Estimate a double sum.

We can evaluate double sums by using the sum key with a function that itself calls the sum function. Here we make a table (using the sum key as a loop control) of partial sums and then extrapolate using the Aitken acceleration formula. So this example uses recursive calls to the sum function — a sum within a sum within a sum (a bit like the movie Inception).

The sum is $$S = \sum_{k=1}^{\infty} \sum_{j=1}^{\infty} \frac{1}{j^3 + k^2}$$

We will do partial sums with j = 1, 2, 3, ..., 2^n, and k = 1, 2, 3, ..., 2^n:

$$S_n = \sum_{k=1}^{2^n} \sum_{j=1}^{2^n} \frac{1}{j^3 + k^2}$$

Convergence will be slow, since for a given n the total number of terms in the partial sum will be $2^n\ 2^n = 4^n$. For $n = 11$ we will be adding only $2^{11} = 2048$ terms in each individual j sum, so there will not be much accuracy. But the total number of terms in S_{11} is 4^{11} = 4,194,304, so $n = 11$ will take a long time to compute.

Choosing a geometric sequence, $2^1, 2^2, 2^3, \ldots$ for the number of terms in the j and k sums is not the only possible option. Sometimes an arithmetic sequence like $100, 200, 300, \ldots$ might make the subsequent Aitken acceleration more effective, depending on the problem we are trying to solve. Here, trying both options shows the geometric sequence is better.

Compute $S_1, S_2, \ldots, S_{11}$ and store the 11 values in registers so Aitken acceleration can then be used to try to increase the accuracy of our estimate for S.

f0: 1, func, 3, y^x, 2, rcl, x^2, +, 1/x

f1: 1, func, 2, sto, 1, enter, 2, enter, 3, rcl, y^x, 7, func, 1, enter, 0, sum

f2: 1, func, 3, sto, 1, enter, 2, enter, 3, rcl, y^x, 7, func, 1, enter, 1, sum, 3, rcl, 20, +, sto

f3: 7, func, 1, enter, 11, enter, 1, enter, 2, sum

f0 takes its input, j, cubes it, gets register 2, k, squares it, adds, and inverts: $1/(j^3 + k^2)$

f1 stores its input, k, in register 2, then sets up the arguments to the sum key:
j from 1 to 2^n by steps of 1, sum function 0. (n is in register 3)

f2 stores its input, n, in register 3, then sets up the arguments to the sum key:
k from 1 to 2^n by steps of 1, sum function 1.
After the sum, store the result in register $20 + n$.

f3 uses the sum key as a loop for $n = 1$ to 11. This calls f2 to compute S_n and save the result.

After running f3, the 11 values $S_1, S_2, \ldots, S_{11}$ will be stored in registers $21, 22, \ldots, 31$.

The "enter" key is used to indicate the end of each number whenever the next item will also be a number. It is not needed when the next item is a command, like "rcl" or "sum". In the function definition, commas separate the numbers, so an "enter" between two numbers is not part of the function definition, and f3 is shown as

f3: 1,11,1,2,sum

f3 will make over 4 million function calls, so we will extend the maximum time allowed to 10 minutes.

600, time, 3, f_n

Looking at the 11 partial sums in registers $21, 22, \ldots, 31$ shows

0.894444444444444444444444444444
1.329053239770864281746232693276
1.735754135770537955696670108364
...
3.055057800606103366971821393841
3.148514381991497694818720532139
3.222726712017357532830526790014

All that work has not given us much accuracy. We might be able to say S is between 3 and 4, but we can't be confident of the second significant digit.

Next, let's try some Aitken acceleration on this list of numbers. Define a general version of Aitken's formula that will take as input a list of n numbers beginning with register a, and will return as output a list of $n-2$ numbers beginning with register b.

f4: 1, func, 13, rcl, 12, rcl, −, x^2, 13, rcl, 12, rcl, 2, *, −, 11, rcl, +, /, chs, 13, rcl, +

f5: 14, sto, rcl, 11, sto, 14, rcl, 1, +, rcl, 12, sto, 14, rcl, 2, +, rcl, 13, sto, 4, f_n

f6: 15, sto, 17, rcl, +, 5, f_n, 15, rcl, 18, rcl, +, sto

f7: 7, func, 18, sto, roll, 17, sto, roll, 16, sto, 0, x↔y, 3, −, 1, enter, 6, sum

f4 computes the basic Aitken formula on the 3 numbers in registers 11, 12, 13.

