QSUM
A Package for the Indefinite and Definite Summation of q-hypergeometric Terms

Harald Böing
Wolfram Koepf

Konrad Zuse Zentrum für Informationstechnik Berlin
Takustr. 7
D-14195 Berlin-Dahlem

email: koepf@zib.de

1 Introduction

This package is an implementation of the q-analogues of Gosper's and Zeilberger's[1] algorithm for indefinite, and definite summation of q-hypergeometric terms, respectively.

An expression a_k is called a *q-hypergeometric term*, if a_k/a_{k-1} is a rational function with respect to q^k. Most q-terms are based on the *q-shifted factorial* or *qpochhammer*. Other typical q-hypergeometric terms are ratios of products of powers, q-factorials, q-binomial coefficients, and q-shifted factorials that are integer-linear in their arguments.

[1] The `ZEILBERG` package (see [7]) contains the hypergeometric versions. Those algorithms are described in [4],[11],[12] and [6].

2 Elementary q-Functions

Our package supports the input of the following elementary q-functions:

- `qpochhammer(a,q,infinity)`

$$(a;\, q)_\infty := \prod_{j=0}^{\infty} \left(1 - a\, q^j\right)$$

- `qpochhammer(a,q,k)`

$$(a;\, q)_k := \begin{cases} \prod_{j=0}^{k-1} \left(1 - a\, q^j\right) & \text{if } k > 0 \\ 1 & \text{if } k = 0 \\ \prod_{j=1}^{k} \left(1 - a\, q^{-j}\right)^{-1} & \text{if } k < 0 \end{cases}$$

- `qbrackets(k,q)`

$$[q, k] := \frac{q^k - 1}{q - 1}$$

- `qfactorial(k,q)`

$$[k]_q! := \frac{(q;\, q)_k}{(1 - q)^k}$$

- `qbinomial(n,k,q)`

$$\binom{n}{k}_q := \frac{(q;\, q)_n}{(q;\, q)_k \cdot (q;\, q)_{n-k}}$$

Furthermore it is possible to use an abbreviation for the *generalized q-hypergeometric series* (*basic generalized hypergeometric series*, see e. g. [3], Chapter 1) which is defined as:

$$ {}_r\phi_s \left[\begin{array}{c} a_1, a_2, \ldots, a_r \\ b_1, b_2, \ldots, b_s \end{array} \middle|\, q, z \right] := \sum_{k=0}^{\infty} \frac{(a_1, a_2, \ldots, a_r;\, q)_k}{(b_1, b_2, \ldots, b_s;\, q)_k} \frac{z^k}{(q;\, q)_k} \left[(-1)^k\, q^{\binom{k}{2}}\right]^{1+s-r}$$

where $(a_1, a_2, \ldots, a_r;\, q)_k$ is a short form to write the product $\prod_{j=1}^{r} (a_j;\, q)_k$. An ${}_r\phi_s$ series terminates if one of its numerator parameters is of the form q^{-n}

with $n \in \mathbb{N}$. The additional factor $\left[(-1)^k\, q^{\binom{k}{2}}\right]^{1+s-r}$ (which does not occur in the corresponding definition of the *generalized hypergeometric function*) is due to a *confluence process*. With this factor one gets the simple formula:

$$\lim_{a_r \to \infty} {}_r\phi_s \left[\begin{array}{c} a_1, a_2, \ldots, a_r \\ b_1, b_2, \ldots, b_s \end{array} \middle| q, z \right] = {}_{r-1}\phi_s \left[\begin{array}{c} a_1, a_2, \ldots, a_{r-1} \\ b_1, b_2, \ldots, b_s \end{array} \middle| q, z \right].$$

Another variation is the *bilateral basic hypergeometric series* (see e. g. [3], Chapter 5) that is defined as

$${}_r\psi_s \left[\begin{array}{c} a_1, a_2, \ldots, a_r \\ b_1, b_2, \ldots, b_s \end{array} \middle| q, z \right] := \sum_{k=-\infty}^{\infty} \frac{(a_1, a_2, \ldots, a_r;\, q)_k}{(b_1, b_2, \ldots, b_s;\, q)_k}\, z^k \left[(-1)^k\, q^{\binom{k}{2}}\right]^{s-r}.$$

The *summands* of those generalized q-hypergeometric series may be entered by

- `qphihyperterm({a1,a2,...,a3},{b1,b2,...,b3},q,z,k)` and
- `qpsihyperterm({a1,a2,...,a3},{b1,b2,...,b3},q,z,k)`

respectively.

