Lesson 32: Iteration and approximation

```
> restart;
```

A numerical iteration

Recurrence relations are often used in numerical methods. You want to compute some sequence y_n which satisfies a recurrence relation, so you start with known values for y_0 or the first few y_n, and iterate the recurrence formula. This may or may not be a good way to calculate y_n. The main thing that can go wrong is that the inevitable roundoff errors can grow with each iteration until they overwhelm the true solution.

For example, suppose you want to approximate the following sequence of numbers, related to the remainders in the series for *e*:

$$w_n = n!\left(\mathrm{e} - \left(\sum_{k=0}^{n} \frac{1}{k!}\right)\right).$$

```
> w:= n -> n! * (exp(1) - Sum(1/k!,k=0..n));
```

$$w := n \rightarrow n!\left(\mathrm{e} - \left(\sum_{k=0}^{n} \frac{1}{k!}\right)\right)$$

```
> Digits:= 10: seq(evalf(w(n)),n=0..15);
1.718281828, 0.718281828, 0.436563656, 0.30969097, 0.23876388, 0.1938194, 0.162916,
    0.14041, 0.1233, 0.1095, 0.098, 0.1, 0., 0., 0., 0.
```

Catastrophic cancellation is giving us fewer and fewer significant digits, because we're subtracting two numbers e and $\sum_{k=0}^{n} \frac{1}{k!}$ that are very close. Can we get around that?

We could increase Digits, but for any particular value of Digits we'll still have a problem when n is large enough (and arbitrary precision isn't always available).

It's easy to see that we have the following recurrence relation:

```
> recurrence := y(n) = n*y(n-1) - 1;
```

$$recurrence := y(n) = n\, y(n-1) - 1$$

```
> eval(recurrence, y=w);
```

$$n!\left(\mathrm{e} - \left(\sum_{k=0}^{n} \frac{1}{k!}\right)\right) = n\,(n-1)!\left(\mathrm{e} - \left(\sum_{k=0}^{n-1} \frac{1}{k!}\right)\right) - 1$$

```
> expand(%);
```

$$n!\,\mathrm{e} - n!\left(\sum_{k=0}^{n} \frac{1}{k!}\right) = n!\,\mathrm{e} - n!\left(\sum_{k=0}^{n-1} \frac{1}{k!}\right) - 1$$

```
> simplify(lhs(%)-rhs(%));
```

$$-n!\left(\sum_{k=0}^{n} \frac{1}{k!}\right) + n!\left(\sum_{k=0}^{n-1} \frac{1}{k!}\right) + 1$$

```
> combine(%);
```

$$\sum_{k=n}^{n}\left(-\frac{n!}{k!}\right)+1 \qquad (1.1)$$

```
> value(%);
```

$$0 \qquad (1.2)$$

This suggests the following method.

```
> W[0]:= evalf(exp(1)-1);
```

$$W_0 := 1.718281828$$

```
> for count from 1 to 10 do
    W[count]:= count*W[count-1] - 1
  end do;
```

$$W_1 := 0.718281828$$
$$W_2 := 0.436563656$$
$$W_3 := 0.309690968$$
$$W_4 := 0.238763872$$
$$W_5 := 0.193819360$$
$$W_6 := 0.162916160$$
$$W_7 := 0.140413120$$
$$W_8 := 0.123304960$$
$$W_9 := 0.109744640$$
$$W_{10} := 0.097446400$$

So far, so good? But...

```
> for count from 11 to 20 do
    W[count]:= count*W[count-1] - 1
  end do;
```

$$W_{11} := 0.071910400$$
$$W_{12} := -0.137075200$$
$$W_{13} := -2.781977600$$
$$W_{14} := -39.94768640$$
$$W_{15} := -600.2152960$$
$$W_{16} := -9604.444736$$
$$W_{17} := -1.632765605\ 10^{5}$$
$$W_{18} := -2.938979089\ 10^{6}$$
$$W_{19} := -5.584060369\ 10^{7}$$
$$W_{20} := -1.116812075\ 10^{9}$$

These results are absurd. The W_n should all be positive, and not very large. Roundoff error has struck again!

What happened here is that whatever roundoff error is in y_{n-1} gets multiplied by n in y_n. So any error in y_0 would be multiplied by $n!$ by the time you get to y_n, even if no further roundoff error was introduced. Eventually the roundoff errors are larger than the actual values.

