SYMBOLIC COMPUTATION OF SUMS WITH MACSYMA

WOLFRAM KOEPF
Fachbereich Mathematik, Freie Universität Berlin
Arnimallee 3, W-1000 Berlin 33

ABSTRACT

In this lecture we give an introduction into the symbolic algebra systems MACSYMA. MACSYMA is a programming language with the capability to do symbolic calculations rather than only numerical ones. In a further section we show some of the difficulties arising by the use of symbolic algebra systems considering as an example MACSYMA commands associated with summation.

1. General Capabilities of Symbolic Algebra Systems

In this section we present an interactive session showing some of the capabilities of MACSYMA[4]. The lines numbered `(C1)`, `(C2)`, etc., are the input lines which we typed into the computer, whereas the lines `(D1)`, `(D2)`, etc., contain MACSYMA's response. We start our session with the statements

```
(C1) writefile("primer.out");

(D1)                    primer.out
(C2) showtime:true$

Time= 0 msecs
```

In the first line, we ask MACSYMA to write our session into file `primer.out` which enables us to present the results in this paper. If we finish any statement by a semicolon, the corresponding `D` line with the calculated result is shown on the screen, whereas the output is suppressed if we finish our statement by a dollar sign. The result, however, is calculated anyway, and may be referred to by its line number. In line `(C2)` we assign the value `true` to the global variable `showtime` with the effect that for each further computation the calculation time is shown in milliseconds. MACSYMA calculates integers and rational numbers with an arbitrary precision, e.g.

```
(C3) 40!;

Time= 16 msecs
(D3)            815915283247897734345611269596115894272000000000
```

```
(C4) 1024/%;

Time= 0 msecs
                                      1
(D4)          ------------------------------------------
              79679226879677513119688600546495692800000000
```

By `%` one refers to the result of the last line. You see that fractions automatically are represented in lowest terms. There is also an arbitrary precision real arithmetic available. `%pi` is the MACSYMA name for the circular constant π.

```
(C5) bfloat(%pi);

Time= 0 msecs
(D5)                        3.141592653589793b0
(C6) fpprec:40$

Time= 83 msecs
(C7) bfloat(%pi);

Time= 0 msecs
(D7)            3.14159265358979323846264338327950288419 7b0
```

By the setting `fpprec:40$` we set the precision to 40 significant digits.

We can work with polynomials. The MACSYMA command **expand** expands a polynomial expression, whereas the **factor** command succeeds in factorizing any polynomial if the factorization does not contain algebraic expressions. Note that the factorization of the next example obviously needs more time, but in spite of the complexity is quickly done.

```
(C8) f:expand(product(product(j*x-y^k,k,1,3),j,1,2));

Time= 516 msecs
     12        11      2  10        10      2  9        9      3  8       2  8
(D8) Y   - 3 X Y   + 2 X  Y   - 3 X Y   + 9 X  Y  - 3 X Y  - 6 X  Y  + 11 X  Y

       3  7      2  7      4  6       3  6      2  6       4  5       3  5
 - 12 X  Y  + 9 X  Y  + 4 X  Y  - 27 X  Y  + 2 X  Y  + 18 X  Y  - 12 X  Y

       4  4      3  4       5  3       4  3       5  2      4  2       5       6
 + 22 X  Y  - 6 X  Y  - 12 X  Y  + 18 X  Y  - 12 X  Y  + 4 X  Y  - 12 X  Y + 8 X
(C9) factor(f);

Time= 8266 msecs
                               2          2        3          3
(D9)       (Y - 2 X) (Y - X) (Y  - 2 X) (Y  - X) (Y  - 2 X) (Y  - X)
```

The **product** function is self-explaining. Note that MACSYMA (in standard setting) is not case-sensitive, and its output uses upper case variable names.