3 q-Gosper Algorithm

The q-Gosper algorithm [8] is a *decision procedure*, that decides by algebraic calculations whether or not a given q-hypergeometric term a_k has a q-hypergeometric term antidifference g_k, i. e. $a_k = g_k - g_{k-1}$ with g_k/g_{k-1} rational in q^k. The ratio g_k/a_k is also rational in q^k — an important fact which makes the *rational certification* (see § 4) of Zeilberger's algorithm possible. If the procedure is successful it returns g_k, in which case we call a_k *q-Gosper-summable*. Otherwise *no q-hypergeometric antidifference exists*. Therefore if the q-Gosper algorithm does not return a q-hypergeometric antidifference, it has *proved* that no such solution exists, an information that may be quite useful and important.

Any antidifference is uniquely determined up to a constant, and is denoted by

$$g_k = \sum a_k\, \delta_k\ .$$

Finding g_k given a_k is called *indefinite summation*. The antidifference operator Σ is the inverse of the downward difference operator $\nabla a_k = a_k - a_{k-1}$.

There is an analogous summation theory corresponding to the upward difference operator $\Delta a_k = a_{k+1} - a_k$.

In case, an antidifference g_k of a_k is known, any sum $\sum_{k=m}^{n} a_k$ can be easily calculated by an evaluation of g at the boundary points like in the integration case:

$$\sum_{k=m}^{n} a_k = g_n - g_{m-1}$$

4 q-Zeilberger Algorithm

The q-Zeilberger algorithm [8] deals with the *definite summation* of q-hypergeometric terms $\mathrm{f}(n,k)$ wrt. n and k:

$$\mathrm{s}(n) := \sum_{k=-\infty}^{\infty} \mathrm{f}(n,k)$$

Zeilberger's idea is to use Gosper's algorithm to find an inhomogeneous recurrence equation with polynomial coefficients for $\mathrm{f}(n,k)$ of the form

$$\sum_{j=0}^{J} \sigma_j(n) \cdot \mathrm{f}(n+j,k) = \mathrm{g}(k) - \mathrm{g}(k-1), \tag{1}$$

where $\mathrm{g}(k)/\mathrm{f}(k)$ is rational in q^k and q^n. Assuming finite support of $\mathrm{f}(n,k)$ wrt. k (i. e. $\mathrm{f}(n,k) = 0$ for any n and all sufficiently large k) we can sum equation (1) over all $k \in \mathbb{Z}$. Thus we receive a homogeneous recurrence equation with polynomial coefficients (called *holonomic equation*) for $\mathrm{s}(n)$:

$$\sum_{j=0}^{J} \sigma_j(n) \cdot \mathrm{s}(n+j) = 0 \tag{2}$$

At this stage the implementation assumes that the summation bounds are infinite and the input term has finite support wrt. k. If those input requirements are not fulfilled the resulting recursion is probably not valid. Thus we strongly advise the user to check those requirements.

Despite this restriction you may still be able to get valuable information by the program: On request it returns the left hand side of the recurrence equation (2) *and* the antidifference $\mathrm{g}(k)$ of equation (1).

Once you have the certificate $\mathrm{g}(k)$ it is trivial (at least theoretically) to prove equation (2) as long as the input requirements are fulfilled. Let's assume somone gives us equation (1). If we divide it by $\mathrm{f}(n,k)$ we get a rational identity (in q^n and q^k) —due to the fact that $\mathrm{g}(k)/\mathrm{f}(n,k)$ is rational in q^n and q^k. Once we confirmed this identity we sum equation (1) over $k \in \mathbb{Z}$:

$$\sum_{k\in\mathbb{Z}} \sum_{j=0}^{J} \sigma_j(n) \cdot \mathrm{f}(n+j,k) = \sum_{k\in\mathbb{Z}} (\mathrm{g}(k) - \mathrm{g}(k-1)), \tag{3}$$

Again we exploit the fact that $\mathrm{g}(k)$ is a rational multiple of $\mathrm{f}(n,k)$ and thus $\mathrm{g}(k)$ has *finite support* which makes the telescoping sum on the right hand side vanish. If we exchange the order of summation we get equation (2) which finishes the proof.