How can we get around this? By using a different numerical method, one that doesn't magnify roundoff errors. In this case, one way is to use the recursion but go backwards:

```
> y(n-1) = solve(recurrence,y(n-1));
```

$$y(n-1) = \frac{1+y(n)}{n}$$

If we use an approximate value for $y(n)$ to calculate $y(n-1)$, any error in the value for $y(n)$ gets divided by n. The errors decrease, so $y(0)$ should end up very accurate (of course, additional error is introduced by roundoff at each step, but it won't be magnified). We could start off with even a very rough approximation for $y(N)$, and after a few backward steps the error will be quite small.

```
> W[50] := 0.0;
  for nn from 50 to 1 by -1 do
    W[nn-1] := (W[nn]+1)/nn
  end do;
```

$$W_{50} := 0.$$
$$W_{49} := 0.02000000000$$
$$W_{48} := 0.02081632653$$
$$W_{47} := 0.02126700681$$
$$W_{46} := 0.02172908526$$
$$W_{45} := 0.02221150185$$
$$W_{44} := 0.02271581116$$
$$W_{43} := 0.02324354116$$
$$W_{42} := 0.02379636142$$
$$W_{41} := 0.02437610383$$
$$W_{40} := 0.02498478302$$
$$W_{39} := 0.02562461958$$
$$W_{38} := 0.02629806718$$
$$W_{37} := 0.02700784387$$
$$W_{36} := 0.02775696876$$
$$W_{35} := 0.02854880469$$
$$W_{34} := 0.02938710871$$
$$W_{33} := 0.03027609144$$
$$W_{32} := 0.03122048761$$
$$W_{31} := 0.03222564025$$

$$W_{30} := 0.03329760129$$
$$W_{29} := 0.03444325337$$
$$W_{28} := 0.03567045700$$
$$W_{27} := 0.03698823061$$
$$W_{26} := 0.03840697152$$
$$W_{25} := 0.03993872969$$
$$W_{24} := 0.04159754920$$
$$W_{23} := 0.04339989788$$
$$W_{22} := 0.04536521296$$
$$W_{21} := 0.04751660059$$
$$W_{20} := 0.04988174290$$
$$W_{19} := 0.05249408715$$
$$W_{18} := 0.05539442563$$
$$W_{17} := 0.05863302367$$
$$W_{16} := 0.06227253082$$
$$W_{15} := 0.06639203319$$
$$W_{14} := 0.07109280220$$
$$W_{13} := 0.07650662871$$
$$W_{12} := 0.08280820223$$
$$W_{11} := 0.09023401683$$
$$W_{10} := 0.09911218336$$
$$W_{9} := 0.1099112183$$
$$W_{8} := 0.1233234687$$
$$W_{7} := 0.1404154336$$
$$W_{6} := 0.1629164906$$
$$W_{5} := 0.1938194152$$
$$W_{4} := 0.2387638830$$
$$W_{3} := 0.3096909708$$
$$W_{2} := 0.4365636570$$
$$W_{1} := 0.7182818285$$
$$W_{0} := 1.718281828$$

Actually $w_n = \sum_{k=n+1}^{\infty} \frac{n!}{k!}$. Maple should be able to evaluate these rapidly converging series quite well. Let's compare its results to the W_n we just computed.

```
> [seq([n,W[n]-evalf(Sum(n!/k!,k=n+1..infinity))],n=0..50)];
```

```
[[0, 0.], [1, 0.], [2, 1. 10^-10], [3, 0.], [4, 0.], [5, 1. 10^-10], [6, 1. 10^-10], [7, 0.], [8, 0.], [9,
    0.], [10, 1. 10^-11], [11, -2. 10^-11], [12, 2. 10^-11], [13, -2. 10^-11], [14, -1. 10^-11], [15,
    1. 10^-11], [16, 2. 10^-11], [17, 2. 10^-11], [18, -1. 10^-11], [19, 1. 10^-11], [20, 1. 10^-11], [21,
    0.], [22, 1. 10^-11], [23, 0.], [24, 1. 10^-11], [25, 2. 10^-11], [26, 2. 10^-11], [27, 0.], [28,
    -1. 10^-11], [29, -1. 10^-11], [30, -1. 10^-11], [31, 1. 10^-11], [32, -1. 10^-11], [33, 1. 10^-11],
    [34, 0.], [35, 0.], [36, 1. 10^-11], [37, 0.], [38, 1. 10^-11], [39, 0.], [40, 0.], [41,
    -1. 10^-11], [42, 0.], [43, 0.], [44, 1. 10^-11], [45, -8. 10^-11], [46, -3.61 10^-9], [47,
    -1.6999 10^-7], [48, -0.00000816007], [49, -0.00039984332], [50, -0.01999216584]]
```

Up to n=45, our results were about as accurate as you could hope for with Digits = 10.

