Extrapolation

The extr key on screen 7 does extrapolation for sequences.

It is fairly common when doing a numerical approximation to the solution of some mathematical problem that we get a sequence of approximations getting closer and closer to the answer.

But it often happens that the rate of convergence for this sequence is fairly slow and even after doing lots of computation we don't have very many correct digits in the result.

Extrapolation tries to help by analyzing the sequence. If the rate at which the sequence is converging matches the model used by a particular extrapolation method, that method can produce an estimate for the limit of the sequence that is more accurate than any of the elements of our computed input sequence.

In general, extrapolation works better on alternating sequences, where one term is above the limit, the next term is below, the next one above, etc. The first example below has monotonic convergence, where all the terms are on one side of the limit, but its rate of convergence is simple enough that extrapolation can still work well.

The Calc-50 extrapolation function uses several different extrapolation methods and tries to return the best result, along with an estimate of its accuracy.

The extrapolation function has 4 inputs: j, a, b, n, extr

This says to extrapolate a sequence using function n from a to b.
j indicates which type of extrapolation is to be done:

j = 1 approximates $\displaystyle\sum_{k=a}^{\infty} f_n(k)$

j = 2 approximates $\displaystyle\prod_{k=a}^{\infty} f_n(k)$

j = 3 approximates $\displaystyle\lim_{k\to\infty} f_n(k)$

In each case, the function values $f_n(a), f_n(a+1), f_n(a+2), \ldots, f_n(b)$ will be used to generate a sequence of partial sums, partial products, or function values. Then extrapolation methods will be used to estimate the limit of that sequence as $k \to \infty$.

Example 1. An infinite sum.

$$\sum_{k=0}^{\infty} \binom{2k}{k} \frac{1}{4^k(2k+1)^2}$$

This is the last example on the "Infinite sums" page. Because using the sum key to approximate infinite series requires evaluating $f_n(k)$ at very large k values, this series will cause trouble due to overflow in the binomial coefficient and the 4^k term.

Extrapolation can be used to avoid overflow problems by using only small values of k.

Define f1 to compute the k^{th} term of the series.

f1: 1, sto, 1, func, 2, $*$, 1, +, x^2, 1, rcl, 4, x↔y, y^x, $*$, 1/x, 3, func, 1, rcl, 2, $*$, 1, rcl, cmb, $*$

We are doing a sum, so the first input to the extr key is 1 (called j above). the second input, a, is where the series starts, so a = 0 in this case. How to choose b?

Extrapolation can be done with as few as three elements, but usually more data is better. The tradeoff is that larger b values mean using more data for the extrapolation, so it often gives better accuracy in exchange for the calculation taking longer due to making more function evaluations.

If $f_n(k)$ uses any slow functions like integrate, solve, or ode, then we might start by trying b = 5 or 10. This series should be faster, so maybe b = 25 or 50 is a good place to start.

The extrapolate key returns an estimated relative error in the y-register, along with its approximation for the limit of the sequence in the x-register. We can try several values for b, increasing as we go.

Extrapolate the sum using b = 25.

40, sci, 7, func, 1, enter, 0, enter, 25, enter, 1, extr

1.088793045151801065250236781771501482963e+0

Pressing x↔y shows the estimated error is about 1.9e−20. Try b = 50.

1, enter, 0, enter, 50, enter, 1, extr 1.088793045151801065250344449118806973669e+0

The estimated error is about 7.6e−36. Try b = 100.

1, enter, 0, enter, 100, enter, 1, extr 1.088793045151801065250344449118806973669e+0

The estimated error is about 6.7e−38. Try b = 200.

1, enter, 0, enter, 200, enter, 1, extr 1.088793045151801065250344449118806973669e+0

The estimated error is about 1.0e−47. Try b = 400.

1, enter, 0, enter, 400, enter, 1, extr 1.088793045151801065250344449118806973669e+0

The estimated error is about 1.0e−55.

For this example problem the exact value of the sum is $\pi \ln(\sqrt{2})$, and subtracting this from the value above gives about −1e−56, so extrapolation using the first 401 terms gave over 50 digits correct.

Example 2. An infinite product.

$$\prod_{k=2}^{\infty} \left(\frac{1}{e} \left(\frac{k^2}{k^2-1} \right)^{k^2-1} \right)$$

We might think of applying a logarithm to this product, which would give an infinite sum of logs. However, like the example above, these terms are too complicated for the sum key to do an automatic approximation for the sum to infinity.

When k gets too large, $k^2 - 1$ rounds to the same value as k^2, so the $k^2 - 1$ power returns 1 instead of something close to e. Then the terms in the product no longer appear to be converging to 1 and the product seems to be going to zero. Extrapolation will avoid this problem by keeping k small.

f2(k) computes the k^{th} term in the product.

f2: 1, func, x^2, 2, sto, 1, −, 1/x, 2, rcl, ∗, 2, rcl, 1, −, y^x, 1, e^x, /

This time j = 2 since we are extrapolating a product, and a = 2. Try b = 25, 50, 100, 200, 400 like before.

