x_M})$ is the list of nonzero values in s. Then, mutatis mutandis, we let $n \mapsto p_n(a,b)$ be the result of interpolating the data points

$$\big((x_0, y_{x_0}), (x_1, y_{x_1}), \ldots (x_{M-3}, y_{x_{M-3}})\big),$$

  now of course for pairs (k,l) with $k + l = M - 4$. Adapting to the new situation, we set

$$q_i := \text{numerator}\,(p_{M+i-3} - y_{M+i-3}) \text{ for } i \in \{1,2,3\}.$$

  If the modified algorithm succeeds, we return

$$n \mapsto (\tilde{a} + \tilde{b}n)^n p_n(\tilde{a},\tilde{b}) \prod_{s_i=0} (n-i)$$

  as a solution.
- Furthermore, we substitute $A - x_M b$ for a in the three polynomials q_1, q_2 and q_3. This substitution reduces the degree of the polynomial q_3 in b by $x_M - x_{l+1}$: the summand that determines the leading term of q_3 in b before the substitution is easily seen to be determined by

$$(a + bx_{l+2})^{x_{l+2}} \ldots (a + bx_{M-3})^{x_{M-3}} (a + bx_M)^{x_M},$$

  whereas after the substitution is

$$(A + bx_{l+1})^{x_{l+1}} \ldots (A + bx_{M-3})^{x_{M-3}}.$$

For the rest of the algorithm we regard the polynomials q_1, q_2 and q_3 as functions in A and b.

- The computation of the two resultants r_1 and r_2 in Step (4) is the most costly step of the algorithm. To make it more efficient, we can apply the following theorem:

  Theorem 7.1. *The order and the degree of the polynomials r_1 and r_2 in A are given by the following formulae:*

  $$\operatorname{ord} r_1 = x_{M-2-l} \sum_{i=0}^{M-3-l} x_i + 2 \sum_{0\le i<j\le M-3-l} x_i x_j$$

  $$\operatorname{ord} r_2 = \begin{cases} x_{M-1-l} \sum_{i=0}^{M-3-l} x_i + 2 \sum_{0\le i<j\le M-3-l} x_i x_j & \text{if } l = 0 \\ \operatorname{ord} r_1 & \text{otherwise} \end{cases}$$

  $$\deg r_1 = (x_M + x_{M-2} + x_l) \sum_{i=l+1}^{M-3} x_i + x_M x_{M-2} + 2 \sum_{l+1\le i<j\le M-3} x_i x_j$$

  $$\deg r_2 = (x_M + x_{M-1} + x_l) \sum_{i=l+1}^{M-3} x_i + x_M x_{M-1} + 2 \sum_{l+1\le i<j\le M-3} x_i x_j$$

  Proof. The proof is given in the appendix. □

  Since the two resultants have rather large orders, it is much faster to calculate them by interpolation. For example, the values of

  $$\operatorname{resultant}_b \big(q_1(A = j, b), q_3(A = j, b)\big) j^{-\operatorname{ord} res_1}$$

  at $j \in \{1, 2, \ldots, \deg r_1 - \operatorname{ord} r_1 + 1\}$ determine r_1 uniquely.

Although these improvements yield a significant speedup, more work needs yet to be done to be able to guess more complicated formulae with `guessExpRat`. We should point out that the bottleneck still seems to be the computation of the two resultants.

For example,

```
guessExpRat [(2*n+3)^n*(4*n^3+3*n^2+2*n+1) for n in 0..6]
```

takes roughly one second, while

```
guessExpRat [(2*n+3)^n*(6*n^5+5*n^4+4*n^3+3*n^2+2*n+1)
             for n in 0..8]
```

takes already 16 seconds on a `Mobile Intel(R) Pentium(R) III CPU` with 1000 megahertz.

8. Further work

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

- One major possibility of speeding up the guessing algorithms is to use machine precision floating point arithmetic. Experiments indicate that in most cases, floating point arithmetic is still close enough to the truth so it does not reject good canditates. However, this would make the implementation of a good interpolation algorithm necessary...