If we wanted w_{20}, would it have been worthwhile to try to get a more accurate value for w_{50}?

A method is **numerically stable** if small errors in intermediate results will have a negligible effect on the final result, or **numerically unstable** if they will have a large effect on the final result. So this recurrence was numerically unstable in the forward direction, but numerically stable in the backward direction. It's a good idea to look for numerical stability when choosing a numerical method.

Approximating a function

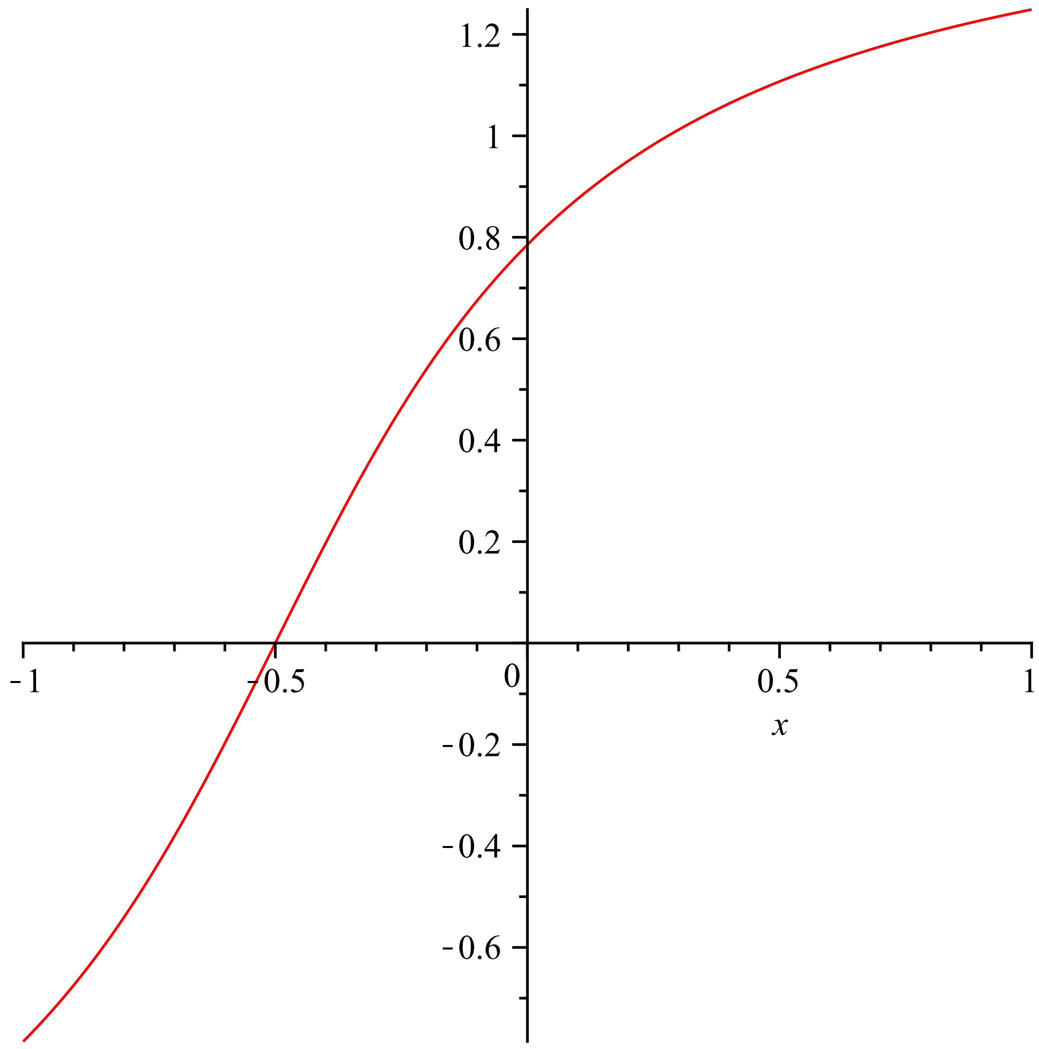
Suppose you're interested in computing a particular function $f(x)$, perhaps a rather complicated one. You're going to need to compute it lots of times, so it's important to have an efficient way of doing that. For example, perhaps you're designing a device that will need to compute this function, and will need to do it very quickly. You're willing to spend some effort in finding an efficient way to find a good method of calculating the function, if that will be rewarded with improved performance when the device is operating. Fortunately, you don't need to compute it exactly: a good approximation will do. Specifically, you are given $\varepsilon > 0$ and an interval $[a, b]$, and you want an easily computed function $g(x)$ such that $|g(x) - f(x)| < \varepsilon$ for all x in this interval. For example, a polynomial $g(x)$ with not too high a degree would be nice.