We can also work with rational expressions, factoring them and calculating their partial fraction decompositions.

```
(C10) g:factor((6+21*x+21*x^2+6*x^3)/(-2*x-4*x^2+6*x^3));

Time= 316 msecs
                          3 (X + 1) (X + 2) (2 X + 1)
(D10)                     ---------------------------
                             2 (X - 1) X (3 X + 1)
(C11) partfrac(g,x);

Time= 216 msecs
                              5        3      27
(D11)                    ----------- - - + --------- + 1
                         4 (3 X + 1)   X   4 (X - 1)
```

As there are algorithms to do so, one can solve linear systems of equations, as well as polynomial equations of order at most 4. In many cases, however, this results in long, unreadable output, which is almost useless. This is not the case for the example

```
(C12) solve(x^3+a*x^2-a*x = 1,x);

Time= 150 msecs
                  2                                  2
             SQRT(A  + 2 A - 3) + A + 1         SQRT(A  + 2 A - 3) - A - 1
(D12) [X = - --------------------------, X = --------------------------, X = 1]
                        2                                  2
```

Note that again symbolic solutions are available.

Besides the algebraic capabilities there are also analytic ones. One can calculate limits and derivatives.

```
(C13) limit((sqrt(1+x)-sqrt(1-x))/x,x,0);

Time= 466 msecs
(D13)                                 1
(C14) diff(((1+x)/(1-x))^n,x);

Time= 166 msecs
                                 X + 1 N   X + 1       1
                      N (1 - X) (-----)  (-------- + -----)
                                 1 - X           2   1 - X
                                           (1 - X)
(D14)                 ------------------------------------
                                      X + 1
(C15) ratsimp(%);

Time= 200 msecs
```

```
                                        X + 1 N
                                  2 N (- -----)
                                        X - 1
(D15)                           - --------------
                                       2
                                      X  - 1
```

Note that the results not always are written in simplest form as there is no general decision procedure to decide which of several equivalent representations is the simplest one. Further the calculation time is minimized if the results are not simplified. It is up to the user to take care of simplifications. In the above example we used the **ratsimp** command to bring the rational result in the usual numerator/denominator form.

One can calculate Taylor polynomials. The following is an example with which the author was able to disprove a conjecture of Robertson[3]. Robertson conjectured that the function considered has only nonnegative Taylor coefficients. By `sqrt` and `exp` the square root and exponential functions are denoted.

```
(C16) taylor(sqrt((exp(x)-1)/x),x,0,14);

Time= 1033 msecs
                     2    3        4      5         6           7
              X   5 X    X     79 X    3 X      71 X       113 X
(D16)/T/ 1 + - + ---- + --- + ----- + ----- + -------- + ---------
              4    96    128   92160   40960   12386304   247726080

           8           9              10             11
     3053 X           X            17 X           19 X
 + ------------ + ----------- + ------------ + ------------
   118908518400   22649241600   930128855040   744103084032

               12                      13                     14
       935917 X              20287103 X              2452337 X
 + ------------------- - -------------------- - ---------------------- + . . .
   1218840851644416000   43878270659198976000   210615699164155084800
```

One of the highlights of symbolic algebra systems is integration. Whereas people must use heuristic techniques for integration, there is an algorithmic procedure by Risch[5] available to decide if an integral is an elementary representable function, and if yes, to find a representation which can be implemented into a symbolic algebra system. MACSYMA is very successful with integration, and the following are only simple examples of its capabilities.

```
(C17) integrate(exp(-x^2),x,0,inf);

Time= 3516 msecs
                                    SQRT(%PI)
(D17)                               ---------
                                        2
```

```
(C18) integrate(exp(x)*sin(x)*x^3,x);

Time= 1633 msecs
             3              X              3      2          X
           (X  - 3 X + 3) %E  SIN(X) + (- X  + 3 X  - 3 X) %E  COS(X)
(D18)      ------------------------------------------------------------
                                      2
```