Note that we may relax the requirements for $\mathrm{f}(n,k)$: An infinite support is possible as long as $\lim\limits_{k\to\infty} \mathrm{g}(k) = 0$. (This is certainly true if $\lim\limits_{k\to\infty} \mathrm{p}(k)\,\mathrm{f}(k) = 0$ for all polynomials $\mathrm{p}(k)$.)

For a quite general class of q-hypergeometric terms (*proper q-hypergeometric terms*) the q-Zeilberger algorithm always finds a recurrence equation, not necessarily of lowest order though. Unlike Zeilberger's original algorithm its q-analogue more often fails to determine the recursion of lowest possible order, however (see [10]).

If the resulting recurrence equation is of first order

$$\mathrm{a}(n)\,\mathrm{s}(n-1) + \mathrm{b}(n)\,\mathrm{s}(n) = 0\ ,$$

$\mathrm{s}(n)$ turns out to be a q-hypergeometric term (as a and b are polynomials in q^n), and a q-hypergeometric solution can be easily established using a suitable initial value.

If the resulting recurrence equation has order larger than one, this information can be used for identification purposes: Any other expression satisfying the same recurrence equation, and the same initial values, represents the same function.

Our implementation is mainly based on [8] and on the hypergeometric analogue described in [6]. More examples can be found in [3], [2], some of which are contained in the test file `qsum.tst`.

5 REDUCE operator QGOSPER

The QSUM package must be loaded by:

```
1: load qsum;
```

The `qgosper` operator is an implementation of the q-Gosper algorithm.

- `qgosper(a,q,k)` determines a q-hypergeometric antidifference. (By default it returns a *downward* antidifference, which may be changed by the switch `qgosper_down`; see also § 8.) If it does not return a q-hypergeometric antidifference, then such an antidifference does not exist.
- `qgosper(a,q,k,m,n)` determines a closed formula for the definite sum $\sum_{k=m}^{n} a_k$ using the q-analogue of Gosper's algorithm. This is only successful if q-Gosper's algorithm applies.

Examples: The following two examples can be found in [3] ((II.3) and (2.3.4)).

```
2: qgosper(qpochhammer(a,q,k)*q^k/qpochhammer(q,q,k),q,k);

   k
 (q *a - 1)*qpochhammer(a,q,k)
-------------------------------
  (a - 1)*qpochhammer(q,q,k)

3: qgosper(qpochhammer(a,q,k)*qpochhammer(a*q^2,q^2,k)*
   qpochhammer(q^(-n),q,k)*q^(n*k)/(qpochhammer(a,q^2,k)*
   qpochhammer(a*q^(n+1),q,k)*qpochhammer(q,q,k)),q,k);

     k*n   k          k    n                1
( - q   *(q *a - 1)*(q  - q )*qpochhammer(----,q,k)
                                            n
                                           q

                2  2                              2*k          n
 *qpochhammer(a*q ,q ,k)*qpochhammer(a,q,k))/((q   *a - 1)*(q  - 1)

              n                         2
 *qpochhammer(q *a*q,q,k)*qpochhammer(a,q ,k)*qpochhammer(q,q,k))
```

Here are some other simple examples:

```
4: qgosper(qpochhammer(q^(-n),q,k)*z^k/qpochhammer(q,q,k),q,k);

***** No q-hypergeometric antidifference exists.

5: off qgosper_down;

6: qgosper(q^k*qbrackets(k,q),q,k);

     k            k
  - q *(q + 1 - q )*qbrackets(k,q)
-----------------------------------
       k
     (q  - 1)*(q + 1)*(q - 1)

7: on qgosper_down;

8: qgosper(q^k,q,k,0,n);

  n
 q *q - 1
----------
  q - 1
```

6 REDUCE operator QSUMRECURSION

The `qsumrecursion` operator is an implementation of the q-Zeilberger algorithm. It tries to determine a homogeneous recurrence equation for $\mathrm{summ}(n)$ wrt. n with polynomial coefficients (in n), where

$$\mathrm{summ}(n) := \sum_{k=-\infty}^{\infty} \mathrm{f}(n,k).$$

If successful the left hand side of the recurrence equation (2) is returned.