The particular example I'll take is $f_0(x) = \arctan(2x+1)$ on the interval $[-1, 1]$ with $\varepsilon = 10^{-8}$.

```
> f0:= x -> arctan(2*x+1);
```

$$f0 := x \to \arctan(2x+1)$$

```
> plot(f0(x),x=-1 .. 1);
```



Hearing the words "polynomial" and "approximation", a Calculus student might first think of Taylor series. However, these turn out not to be useful for our purpose. The idea of a Taylor series is that it gives a really good approximation of $f_0(x)$ near one point. On the rest of the interval, however, the approximation might be terrible.

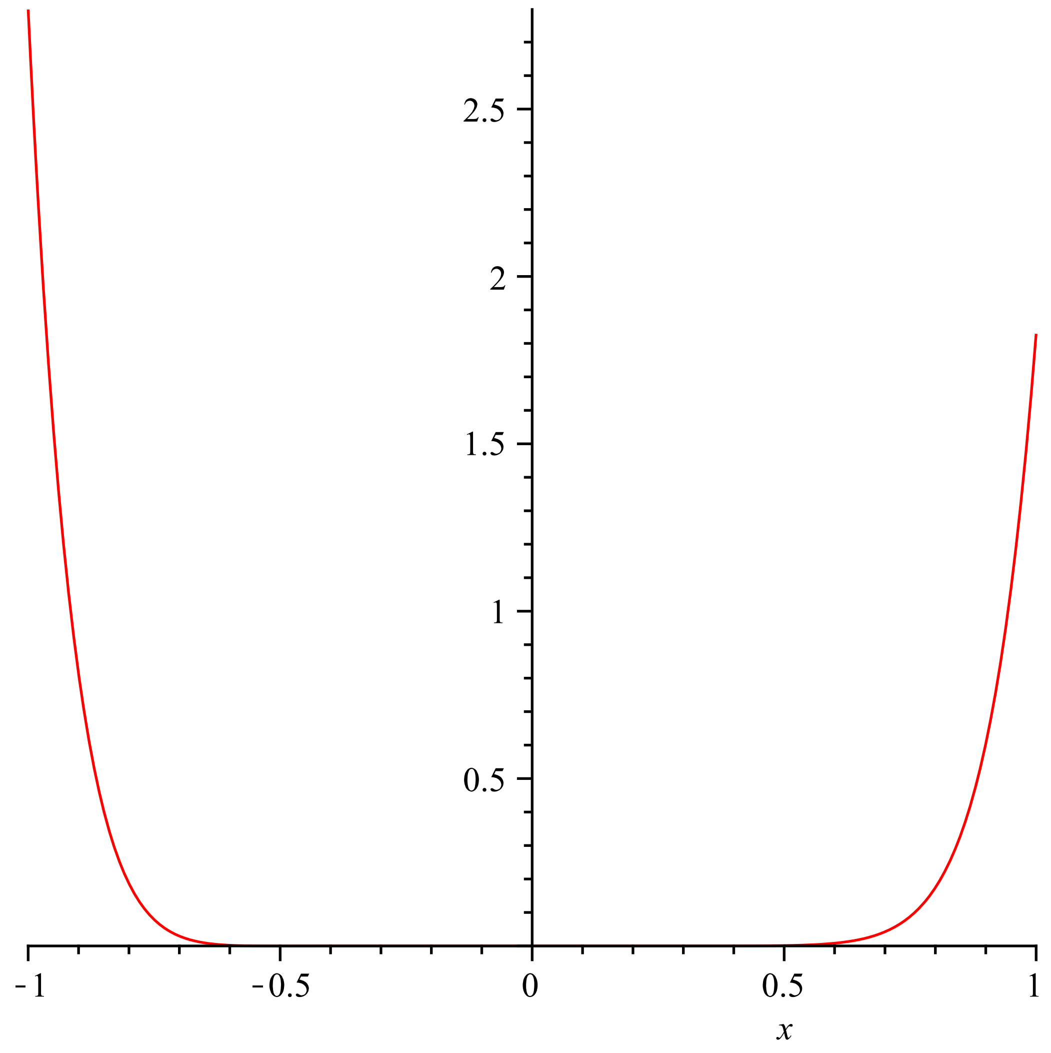
```
> taylor(f0(x), x=0, 11);
```

$$\frac{1}{4}\pi + x - x^2 + \frac{2}{3}x^3 - \frac{4}{5}x^5 + \frac{4}{3}x^6 - \frac{8}{7}x^7 + \frac{16}{9}x^9 - \frac{16}{5}x^{10} + \mathrm{O}(x^{11})$$