$$x_{n+2} - \frac{(x_{n+2} - x_{n+1})^2}{x_{n+2} - 2\,x_{n+1} + x_n}\ , \quad \text{where } x_n \text{ is register 11, } x_{n+1} \text{ is register 12, } x_{n+2} \text{ is register 13.}$$

f5 uses its input as the register number of x_n and applies Aitken's formula to the next 3 numbers. It stores its input in register 14, then moves the 3 numbers starting with the register indexed by register 14 to registers 11, 12, 13 and calls f4 to get the Aitken value.

f6 is called by the general Aitken function. Its stores its input, k, in register 15, then computes the starting location, $a + k$, of the 3 registers used in f5. After calling f5, the result is stored in location $b + k$.

f7 is the general Aitken function. Its 3 inputs are n, a, b, which it stores in registers 16, 17, 18. It uses the sum function to loop $k = 0, 1, \ldots, n-3$ with sum calling f6.

Now use f7 to apply Aitken to the 11 values starting in register 21 and put the results into 9 registers beginning with register 41.

11, enter, 21, enter, 41, enter, 7, f_n

Looking at those 9 values in registers 41, 42, ..., 49 shows:

7.662592873088799125611436539722
4.193364341327199575443003040526
3.708881639676425044742046345156
...
3.511988108217907602269316966406
3.509774005321595737061042687821
3.508914510387517721511405139840

The last two values now agree to 3 significant digits.

We can try another f7 to apply Aitken to the 9 values starting in register 41 and put the results into 7 registers beginning with register 51.

9, enter, 41, enter, 51, enter, 7, f_n

The 7 values in registers 51, 52, ..., 57 are:

3.630240609063068592736599145039
3.527132433023607388485558816581
3.512030054443020941743352359530
3.509160443996320162745622878039
3.508542835384436018043021278614
3.508402094736892586850405298384
3.508369163274329674961947510632

The last two values now agree to 5 significant digits, so try another f7 to apply Aitken to the 7 values starting in register 51 and put the results into 5 registers beginning with register 61.

7, enter, 51, enter, 61, enter, 7, f_n

The 5 values in registers 61, 62, ..., 65 are:

3.509438386105512389688328033866
3.508487279588584548193835675364
3.508373457015566030746064564328
3.508360557180688052385133156620
3.508359104008723238657482983911

The last two values agree to 6 significant digits, so try another f7 to apply Aitken to the 5 values starting in register 61 and put the results into 3 registers beginning with register 71.

5, enter, 61, enter, 71, enter, 7, f_n

The 3 values in registers 71, 72, 73 are:

3.508357983676774813228286576707

3.508358908337791544958639800815

3.508358919526243467898660445997

The last two values agree to 8 significant digits, so try another f7 to apply Aitken to the 3 values starting in register 71 and put the result into register 81.

3, enter, 71, enter, 81, enter, 7, f_n

The value in register 81 is:

3.508358919663282566011194833986

The last two values above and this one all agree to about 8 significant digits: 3.5083589

There is no guarantee when doing extrapolation, but the general agreement of all the numbers from the various Aitken steps and the gradual increase in apparent accuracy makes it seem likely that we have found the value of S to about 8 digits.

The terms of most double sums will be more complicated than this example, forcing us to use summation plus extrapolation as we did above. But sometimes, as in this case, the terms have a simple enough form that a computer algebra system can give a closed form for the double sum itself, or one of the single sums in terms of j or k.

If we change the order of summation to put the k sum on the inside,

$$S = \sum_{j=1}^{\infty} \left(\sum_{k=1}^{\infty} \frac{1}{j^3 + k^2} \right)$$

Wolfram Alpha, available on the internet, and Mathematica both give a closed form for the inner sum. Either will accept this form of input for the k series: Sum[1/(j^3 + k^2), {k, 1, Infinity}]

They return a formula for the inner sum:

$$S(j) = \sum_{k=1}^{\infty} \frac{1}{j^3 + k^2} = \frac{\pi\, j^{3/2}\, \coth(\pi\, j^{3/2}) - 1}{2\, j^3}$$

Here $\coth(x)$ is the hyperbolic cotangent function, which we can get using hyperbolic tangent on screen 1: $\coth(x) = 1/\tanh(x)$.