There are three different ways to pass a summand $\mathrm{f}(n,k)$ to `qsumrecursion`:

- `qsumrecursion(f,q,k,n)`, where `f` is a q-hypergeometric term wrt. `k` and `n`, `k` is the summation variable and `n` the recursion variable, `q` is a symbol.

- `qsumrecursion(upper,lower,q,z,n)` is a shortcut for `qsumrecursion(qphihyperterm(upper,lower,q,z,k),q,k,n)`
- `qsumrecursion(f,upper,lower,q,z,n)` is a similar shortcut for `qsumrecursion(f*qphihyperterm(upper,lower,q,z,k),q,k,n)`,

i. e. `upper` and `lower` are lists of upper and lower parameters of the generalized q-hypergeometric function. The third form is handy if you have any additional factors.

For all three instances the following variations are allowed:

- If for some reason the recursion order is known in advance you can specify it as an additional (*optional*) argument at the very end of the parameter sequence. There are two ways. If you just specify a positive integer, `qsumrecursion` looks only for a recurrence equation of this order. You can also specify a range by a list of two positive integers, i. e. the first one specifying the lowest and the second one the highest order.

  By default `qsumrecursion` will search for recurrences of order from 1 to 5. (The global variable `qsumrecursion_recrange!*` controls this behavior, see § 8.)
- Usually `qsumrecursion` uses `summ` as a name for the summ-function defined above. If you want to use another operator, say e. g. `s`, then the following syntax applies: `qsumrecursion(f,q,k,s(n))`

As a first example we want to consider the *q-binomial theorem*:

$$\sum_{k=0}^{\infty} \frac{(a;\,q)_k}{(q;\,q)_k} z^k = \frac{(a\,z;\,q)_\infty}{(z;\,q)_\infty},$$

provided that $|z|, |q| < 1$. It is the q-analogue of the binomial theorem in the sense that

$$\lim_{q\to 1^-} \sum_{k=0}^{\infty} \frac{(q^a;\,q)_k}{(q;\,q)_k} z^k \quad = \quad \sum_{k=0}^{\infty} \frac{(a)_k}{k!} z^k \quad = \quad (1-z)^{-a}\ .$$

For $a := q^{-n}$ with $n \in \mathbb{N}$ our implementation gets:

```
9: qsumrecursion(qpochhammer(q^(-n),q,k)*z^k/
   qpochhammer(q,q,k),q,k,n);

        n                      n
 - ((q   - z)*summ(n - 1) - q *summ(n))
```

Notice that the input requirements are fulfilled. For $n \in \mathbb{N}$ the summand is zero for all $k > n$ as $(q^{-n};\, q)_k = 0$ and the $(q;\, q)_k$-term in the denominator makes the summand vanish for all $k < 0$.

With the switch `qsumrecursion_certificate` it is possible to get the antidifference g_k described above. When switched on, `qsumrecursion` returns a list with five entries, see § 8. For the last example we get:

```
10: on qsumrecursion_certificate;

11: proof:= qsumrecursion(qpochhammer(q^(-n),q,k)*z^k/
    qpochhammer(q,q,k),q,k,n);

                 n                     n
proof :=  - ((q   - z)*summ(n - 1) - q *summ(n)),

                k    n
            - (q  - q )*z
          ---------------,
                 n
                q  - 1

            k             1
           z *qpochhammer(----,q,k)
                           n
                          q
          -------------------------,
              qpochhammer(q,q,k)

          k,

          downward_antidifference

12: off qsumrecursion_certificate;
```

Let's define the list entries as `{rec,cert,f,k,dir}`. If you substitute summ$(n+j)$ by f$(n+j, k)$ in `rec` then you obtain the left hand side of equation (1), where `f` is the input summand. The function g(k) := `f*cert` is the corresponding antidifference, where `dir` states which sort of antidifference was calculated `downward_antidifference` or `upward_antidifference`, see also § 8. Those informations enable you to prove the recurrence equation for the sum or supply you with the necessary informations to determine an inhomogeneous recurrence equation for a sum with nonnatural bounds.