Last but not least MACSYMA's capabilities to solve ordinary differential equations are quite astonishing, too. Here are some simple examples.

```
(C19) ode(n*r(x)*diff(y(x),x)-diff(r(x),x)*y(x)=0,y(x),x);

/usr/local/lib/macsyma_417/ode/ode.o being loaded.
/usr/local/lib/macsyma_417/ode/odeaux.o being loaded.
/usr/local/lib/macsyma_417/ode/ode2.o being loaded.
Time= 1450 msecs
                                          LOG(R(X))
                                          ---------
                                             N
(D19)                          Y(X) = %C %E
(C20) radcan(%);

Time= 150 msecs
                                            1/N
(D20)                            Y(X) = %C R(X)
(C21) ode(diff(diff(y(x),x),x)-2*diff(y(x),x)+y(x)=0,y(x),x);

Time= 166 msecs
                                                 X
(D21)                         Y(X) = (%K2 X + %K1) %E
```

Again we requested the simplification of one of the results, this time by the **radcan** command which simplifies expressions involving `exp` and `log` terms.

You can see that the capabilities of MACSYMA are broad even though we only mentioned few of the techniques available. However, things are not always that easy as the examples of this sections seemed to promise. The following price has to be paid for so much convenience: One must study the language carefully. This will be illustrated in the next section.

2. Working with Sums

In this section we show some of the difficulties that may arise by the use of symbolic algebra systems. As an example we consider the MACSYMA commands associated with summation.

The MACSYMA command **sum***(expression, variable, lo, hi)* is designated to do two entirely different things. First it enables us to treat with formal manipulation of

sums like the change of the summation index, and the combination of formal sums. Connected with this feature are the MACSYMA functions **changevar**, **intosum**, and **sumcontract**. On the other hand, **sum** has some capabilities to give closed form solutions to certain sums, i.e. it can evaluate specific sums. For the reason to have these two different types of behavior, **sum** is not a normal procedure like the **integrate** command, e.g., but is a special form. The behavior of **sum** as that of special forms in MACSYMA, in general, is mostly determined by its evaluation mechanism.

Let us first give a particularly astonishing example of the behavior of the **sum** command. Suppose we want to evaluate $\sum_{k=1}^{10} k$, and therefore produce the following MACSYMA code:

```
(C1) f:k;

(D1)                                  K
(C2) sum(f,k,1,10);

(D2)                                 10 K
```

What happens here? How can the result depend on the variable k that was only intended to be an index variable? The answer lies in the way in which order expressions are evaluated by MACSYMA. What we had in mind, was: First evaluate the arguments, especially the first argument, and then sum the corresponding expression. What MACSYMA makes, is something different. First MACSYMA recognizes that the sum is of a special type; as the difference of the upper and the lower index for the summation evaluates to a finite positive integer, we have a finite sum. In these cases the sum immediately is transformed to its finite counterpart, i.e. in our case to $f+f+f+f+f+f+f+f+f+f$. Then finally evaluation occurs, and f is replaced by its value k, that was defined in line `(C1)`; internal simplification yields `(D2)`. In the same way e.g. the (nonsense) expression

```
(C3) sum(f,k,f,f+10);

(D3)                                 11 K
```

is handled. Note that this expression does not even have a well-defined counterpart ($\sum_{k=k}^{k+10} k$?) if evaluation should occur before writing the sum out.

If we really want to have the first argument evaluated, we must tell this explicitly to MACSYMA using the **ev** command,

```
(C4) sum(ev(f),k,1,10);

(D4)                                  55
```

which does the job required.

If the difference between the upper and lower index is not a positive integer, then the behavior of the **sum** command may be assumed to be similar to the **integrate** command. This is not the case either. When using the command **integrate**, it will immediately invoke the integration package and try to integrate the corresponding expression. Only if this fails, the noun form is returned. The default return value of the **sum** command, however, is it's noun form. As mentioned earlier, the philosophy behind this behavior is to be able to transform sums formally. The valid transformations on sums will be considered in more detail in another lecture.