40, sci, 7, func, 2, enter, 2, enter, 25, enter, 2, extr

Checking the estimated errors after each value for b gives: 8e−14, 6e−37, 2e−39, 3e−50, 6e−54. The last few approximations all agree to 40 digits on the screen:

7.132829689452240195019882372298500821854e-1

The extr key's estimated relative error is 6.2e−54 when b = 400. The true value of this infinite product is $e^{3/2}/(2\pi)$, and subtracting that from the last approximation gives 3.3e−56.

So in this case extrapolation gave over 50 digit accuracy and was fairly quick and easy compared to using logs to get a sum and then generating a messy asymptotic approximation that the sum key could use for large k, in a way similar to what we did for the last example in the "Infinite sums" page.

Example 3. A convergent sequence.

$$S_k = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{k} - \ln(k)$$

This sequence converges to Euler's constant $\gamma = 0.57721566490153286060651209008 2...$, which is found on screen 3, so we can check the accuracy of our results.

Define a function to generate this sequence. f3(k) will compute S_k by summing f4(i) = $1/i$ from $i = 1$ to k, then subtracting $\ln(k)$.

f3: 7, func, 3, sto, 1, x↔y, 1, enter, 4, sum, 3, rcl, 1, func, ln, −

f4: 1, func, 1/x

This is a sequence extrapolation, not an infinite sum or product, so j = 3. Starting with S_1 means a = 1. This time we tried b = 25, 50, 100, and 200, stopping when the estimated error suggested that we had about 50 digits correct.

40, sci, 7, func, 3, enter, 1, enter, 25, enter, 3, extr

The estimated errors after each value for b are: 1e−17, 2e−36, 7e−40, 4e−49. The last few approximations all agreed to 40 digits on the screen:

5.772156649015328606065120900824024310422e-1

The extr key's estimated relative error is 4.5e−49 when b = 200. The true value of the limit of this sequence is γ, and subtracting that from the last approximation gives the true error as 3.9e−57.

Again extrapolation gave over 50 digit accuracy.

We were being a bit lazy with these "true error" calculations. The estimated error is a relative error, defined by $\left|\,(\text{true} - \text{estimated})/\text{true}\,\right|$. So we should have divided by Euler's constant to get a true relative error of 6.8e−57.

Because the true answer is 1 to within a factor of ten, it didn't make much difference whether we checked the absolute error $\left|\,(\text{true} - \text{estimated})\,\right|$ or the relative error. It is the "e−57" that we are interested in.

See the e^{-50} Taylor series evaluation in the "Infinite sums" page for a case where the distinction between relative and absolute error is important.

Example 4. An infinite limit.

This limit comes from the College Mathematics Journal, March 2021.

$$\lim_{n\to\infty} \frac{\binom{n^2-2n-1}{n}}{\binom{n^2}{n}}$$

The two binomial coefficients cause a problem we have seen before. We can't just evaluate the quotient at very large values of n, because any n bigger than 10^9 causes both the top and bottom of the quotient to overflow. Evaluating at $n = 10^8$ turns out to give an approximation to the limit that is accurate to only about 8 significant digits.

The "limit" key on screen 7 is for finite limits of indeterminate expressions, so it does not handle limits to infinity.

Extrapolation is the easiest thing to try first. If that fails we will have to get an asymptotic approximation for the entire quotient, as with example 8 in the "Infinite Sums" page.

f1: 1, func, 1, sto, x^2, 1, rcl, 2, *, −, 1, −, 1, rcl, 3, func, cmb, 1, rcl, enter, *, 1, rcl, cmb, /

Extrapolating with b = 50, 100, and 200 gave an estimated error of 5.3e–49 for the 200 case.

40, fix, 7, func, 3, enter, 1, enter, 200, enter, 1, extr, 2, sto

0.13533528323661269189399949497248440340 76

Assuming we do not recognize this constant (call it x), try to guess a closed form for it. The quickest thing to try is to look at x^2, x^3, $\sqrt{x}$, $x^{1/3}$, x/π, x/π^2, $x*\pi$, $x*\pi^2$, e^x, $\ln(x)$, etc. If we recognize one of those then we have a good conjecture for the exact value of x.

In this case we are lucky and find $\ln(x) = -2$, so it seems that $x = e^{-2}$. Subtracting e^{-2} from our computed x gives 5.796e–60.

What if we do not come across any recognizable numbers after trying various simple functions like those is the list above? x still might have a nice closed form, just not quite nice enough for us to find it easily.

Another thing we can try is the Inverse Symbolic Calculator. There is a copy of that program at

http://wayback.cecm.sfu.ca/projects/ISC/ISCmain.html

There are four methods that can be tried. Sometimes it helps to only use the first 15 or 20 significant digits of our number. Choosing "Simple Lookup" and entering

0.13533528323661269

the first result says

K**n: K is a base constant

$1353352832366126 = (1/e)^n \qquad n = 2$

This agrees that our number could be e^{-2}. The Inverse Symbolic Calculator can often suggest closed forms that are more complicated than anything we might find with a simple search like we tried above.