For our last example we can now calculate both sides of equation (1):

```
13: lhside:= qsimpcomb(sub(summ(n)=part(proof,3),
    summ(n-1)=sub(n=n-1,part(proof,3)),part(proof,1)));

               k    k    n          n                        1
             z *(q *(q   - z) + q *(z - 1))*qpochhammer(----,q,k)
                                                          n
                                                         q
lhside := -----------------------------------------------------
                              n
                            (q  - 1)*qpochhammer(q,q,k)

14: rhside:= qsimpcomb((part(proof,2)*part(proof,3)-
    sub(k=k-1,part(proof,2)*part(proof,3))));

                  k    k    n       n   k                    1
              - z *((q  - q )*z - q *(q  - 1))*qpochhammer(----,q,k)
                                                             n
                                                            q
rhside := ---------------------------------------------------------
                                n
                              (q  - 1)*qpochhammer(q,q,k)

15: qsimpcomb((rhside-lhside)/part(proof,3));

0
```

Thus we have proved the validity of the recurrence equation.

As some other examples we want to consider some generalizations of orthogonal polynomials from the Askey–Wilson–scheme [9]: The q-Laguerre (3.21), q-Charlier (3.23) and the continuous q-Jacobi (3.10) polynomials.

```
16: operator qlaguerre,qcharlier;

17: qsumrecursion(qpochhammer(q^(alpha+1),q,n)/qpochhammer(q,q,n),
    {q^(-n)}, {q^(alpha+1)}, q, -x*q^(n+alpha+1), qlaguerre(n));

           n       alpha + n   n
((q + 1 - q )*q - q         *(q *x + q))*qlaguerre(n - 1)

      alpha + n                           n
 + ((q         - q)*qlaguerre(n - 2) + (q  - 1)*qlaguerre(n))*q
```

```
18: qsumrecursion({q^(-n),q^(-x)},{0},q,-q^(n+1)/a,qcharlier(n));

      x             n       n       2*n
 - ((q *((q + 1 - q )*a + q )*q - q   )*qcharlier(n - 1)

  x    n          n
  + q *((q  + a*q)*(q  - q)*qcharlier(n - 2) - qcharlier(n)*a*q))

19: on qsum_nullspace;

20: term:= qpochhammer(q^(alpha+1),q,n)/qpochhammer(q,q,n)*
    qphihyperterm({q^(-n),q^(n+alpha+beta+1),
    q^(alpha/2+1/4)*exp(I*theta), q^(alpha/2+1/4)*exp(-I*theta)},
    {q^(alpha+1), -q^((alpha+beta+1)/2), -q^((alpha+beta+2)/2)},
    q,q,k)$

21: qsumrecursion(term,q,k,n,2);

     n  i*theta   alpha   beta   n
 - ((q *e       *(q     *(q    *(q *(q + 1) - q) - q

             alpha + beta + n           n    beta + n
           + q                *(q + 1 - q  - q        )) -

        (alpha + beta)/2   alpha   n                beta + n           n
       q                *(q     *(q *(q + 1) - q + q        *(q + 1 - q ))

              2*alpha + beta + 2*n
           - (q                    + q)))*(sqrt(q) + q) +

     (2*alpha + 1)/4   2*i*theta        alpha + beta + 2*n    2
    q               *(e          + 1)*(q                  - q )

       alpha + beta + 2*n         alpha + beta + 2*n
    *(q                  - 1))*(q                  - q)*summ(n - 1) -

    i*theta    (alpha + beta + 2*n)/2   (alpha + beta + 2*n)/2
   e       *((q                      *(q                      + q)

               (alpha + beta + 2*n)/2
             *(q                      - q)*(sqrt(q) + q) +

              (2*alpha + 2*beta + 4*n + 1)/2
             (q                              + q)
```

```
    alpha + beta + 2*n    2     alpha + beta + n
 *(q                  - q ))*(q                 - 1)

  n                 alpha               alpha + beta + 2*n
*(q  - 1)*summ(n) + (q     *(sqrt(q)*q + q                  )

     (3*alpha + beta + 2*n)/2
    + q                         *(sqrt(q) + q))

   alpha + beta + 2*n        alpha + n        beta + n
*(q                  - 1)*(q          - q)*(q         - q)

*summ(n - 2)))
```

```
22: off qsum_nullspace;
```

The setting of `qsum_nullspace` (see [10] and § 8) results in a faster calculation of the recurrence equation for this example.