Here is a typical example of the difference between the **integrate** and the **sum** commands. Note that the noun form of a procedure is returned if we precede the command name by '.

```
(C5) integrate(f,k,1,n);

                                    2
                                   N    1
(D5)                               -- - -
                                   2    2
(C6) 'integrate(f,k,1,n);

                                   N
                                  /
                                  [
(D6)                              I  K dK
                                  ]
                                  /
                                   1
(C7) sum(f,k,1,n);

                                   N
                                  ====
                                  \
(D7)                               >    K
                                  /
                                  ====
                                  K = 1
```

By default, the option **simpsum** has the value **false**. Setting it to **true** causes simplification of the sum under investigation.

```
(C8) simpsum:true$

(C9) 'sum(f,k,1,n);

                                    2
                                   N  + N
(D9)                               ------
                                     2
```

As you see, if **simpsum** has value `true`, even the noun form of **sum** is evaluated, if the result is available.

Now we set **simpsum** back to false to study the behavior of **sum** with different kinds of arguments to learn more about which type of evaluation occurs. If the difference of the upper and lower bounds is an integer, **sum** always simplifies as we already saw. Here the behavior differs again if the noun form is called.

```
(C10) simpsum:false$

(C11) sum(a[k]*x^k,k,0,10);

          10     9      8      7      6      5      4      3      2
(D11) A  X  + A X  + A X  + A X  + A X  + A X  + A X  + A X  + A X  + A X + A
       10      9      8      7      6      5      4      3      2      1     0

(C12) 'sum(a[k]*x^k,k,0,10);

                                    10
                                   ====
                                   \          K
(D12)                               >    A  X
                                   /      K
                                   ====
                                   K = 0
```

If the difference between the upper and lower bounds is not a positive integer, always the noun form is replied,

```
(C13) sum(a[k]*x^k,k,0,inf);

                                   INF
                                   ====
                                   \          K
(D13)                               >    A  X
                                   /      K
                                   ====
                                   K = 0
```

but calling the noun form is faster as MACSYMA does not try to simplify the expression.

Now we show how arrays (subscripted functions) and functions that occur as input in sums are evaluated. If we give $a[n]$ a value by the **define** command, this produces a pointer to its formula for further evaluations, not more, and so the argument of the following sum is not evaluated.

```
(C14) define(a[k],k)$

(C15) expr1:sum(a[k]*x^k,k,0,n);
```

```
                                  N
                                 ====
                                 \          K
(D15)                             >     A  X
                                 /       K
                                 ====
                                 K = 0
```

If MACSYMA calculates $a[j]$ for a certain index j, just that particular value $a[j]$ will be stored in memory for further evaluation, but not $a[k]$ for any other index k. Which particular values are stored in memory, can be requested by the command **arrayinfo**.

```
(C16) arrayinfo(a);

(D16)                              [HASHED, 1]
(C17) a[j]-a[k-1]$

(C18) arrayinfo(a);

(D18)                       [HASHED, 1, [J], [K - 1]]
(C19) sum(a[k]*x^k,k,0,n);

                                  N
                                 ====
                                 \          K
(D19)                             >     A  X
                                 /       K
                                 ====
                                 K = 0
(C20) a[k]/a[n+1]$

(C21) arrayinfo(a);

(D21)               [HASHED, 1, [J], [K - 1], [K], [N + 1]]
(C22) sum(a[k]*x^k,k,0,n);

                                   N
                                  ====
                                  \         K
(D22)                              >     K X
                                  /
                                  ====
                                  K = 0
```

Line `(D16)` tells that `a` is a hashed array of dimension 1 with no precalculated values. Calculating $a[k]$ for the index k in line `(C20)` stores that value in memory so that the argument of the sum now is evaluated. The same effect occurs if we substitute the **sum** commands by **ev** commands. Note that `expr1` was defined in line `(C15)`.

```
(C23) ev(expr1);
```

```
                                  N
                                 ====
                                 \         K
(D23)                             >     K X
                                 /
                                 ====
                                 K = 0
```