```
> TaylorApp:= unapply(convert(%, polynom), x);
```

$$TaylorApp := x \rightarrow \frac{1}{4}\pi + x - x^2 + \frac{2}{3}x^3 - \frac{4}{5}x^5 + \frac{4}{3}x^6 - \frac{8}{7}x^7 + \frac{16}{9}x^9 - \frac{16}{5}x^{10}$$

```
> plot(f0(x) - TaylorApp(x), x=-1..1);
```



The approximation is excellent for x near 0, but poor near the ends of the interval, so the maximum error is about 2.8. In fact, the radius of convergence of this Taylor series is $\frac{\sqrt{2}}{2}$, and for $|x|$ larger than this the Taylor series is basically useless.

Fourier Series

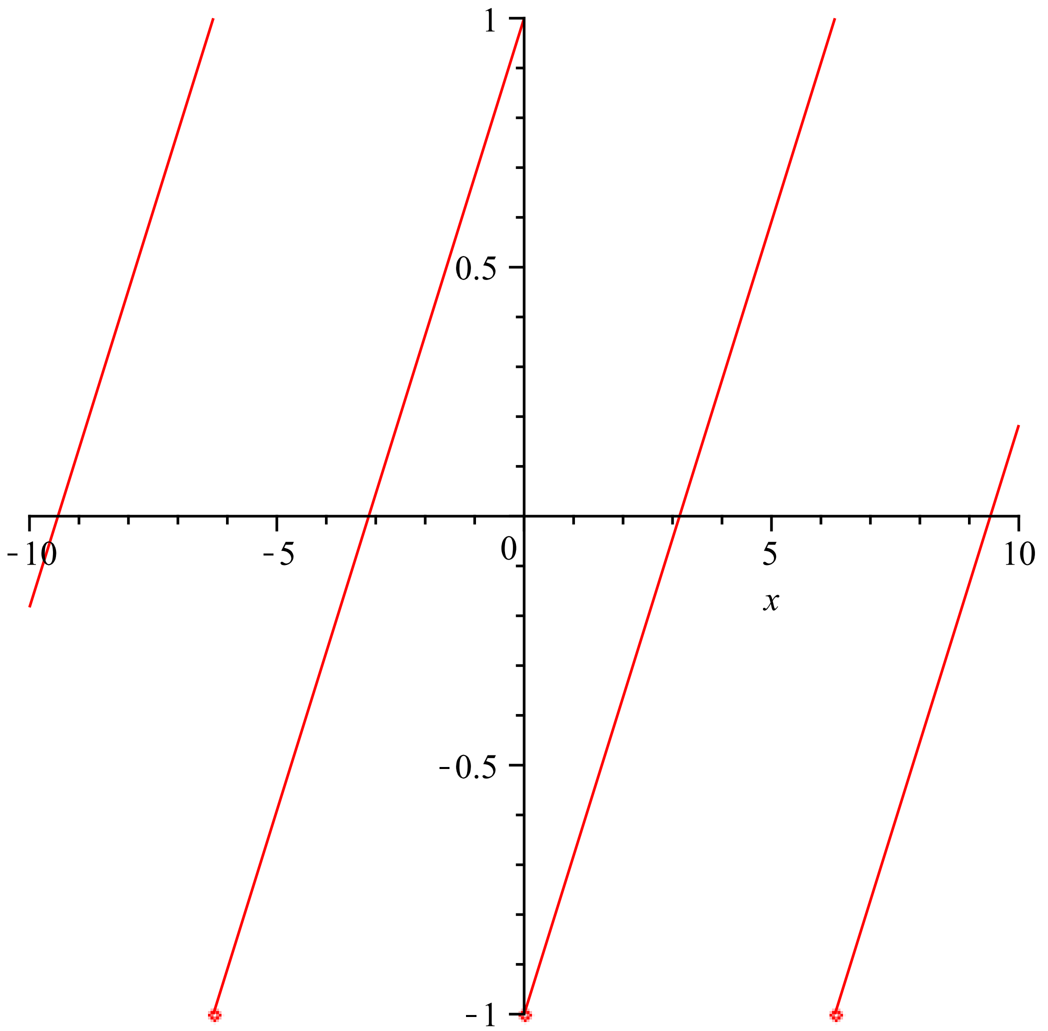
The next type of series to look at is a Fourier series. We're looking at a function on $[-1, 1]$, so we need to relate the interval