7 Simplification Operators

An essential step in the algorithms introduced above is to decide whether a term a_k is q-hypergeometric, i.e. if the ratio a_k/a_{k-1} is rational in q^k.

The procedure `qsimpcomb` provides this facility. It tries to simplify all exponential expressions in the given term and applies some transformation rules to the known elementary q-functions as `qpochhammer`, `qbrackets`, `qbinomial` and `qfactorial`. Note that the procedure may fail to completely simplify some expressions. This is due to the fact that the procedure was designed to simplify ratios of q-hypergeometric terms in the form $\mathrm{f}(k)/\mathrm{f}(k-1)$ and not arbitrary q-hypergeometric terms.

E.g. an expression like $(a;\, q)_{-n} \cdot (a/q^n;\, q)_n$ is not recognized as 1, despite the transformation formula

$$(a;\, q)_{-n} \;=\; \frac{1}{(a/q^n;\, q)_n},$$

which is valid for $n \in \mathbb{N}$.

Note that due to necessary simplification of powers, the switch `precise` is (locally) turned off in `qsimpcomb`. This might produce wrong results if the input term contains e.g. complex variables.

The following synomyms may be used:

- `up_qratio(f,k)` or `qratio(f,k)` for `qsimpcomb(sub(k=k+1,f)/f)` and
- `down_qratio(f,k)` for `qsimpcomp(f/sub(k=k-1,f))`.

8 Global Variables and Switches

The following switches can be used in connection with the `QSUM` package:

- `qsum_trace`, default setting is off. If it is turned on some intermediate results are printed.
- `qgosper_down`, default setting is on. It determines whether `qgosper` returns a downward or an upward antidifference g_k for the input term a_k, i.e. $a_k = g_k - g_{k-1}$ or $a_k = g_{k+1} - g_k$ respectively.
- `qsumrecursion_down`, default setting is on. If it is switched on a downward recurrence equation will be returned by `qsumrecursion`. Switching it off leads to an upward recurrence equation.
- `qsum_nullspace`, default setting is off. The antidifference $\mathrm{g}(k)$ is always a rational multiple (in q^k) of the input term $\mathrm{f}(k)$. `qgosper` and `qsumrecursion` determine this certificate, which requires solving a set of linear equations. If the switch `qsum_nullspace` is turned on a modified nullspace-algorithm will be used for solving those equations. In general this method is slower. However if the resulting recurrence equation is quite complicated it might help to switch on `qsum_nullspace`. See also [5] and [10].
- `qgosper_specialsol`, default setting is on. The antidifference $\mathrm{g}(k)$ which is determined by `qgosper` might not be unique. If this switch is turned on, just one special solution is returned. If you want to see all solutions, you should turn the switch off.
- `qsumrecursion_exp`, default setting is off. This switch determines if the coefficients of the resulting recurrence equation should be factored. Turning it off might speed up the calculation (if factoring is complicated). Note that when turning on `qsum_nullspace` usually no speedup occurs by switching `qsumrecursion_exp` on.
- `qsumrecursion_certificate`, default off. As Zeilberger's algorithm delivers a recurrence equation for a q-hypergeometric term $\mathrm{f}(\mathrm{n},\mathrm{k})$, see

equation (1), this switch is used to get all necessary informations for proving this recurrence equation.

If it is set on, instead of simply returning the resulting recurrence equation (for the sum)—if one exists—`qsumrecursion` returns a list `{rec,cert,f,k,dir}` with five items: The first entry contains the recurrence equation, while the other items enable you to prove the recurrence a posteriori by rational arithmetic.

If we denote by `r` the recurrence `rec` where we substituted the `summ`-function by the input term `f` (with the corresponding shifts in `n`) then the following equation is valid:

```
r = cert*f - sub(k=k-1,cert*f)
```

or

```
r = sub(k=k+1,cert*f) - cert*f
```

if `dir=downward_antidifference` or `dir=upward_antidifference` respectively.