If **simpsum** has value **true**, then further simplification may occur.

```
(C24) simpsum:true$

(C24) x:1$

(C25) ev(expr1);

                                   2
                                  N  + N
(D25)                             ------
                                    2
```

The fact that **sum** only simplifies its first argument if it is stored in memory has as a result that an expression involving functions is only simplified in the case that the difference of its upper and lower bound is a positive integer, as in that case an ordinary sum is substituted for the formal sum.

```
(C26) expr2:'sum(a(k),k,1,n)$

(C27) define(a(k),k)$

(C28) ev(expr2);

                                  N
                                 ====
                                 \
(D28)                             >     A(K)
                                 /
                                 ====
                                 K = 1
```

With one of the following constructions it is possible to get this expression evaluated, too.

```
(C29) apply('sum,[a(k),k,1,n]);

                                   2
                                  N  + N
(D29)                             ------
                                    2
```

```
(C30) sum(''(a(k)),k,1,n);
```

$$\text{(D30)}\qquad \frac{N^2+N}{2}$$

On the other hand, soon we will introduce the **nusum** command that resolves all the problems that are connected with **sum**, and makes the above constructions obsolete.

3. The Computation of Sums

Before doing so we want to mention that the **sum** function with **simpsum** option `true` is able to give closed form solutions in the case of infinite sums of the form

$$\sum_{n=1}^{\infty} \frac{1}{n^{2k}}$$

for fixed $k \in \mathbb{N}$, e.g.

```
(C31) sum(1/k^(30),k,1,inf);
```

$$\text{(D31)}\qquad \frac{6892673020804\ \%PI^{30}}{566087880466908267407001562 5}$$

or in the case of the sums $\sum_{k=0}^{n} k^m$ for fixed $m \in \mathbb{N}$, e.g.

```
(C32) sum(k^10,k,0,n);
```

$$\text{(D32)}\qquad \frac{6\,N^{11}+33\,N^{10}+55\,N^{9}-66\,N^{7}+66\,N^{5}-33\,N^{3}+5\,N}{66}$$

and finally in certain other cases as e.g.

```
(C33) sum(binomial(n,k),k,0,n);
```

$$\text{(D33)}\qquad 2^N$$

On the other hand, there is no way to decide in advance for which arguments the **sum** function will find a closed form solution. For example the similar example is not solved

```
(C34) sum((-1)^k*binomial(n,k),k,0,n);

                          N
                         ====
                         \          K
(D34)                     >    (- 1)  BINOMIAL(N, K)
                         /
                         ====
                         K = 0
```

For the purpose of getting closed form solutions of sums, there is the more powerful function **nusum** that is an implementation of the so-called *Gosper algorithm*[2,1]. This algorithm is successful in all cases where the closed form solution $F_n := \sum_{k=k_0}^{n} a_k$ of the indefinite summation, i.e. the summation with variable upper bound, has rational ratio F_{n+1}/F_n. This is the case, e.g., if F_n is expressible in terms of factorials and rational functions.

As the **nusum** command is a function rather than a special form, none of the problems above occurs here. Its behavior depends on the setting of the option **simpsum**, too. We highly recommend the following usage of **sum** and **nusum**:

1. For formal manipulations of sums use **sum** with the setting `simpsum: false`, calling the noun form,

2. for closed form solutions of sums use **nusum** with the setting `simpsum: true`.

All closed form results that had been found by the **sum** command in the above MACSYMA session are found by the Gosper algorithm, and so by **nusum**, as well. But we want to mention that sometimes the Gosper algorithm is pretty much slower. This is, e.g., the case in the example on line `(C32)`.