- Obviously, it would be nice to be able to guess formulae of a more general type then those guessed by `guessExpRat`. This would involve finding a fast algorithm that tests whether an overdetermined systems of equations has a solution. Much work has been done in this direction, for starting points see [5, 6, 8], but strange enough, the naive elimination algorithm detailed above seems to be the fastest method available currently.
- 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.

Appendix: Proof of Theorem 7.1

Proof. Setting $\psi(j) := (A + z_j b)^{x_j}$ for $j \in \{0, 2, \ldots, M-1\}$ and $\psi(M) := A^{x_M}$, it is quite straightforward to see that, for certain non-zero constants z_j and $c_{j_0,j_2,\ldots,j_l}$, $d_{j_0,j_2,\ldots,j_l}$ and $e_{j_0,j_2,\ldots,j_l}$, we have

(10)
$$q_1 = \psi(0)\ldots\psi(M-1)\psi(M-2) \sum_{0\le j_0<j_2<\cdots<j_l\le M-2} c_{j_0,j_2,\ldots,j_l}/\big(\psi(j_0)\ldots\psi(j_l)\big),$$

(11)
$$q_2 = \psi(0)\ldots\psi(M-3)\psi(M-1) \sum_{\substack{0\le j_0<j_2<\cdots<j_l\le M-1\\ j_i\ne M-2}} d_{j_0,j_2,\ldots,j_l}/\big(\psi(j_0)\ldots\psi(j_l)\big),$$

and

(12)
$$q_3 = \psi(0)\ldots\psi(M-3)\psi(M) \sum_{\substack{0\le j_0<j_2<\cdots<j_l\le M\\ j_i\notin\{M-1,M-2\}}} e_{j_0,j_2,\ldots,j_l}/\big(\psi(j_0)\ldots\psi(j_l)\big),$$

To determine the order and degree of the resultants r_1 and r_2, we use the following formula:

$$\operatorname{resultant}_b\big(P(A,b),Q(A,b)\big) = \prod_{\tilde b(A):P(A,\tilde b(A))=0} Q\big(A,\tilde b(A)\big),$$

where the product is over all zeros of P, with multiplicities. We obtain

$$\operatorname{ord}_A \operatorname{resultant}_b(P(A,b),Q(A,b)) = \sum_{\tilde b(A):P(A,\tilde b(A))=0} \operatorname{ord}_A Q\big(A,\tilde b(A)\big), \tag{13}$$

and

$$\deg_A \operatorname{resultant}_b(P(A,b),Q(A,b)) = \sum_{\tilde b(A):P(A,\tilde b(A))=0} \deg_A Q\big(A,\tilde b(A)\big). \tag{14}$$

Of course, $Q\big(A,\tilde b(A)\big)$ is not necessarily a polynomial, so we have to interpret $\operatorname{ord}_A Q\big(A,\tilde b(A)\big)$ as the order of the leading term in the Puiseux expansion of Q as A tends to zero and $\deg_A Q\big(A,\tilde b(A)\big)$ as the exponent of the leading term in the asymptotic expansion of Q as A tends to infinity.

Thus, we are led to determine order and degree of $\tilde b(A)$, where $q_i\big(A,\tilde b(A)\big)$ vanishes, for $i \in \{1,2\}$. We determine these values for q_1, for q_2 they follow by replacing

x_{M-3} with x_{M-2}. Below we compute the Puiseux expansion of $\tilde{b}(A)$. Since the calculations for the asymptotic expansions are essentially the same, we leave them to the reader.

Setting $A = 0$ in (10), we obtain a polynomial in b, whose order is

$$x_0 + \cdots + x_{M-l-3}$$

and whose degree is

$$x_{l+1} + \cdots + x_{M-2}.$$

Thus, (10) has

$$x_{M-l-2} + \cdots + x_{M-2} - x_0 - \cdots - x_l$$

zeros of the form $\tilde{b}(A) = c + o(1)$, for some constants c. Suppose now that $\tilde{b}(A) = cA^r\big(1 + o(1)\big)$ for some $r > 0$.