Example 5. Extrapolation can sometimes even produce useful results from divergent sequences.

To get a problem with a known result, use the Taylor series for the natural logarithm of x, expanded about $x = 1$.

$$\ln(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}\,(x-1)^k}{k} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots$$

In calculus it is shown that this series has a radius of convergence equal to 1, and converges for $0 < x \le 2$. Trying to use $x = 3$ to compute $\ln(3)$ from this formula leads to a divergent series with terms that do not approach zero as $k \to \infty$.

$$(3-1) - \frac{(3-1)^2}{2} + \frac{(3-1)^3}{3} - \frac{(3-1)^4}{4} + \frac{(3-1)^5}{5} - \frac{(3-1)^6}{6} + \cdots =$$

$$2.0000 - 2.0000 + 2.6667 - 4.0000 + 6.4000 - 10.6667 + \cdots$$

Some of the methods tried by the extr function can interpret a divergent sequence as being a convergent one that is in reverse order, so they might still return a useful result. Start with this divergent sequence of partial sums,

$$2.00,\ 0.00,\ 2.67,\ -1.33,\ 5.07,\ -5.60\ \cdots.$$

Taking those terms in reverse order gives what looks like an alternating convergent sequence, so perhaps extrapolation can help. Try it for the $\ln(x)$ series. x will be stored in register 3, and f5 will compute the k^{th} term in the sum, $\text{f5}(k) = (-1)^{k+1}(x-1)^k/k$.

f5: 1, func, 5, sto, 3, rcl, 1, −, 5, rcl, y^x, 5, rcl, /, −1, enter, 5, rcl, 1, +, y^x, ∗

This is a series extrapolation, so j = 1. Starting with S_1 means a = 1.
Again we tried b = 25, 50, 100, and 200.

40, sci, 3, enter, 3, sto, 7, func, 1, enter, 1, enter, 25, enter, 5, extr

The estimated errors after each value for b are: 2e–16, 7e–27, 5e–43, 2e–15.

The error with b = 200 is much larger than the previous two. Checking the 200^{th} term gives a very large value: $f_5(200) = -8.0\text{e+}57$. The divergence of the series may have finally caught up with us.

The b = 100 approximation is

1.0986122886681096913952452369225257046 47e+0

Subtracting ln(3) gives a true error about 8e–44.

It is a bit spooky that we can take sums from this "worthless" divergent series and produce over 40 digits of accuracy in our estimate for ln(3).

Example 6. A sum where extrapolation does not help much.

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

Here $H_k = 1 + 1/2 + 1/3 + \cdots + 1/k$ is the k^{th} harmonic number. If we use the sum key to define H_k and then use the sum key again for the outer k sum, it will be too slow for the outer sum to approximate the infinite sum on k.

As often happens, it will be easiest to replace the outer sum by an extrapolation, so we will try that first. That will not need any really large values of k.

f6(k) computes $(H_k/k)^2$ by summing f7(i) $= 1/i$ from $i = 1$ to k to get H_k.

f6: 6, sto, 7, func, 1, x↔y, 1, enter, 7, sum, 6, rcl, /, 1, func, x^2

f7: 1, x↔y, /

Try extrapolating using b = 100, 200, 400, 800 terms. The first one is:

30, sci, 7, func, 1, enter, 1, enter, 100, enter, 6, extr 4.599873719155561596967189421 03e+0

The estimated errors after each value for b are: 2e–13, 1e–9, 1e–6, 4e–7.

The fact that the estimated errors are not getting smaller as we sample more data looks like a bad sign. We should probably be skeptical of the claim to have about 13 digits correct for the b = 100 case.

The true value of this sum is $17\pi^4/360$, so we can check and find that the actual relative errors in our four cases are 5e–9, 4e–10, 1e–6, 6e–7.

Trying even larger values of b gives smaller estimated errors, but the actual errors do not get smaller.

Recall that for extrapolation to work well, we need for the convergence rate of the sequence to match the mathematical model assumed by one of the extrapolation methods tried by the calculator. Apparently in this case none of the extrapolation models is a good fit for this particular series.

When this happens, the next thing we can try is to find an asymptotic approximation for H_k that will be accurate for large values of k. Then we can use the sum key's approximation for infinite sums. Since harmonic numbers are fairly common, we can just look up the formula (see "Harmonic number" in Wikipedia):

$$H_k \approx \ln(k) + \gamma + \frac{1}{2k} - \sum_{j=1}^{\infty} \frac{B_{2j}}{2j\,k^{2j}} = \ln(k) + \gamma + \frac{1}{2k} - \frac{1}{12k^2} + \frac{1}{120k^4} - \cdots$$

Now this sum can be done in the same way as example 8 from the "Infinite sums" page. Selecting the direct formula for small k and the asymptotic approximation for large k allows the sum key to evaluate S.

For more examples using extrapolation, see the "Prime sum", "Double sum", and "Oscillating integrals" pages.