On the other hand the Gosper algorithm is able to handle, e.g., the following examples that are not done by the **sum** function.

```
(C35) nusum((-1)^k*binomial(n,k),k,0,n);

/usr/local/lib/macsyma_417/share/nusum.o being loaded.
(D35)                                0

(C36) nusum(k*k!,k,0,n);

(D36)                           (N + 1)! - 1
```

Sometimes it is necessary to load an even stronger package, **nusum1**. This is, e.g., the case if one of the lower or upper bounds equals $\pm\infty$.

```
(C37) nusum(a[k]*y^k,k,0,inf);
```

```
                     INF + 1
                    Y        (INF Y - INF - 1)      Y
(D37)               ------------------------- + --------
                                  2                   2
                           (Y - 1)             (Y - 1)
(C38) load(nusum1)$

/usr/local/lib/macsyma_417/share/nusum1.o being loaded.
(C39) nusum(a[k]*y^k,k,0,inf);

Is   ABS(Y) - 1   positive, negative, or zero?
n;
Is   Y   positive, negative, or zero?
p;
Is   Y - 1   positive or negative?
n;
                                      Y
(D37)                               --------
                                           2
                                    (Y - 1)
```

As you see, without **nusum1**, **nusum** treats ∞ as an ordinary variable and does not carry out the limit.

If MACSYMA needs some information for its calculations, it asks the user. Here we answered that y should be in the interval $(0, 1)$.

With **nusum1** loaded, it is best to use the **closedform** command. It can handle sums, and products, and uses all available techniques to solve finite, symbolic indefinite, and infinite sums and products. So instead of the above statement we can also use

```
(C40) assume(y>0,y<1)$

(C41) closedform(sum(a[k]*y^k,k,0,inf));

                                      Y
(D41)                           ------------
                                 2
                                Y  - 2 Y + 1
```

The **assume** command tells MACSYMA in advance that $y \in (0, 1)$, so it will not have to ask any more.

Quite complicated infinite sums can be handled by the **closedform** procedure:

```
(C42) closedform(sum(k^2/(k^2+a^2)^3,k,1,inf));

          3                 2             2     2
       %PI  COTH(%PI A) CSCH (%PI A)   %PI  CSCH (%PI A)   %PI COTH(%PI A)
(D42) - ---------------------------- + ----------------- + ---------------
                    8 A                          2                3
                                              16 A             16 A
```

```
(C43) trigsimp(trigreduce(trigexpand(%)));

/usr/local/lib/macsyma_417/share/trigsimp.o being loaded.
                          2            2                      3  2
      %PI COSH(%PI A) SINH (%PI A) + %PI  A SINH(%PI A) - 2 %PI  A  COSH(%PI A)
(D43) ------------------------------------------------------------------------
                                    3     3
                                16 A  SINH (%PI A)
```

The **trigsimp**, **trigreduce** and **trigexpand** commands simplify trigonometric expressions.

We list some more examples for the use of the **sum**, **nusum** and **closedform** commands.

Example 1. Calculate $\sum_{k=0}^{n} k^9$ using the **sum** and the **nusum** commands, and compare the calculation times.

The following MACSYMA statements produce the result.

```
simpsum:true$
showtime:true$
sum(k^9,k,0,n);

Time= 233 msecs
                    10       9       8       6       4      2
                  2 N   + 10 N  + 15 N  - 14 N  + 10 N  - 3 N
                  ------------------------------------------
                                      20
nusum(k^9,k,0,n);

Time= 25283 msecs
               2        2   2              4      3    2
              N  (N + 1)  (N  + N - 1) (2 N  + 4 N  - N  - 3 N + 3)
              ---------------------------------------------------
                                    20

factor(sum(k^9,k,0,n));

Time= 666 msecs
               2        2   2              4      3    2
              N  (N + 1)  (N  + N - 1) (2 N  + 4 N  - N  - 3 N + 3)
              ---------------------------------------------------
                                    20
```

In this example the Gosper algorithm is much slower than the built-in solver used by the **sum** command.

Example 2. Calculate the following sums.