Plugging in this expression into (10), multiplying with $A^{-r(x_0+\cdots+x_{M-l-3})}$, and letting A tend to zero, we find that all summands vanish, except

$$\begin{aligned} A^{-r(x_0+\cdots+x_{M-l-3})}&\psi(1)\cdots\psi(M-l-3) \\ &= \big(A^{1-r} + z_0c + o(1)\big)^{x_0} \cdots \big(A^{1-r} + z_{M-l-3}c + o(1)\big)^{x_{M-l-3}}. \end{aligned}$$

Thus, for $r < 1$, this expression tends to a non-zero constant, which implies that $\tilde{b}(A) = cA^r\big(1 + o(1)\big)$ cannot be a root of (10) for $r < 1$. However, for $r = 1$, we obtain

$$\big(1 + z_0c + o(1)\big)^{x_0} \cdots \big(1 + z_{M-l-3}c + o(1)\big)^{x_{M-l-3}},$$

which indeed vanishes for $c = -1/z_j$, $j \in \{0, \ldots, M-l-3\}$, with multiplicity x_j. It is easy to check by counting that we have thus obtained all roots of (10).

By setting $\tilde{b}(A) = -\frac{1}{z_j}A + cA^r\big(1 + o(1)\big)$, we can also determine the exponent of the next term in the Puiseux expansion of $\tilde{b}(A)$. Multiplying (10) with $A^{-(1-r)x_j - x_0 - \cdots - x_{M-l-3}}$ we find that the zeros must be of the form

$$\tilde{b}(A) = -\frac{1}{z_j}A + cA^{\frac{x_{M-l-2}}{x_j}}\big(1 + o(1)\big), \quad j \in \{0, \ldots, M-l-3\},$$

for some non-zero constant c, the multiplicity being x_j. Using (13) it is now an easy matter to derive the order of r_1 and r_2. For the former resultant, we find that for the any zero the term of minimal order in q_3 is given by

$$\psi(0)\cdots\psi(M-l-3).$$

For the j^{th} zero this evaluates to $x_0 + \cdots + x_{M-l-2} - x_j$, taking into account that $\operatorname{ord}_A \psi(j) = x_{M-l-2}$ because of cancellation.

Since the j^{th} zero has multiplicity x_j, we obtain for the order of r_1

$$\sum_{j=0}^{M-l-3} x_j \left(\sum_{i=0}^{M-l-2} x_i - x_j \right).$$

Some trivial manipulations then yield the expressions stated in the theorem. □

References

[1] Paul W. Abrahams, *Application of LISP to sequence prediction*, Proceedings of the first ACM symposium on Symbolic and algebraic manipulation, ACM, 1966, doi:10.1145/800005.807960.

[2] Bernhard Beckermann and George Labahn, *Fraction-free computation of matrix rational interpolants and matrix GCDs*, SIAM Journal on Matrix Analysis and Applications **22** (2000), no. 1, 114–144 (electronic).

[3] 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.

[4] R. Brak and A. J. Guttmann, *Algebraic approximants: a new method of series analysis*, Journal of Physics. A. Mathematical and General **23** (1990), no. 24, L1331–L1337.

[5] David Cox, John Little, and Donal O'Shea, *Using algebraic geometry*, Graduate Texts in Mathematics, vol. 185, Springer-Verlag, New York, 1998.

[6] Ioannis Z. Emiris and Victor Y. Pan, *Improved algorithms for computing determinants and resultants*, J. Complexity **21** (2005), no. 1, 43–71.

[7] 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`.

[8] Amit Khetan, *The resultant of an unmixed bivariate system*, J. Symbolic Comput. **36** (2003), no. 3-4, 425–442, International Symposium on Symbolic and Algebraic Computation (ISSAC'2002) (Lille).

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

[10] 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`.

[11] N. J. A. Sloane, *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`.

[12] 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).

[13] Doron Zeilberger, *The HOLONOMIC ANSATZ I. Foundations and Applications to Lattice Path Counting*.

Fakultät für Mathematik, Universtät Wien