$[0, 2\pi]$ that we used for Fourier series to $[-1, 1]$. The appropriate **saw** function is

```
> saw := x -> x/Pi - 2*floor(x/(2*Pi)) - 1;
```

$$saw := x \rightarrow \frac{x}{\pi} - 2\,\text{floor}\left(\frac{1}{2}\,\frac{x}{\pi}\right) - 1$$

```
> plot(saw(x),x=-10 .. 10, discont=true);
```

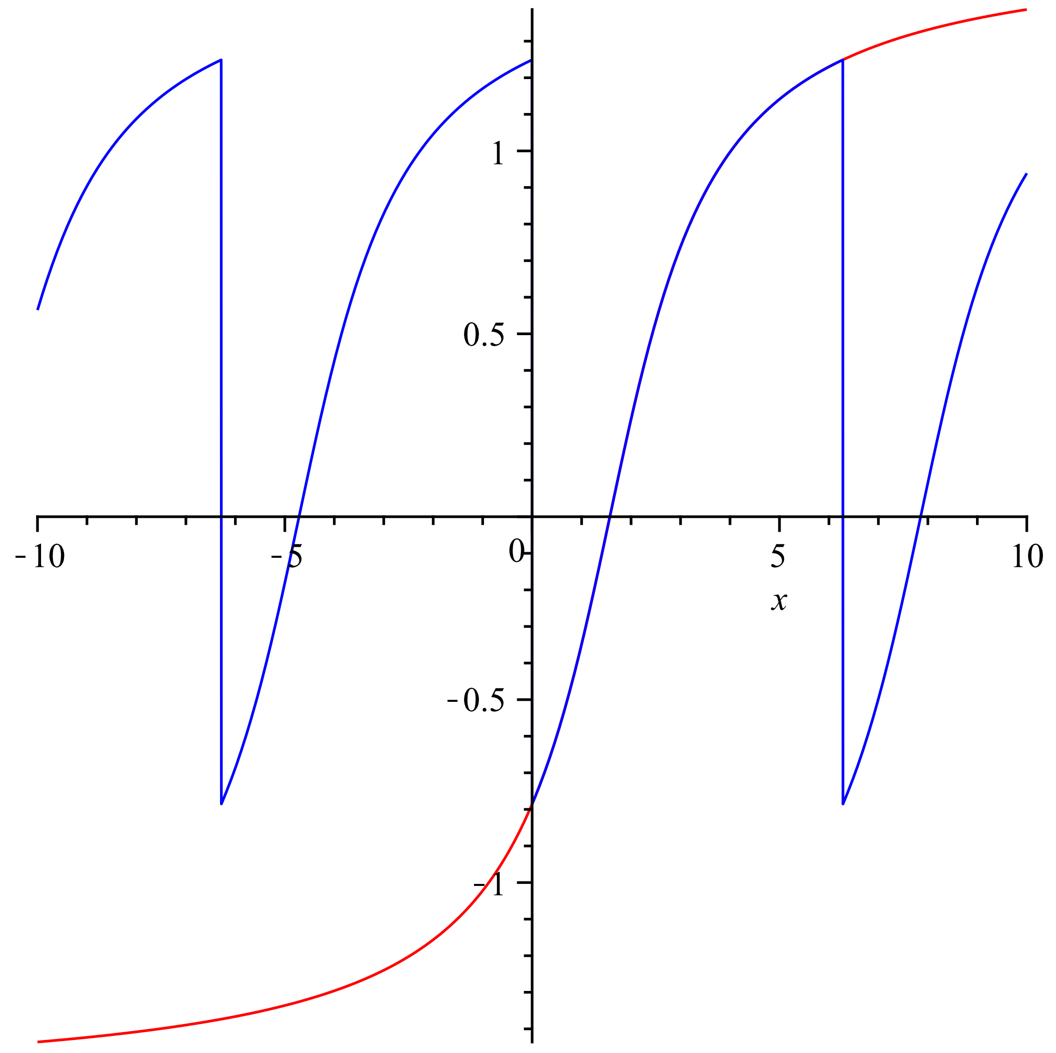


The periodic version of $f_0(x)$ is then $f_0(saw(x))$.

```
> f1 := f0 @ saw;
```

$$f1 := f0@saw$$

```
> plot([f0(x/Pi-1), f1(x)],x=-10..10, colour=[red,blue]);
```



Now for the Fourier series coefficients. Maple can't do these integrals symbolically.

```
> 1/Pi * int(f1(t)*cos(k*t),t=0..2*Pi) assuming k::integer;
```

$$\frac{\int_{0}^{2\pi} \arctan\left(\frac{2t}{\pi} - 4\,\mathrm{floor}\left(\frac{1}{2}\,\frac{t}{\pi}\right) - 1\right)\cos(k\,t)\,\mathrm{d}t}{\pi}$$

That $\mathrm{floor}\left(\frac{1}{2}\,\frac{t}{\pi}\right)$ should be 0 for *t* in this interval. It's curious that Maple can't simplify this:

```
> simplify(floor(t/(2*Pi))) assuming t>0,t<2*Pi;
```

$$\mathrm{floor}\left(\frac{1}{2}\,\frac{t}{\pi}\right) \tag{3.1}$$

That should be 0. If we manually make it 0, will it work?

```
> Ak:=eval(%%, floor=0) assuming k::posint;
  simplify(%) assuming k::posint;
```

$$Ak := \frac{\int_0^{2\pi} \arctan\left(\frac{2\,t}{\pi} - 1\right)\cos(k\,t)\,\mathrm{d}t}{\pi}$$

$$-\frac{\int_0^{2\pi} \arctan\left(\frac{-2\,t+\pi}{\pi}\right)\cos(k\,t)\,\mathrm{d}t}{\pi} \quad \textbf{(3.2)}$$