(a) $\sum_{k=0}^{\infty} k^2 y^k \ (0<y<1)$, (b) $\sum_{k=1}^{\infty} \frac{1}{k^6}$, (c) $\sum_{k=1}^{n} (k!)^2 (k^2+2k)$,

(d) $\displaystyle\sum_{k=0}^{n}\left(\sum_{j=0}^{4}k^j\right)$, (e) $\displaystyle\sum_{k=1}^{\infty}\frac{(-1)^k\,k^2}{(k^2+a^2)^3}$, (f) $\displaystyle\sum_{k=1}^{\infty}\frac{k^2}{(k^2+a^2)^4}$.

The following MACSYMA commands produce the results.

```
simpsum:true$
load(nusum)$
load(nusum1)$
assume(y>0,y<1)$
nusum(k^2*y^k,k,0,inf);
                                  Y (Y + 1)
                                - ---------
                                         3
                                  (Y - 1)
nusum(1/k^6,k,1,inf);
                                     6
                                  %PI
                                  ----
                                  945
nusum(k!^2*(k^2+2*k),k,1,n);
                                      2
                                (N + 1)!  - 1
nusum(sum(k^j,j,0,4),k,0,n);
                           4       3       2
                (N + 1) (12 N  + 33 N  + 37 N  + 38 N + 60)
                ------------------------------------------
                                  60
closedform(sum((-1)^k*k^2/(k^2+a^2)^3,k,1,inf));

     3     2 %PI A       %PI A            %PI A       2     2 %PI A
  %PI  SECH (-----) TANH(-----)   %PI TANH(-----)   %PI  SECH (-----)
               2           2                2                   2
- ---------------------------- - -------------- + -----------------
             64 A                         3                2
                                       32 A             64 A

          3      %PI A      2 %PI A       2     2 %PI A            %PI A
       %PI  COTH(-----) CSCH (-----)   %PI  CSCH (-----)   %PI COTH(-----)
                 2           2                  2                 2
    - ---------------------------- + ---------------- + --------------
                64 A                          2               3
                                           64 A            32 A
trigsimp(trigreduce(trigexpand(%)));

       2       3 %PI A       2        %PI A        %PI A
((2 %PI  A COSH (-----) - %PI  A COSH(-----)) SINH(-----)
                  2                   2           2

                3  2      4 %PI A          3  2             2 %PI A
 + (2 %PI - 2 %PI  A ) COSH (-----) + (2 %PI  A  - 2 %PI) COSH (-----)
                             2                                  2
```

```
      3  2       3     3 %PI A      3 %PI A
 - %PI  A )/(64 A  COSH (-----) SINH (-----))
                           2            2
```

```
closedform(sum(k^2/(k^2+a^2)^4,k,1,inf));
```

```
     4     2               2                   2     2
  %PI  CSCH (%PI A) (3 CSCH (%PI A) + 2)   %PI  CSCH (%PI A)   %PI COTH(%PI A)
- ---------------------------------------- + ----------------- + ---------------
                      2                                4                   5
                  48 A                              32 A                32 A
```

```
trigsimp(trigreduce(trigexpand(%)));
```

```
                        3                2          4  3      2
(3 %PI COSH(%PI A) SINH (%PI A) + (3 %PI  A - 4 %PI  A ) SINH (%PI A)

                                              4  3       5     4
                                       - 6 %PI  A )/(96 A  SINH (%PI A))
```

References

1. R. L. Graham, D. E. Knuth ans O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science* (Addison-Wesley Publ. Co., Reading, Massachusetts, 1988), § 5.7.

2. R. W. Gosper Jr., *Decision procedure for indefinite hypergeometric summation* (Proc. Natl. Acad. Sci. USA **75**, 1978), p. 40–42.

3. W. Koepf, *On two conjectures of M. S. Robertson* (Complex Variables **16**, 1991), p. 127–130.

4. MACSYMA: *Reference Manual, Version 13* (Symbolics, USA, 1988).

5. R. H. Risch, *The problem of integration in finite terms* (Trans. Amer. Math. Soc. **139**, 1969), p. 167–189.