The global variable `qsumrecursion_recrange!*` controls for which recursion orders the procedure `qsumrecursion` looks. It has to be a list with two entries, the first one representing the lowest and the second one the highest order of a recursion to search for. By default it is set to `{1,5}`.

9 Messages

The following messages may occur:

- If your call to `qgosper` or `qsumrecursion` reveals some incorrect syntax, e. g. wrong number of arguments or wrong type you may receive the following messages:

  ```
  ***** Wrong number of arguments.
  ```

  or

  ```
  ***** Wrong type of arguments.
  ```

- If you call `qgosper` with a summand term that is free of the summation variable you get

  ```
  WARNING: Summand is independent of summation variable.
  ***** No q-hypergeometric antidifference exists.
  ```

- If `qgosper` finds no antidifference it returns:

  `***** No q-hypergeometric antidifference exists.`

- If `qsumrecursion` finds no recursion in the specified range it returns:

  `***** Found no recursion. Use higher order.`

  (If you do not pass a range as an argument to `qsumrecursion` the default range in `qsumrecursion_recrange!*` will be used.)

- If the input term passed to `qgosper` (`qsumrecursion`) is not q-hypergeometric wrt. the summation variable — say k — (and the recursion variable) then you get

  `***** Input term is probably not q-hypergeometric.`

  With all the examples we tested, our procedures decided properly whether the input term was q-hypergeometric or not. However, we cannot guarantee in general that `qsimpcomb` *always* returns an expression that *looks* rational in q^k if it actually is.

- If the global variable `qsumrecursion_recrange!*` was assigned an invalid value:

```
Global variable qsumrecursion_recrange!* must be a list
of two positive integers: {lo,hi} with lo<=hi.
***** Invalid value of qsumrecursion_recrange!*
```

References

[1] Askey R. and Wilson, J.: *Some Basic Hypergeometric Orthogonal Polynomials that Generalize Jacobi Polynomials.* Memoirs Amer. Math. Soc. 319, Providence, RI, 1985.

[2] Gasper, G.: *Lecture Notes for an Introductory Minicourse on q-Series.* 1995. To obtain from ftp://unvie6.un.or.at/siam/opsf_new/00index_by_author.html.

[3] Gasper, G. and Rahman, M.: *Basic Hypergeometric Series*, Encyclopedia of Mathematics and its Applications, **35**, (G.-C. Rota, ed.), Cambridge University Press, London and New York, 1990.

[4] Gosper Jr., R. W.: Decision procedure for indefinite hypergeometric summation. Proc. Natl. Acad. Sci. USA **75**, 1978, 40–42.

[5] Knuth, D. E.: *The Art of Computer Programming, Seminumerical Algorithms.* 2nd ed., 1981, Addison-Wesley Publishing Company.

[6] Koepf, W.: Algorithms for m-fold hypergeometric summation. Journal of Symbolic Computation **20**, 1995, 399–417.

[7] Koepf, W.: REDUCE package for indefinite and definite summation. SIGSAM Bulletin **29**, 1995, 14–30.

[8] Koornwinder, T. H.: On Zeilberger's algorithm and its q-analogue: a rigorous description. J. of Comput. and Appl. Math. **48**, 1993, 91–111.

[9] Koekoek, R. und Swarttouw, R.F.: *The Askey-scheme of Hypergeometric Orthogonal Polynomials and its q-analogue.* Report 94–05, Technische Universiteit Delft, Faculty of Technical Mathematics and Informatics, Delft, 1994.

[10] Paule, P. und Riese, A.: A Mathematica q-analogue of Zeilberger's algorithm based on an algebraically motivated approach to q-hypergeometric telescoping. Fields Proceedings of the Workshop 'Special Functions, q-Series and Related Topics', organized by the Fields Institute for Research in Mathematical Sciences at Univerisity College, 12-23 June 1995, Toronto, Ontario,179–210.

[11] Zeilberger, D.: A fast algorithm for proving terminating hypergeometric identities. Discrete Math. **80**, 1990, 207–211.

[12] Zeilberger, D.: The method of creative telescoping. J. Symbolic Computation **11**, 1991, 195–204.