This reduces the problem from a double sum to a single sum in terms of j, so that should make the calculation much simpler.

There is also a closed form for the sum over j, giving a single sum in terms of k, but each of these k terms involves finding the three roots of a cubic polynomial, two of which are complex numbers, and evaluating the polygamma function at those roots. That is probably too slow.

Now that we have a single sum to evaluate with the terms coming from a smooth function when extended to positive real numbers, we can try the Euler-Maclaurin formula with $f(x) = (\pi\, x^{3/2}\, \coth(\pi\, x^{3/2}) - 1)/(2\, x^3)$.

See the Infinite Sums example page for more discussion of Euler-Maclaurin.

$$\sum_{n=1}^{\infty} f(n) \;=\; \sum_{n=1}^{k-1} f(n) \;+\; \sum_{n=k}^{\infty} f(n)$$

$$\approx \sum_{n=1}^{k-1} f(n) \;+\; \int_k^{\infty} f(x)\, dx \;+\; \frac{1}{2}\, f(k) \;-\; \sum_{j=1}^{m} \frac{B_{2j}\; f^{(2j-1)}(k)}{(2j)!}$$

Use k = 100,000 and define f1 to be $f(x)$, and f2 to be the j term of the Bernoulli sum, with k having been stored in register 0.

f1: 1, func, 10, sto, 1.5, y^x, π, *, tanh, 1/x, 10, rcl, 1.5, y^x, *, π, *, 1, -, 10, rcl, 3, y^x, 2, *, /

f2: 2, sto, 6, func, 0, rcl, 2, rcl, 2, *, 1, sto, 1, −, 1, $f^{(n)}$, 3, func, 1, rcl, B_n, *, 1, rcl, x!, /

Evaluate the four terms in the sum approximation. Keep the sum of these terms in register 11. First store k = 100,000 in register 0 then add the first 99,999 terms of the series.

7, func, e5, enter, 0, sto, 1, −, 1, x↔y, 1, enter 1, sum, 11, sto 3.498424279199476112629489332430

Next do the integral for the second term.

6, func, 0, rcl, e9999, enter, 1, $\int_a^b$ 0.009934588240796101234433550670

11, rcl, +, 11, sto 3.508358867440272213863922883100

Not all integrals can be found accurately by the integrate key, so check the estimated error by pressing x↔y. That shows 9.2e−45, so in this case the integral should be ok.

The third term is $f(k)/2$.

0, rcl, 1, f_n, 2, / 0.000000024836470414490253086084

11, rcl, +, 11, sto 3.508358892276742628354175969184

Do the j terms one at a time to show how many digits of agreement we are getting as m increases.

1, enter, 2, f_n −0.000000000000062091175411225633

11, rcl, x↔y, −, 11, sto 3.508358892276804719529587194816

2, enter, 2, f_n 0.00000000000000000000000905496

11, rcl, x↔y, −, 11, sto 3.508358892276804719529586289320

3, enter, 2, f_n −5.335960077657411144796666621290e−35

11, rcl, x↔y, −, 11, sto 3.508358892276804719529586289320

4, enter, 2, f_n 6.503201102457469832720936946 97e−45

11, rcl, x↔y, −, 11, sto 3.508358892276804719529586289320

5, enter, 2, f_n −1.326094613853385578263170854 72e−54

11, rcl, x↔y, −, 11, sto 3.508358892276804719529586289320

At this point, we would have to stop the m sum even if the f2 function value had not been less than 1.0e−50 in magnitude, because the $m = 5$ term uses a ninth derivative of f1 and ten is the highest derivative that the $f^{(n)}$ key can return.

Getting 9.2e−45 as the estimated maximum error in the integral term means that the final two iterations in the sequence of approximations done by the integrate function agreed to about 44 digits. Because the sequence usually converges quickly to the integral value, that normally means the next-to-last approximation was good to 44 digits and the last approximation, the one returned by the integrate key, should be accurate to over 50 digits.

That means it is likely that register 11 now gives the value of our double sum correct to more than 50 digits.