What about for particular values of k?

```
> eval(%,k=2);
```

$$-\frac{\int_0^{2\pi} \arctan\left(\frac{-2\,t+\pi}{\pi}\right)\cos(2\,t)\,\mathrm{d}t}{\pi} \quad \textbf{(3.3)}$$

```
> eval(%%,k=0);
```

$$-\frac{\frac{1}{8}\pi^2 - \frac{3}{2}\pi\arctan(3) + \frac{1}{4}\pi\ln(5)}{\pi} \quad \textbf{(3.4)}$$

Except for $k = 0$, it still can't do the integral in closed form. No problem, we'll do the coefficients using numerical integration. Knowing this, it's better to use Int rather than int, to save us from wasting time with symbolic integrations that won't work.

```
> a:= unapply(1/Pi*Int(arctan(2*t/Pi-1)*cos(k*t),t=0..2*Pi), k)
  ;
  a(0):= value(a(0));
```

$$a := k \rightarrow \frac{\int_0^{2\pi} \arctan\left(\frac{2\,t}{\pi} - 1\right)\cos(k\,t)\,\mathrm{d}t}{\pi}$$

$$a(0) := \frac{-\frac{1}{8}\pi^2 + \frac{3}{2}\pi\arctan(3) - \frac{1}{4}\pi\ln(5)}{\pi}$$

```
> seq(evalf(a(k)),k=0..5);
1.078510098, -0.2613646026, -0.01778157083, -0.005502151119, -0.004907326798,      (3.5)
    -0.003256263089
```

Similarly for b(k).

```
> b:= unapply(1/Pi * Int(arctan(2*t/Pi-1)*sin(k*t),t=0..2*Pi),
  k);
```

$$b := k \rightarrow \frac{\int_0^{2\pi} \arctan\left(\frac{2\,t}{\pi} - 1\right)\sin(k\,t)\,\mathrm{d}t}{\pi}$$

For an approximation to the sum of the series, we can use a partial sum, i.e. take the sum up to some N instead of up to infinity.

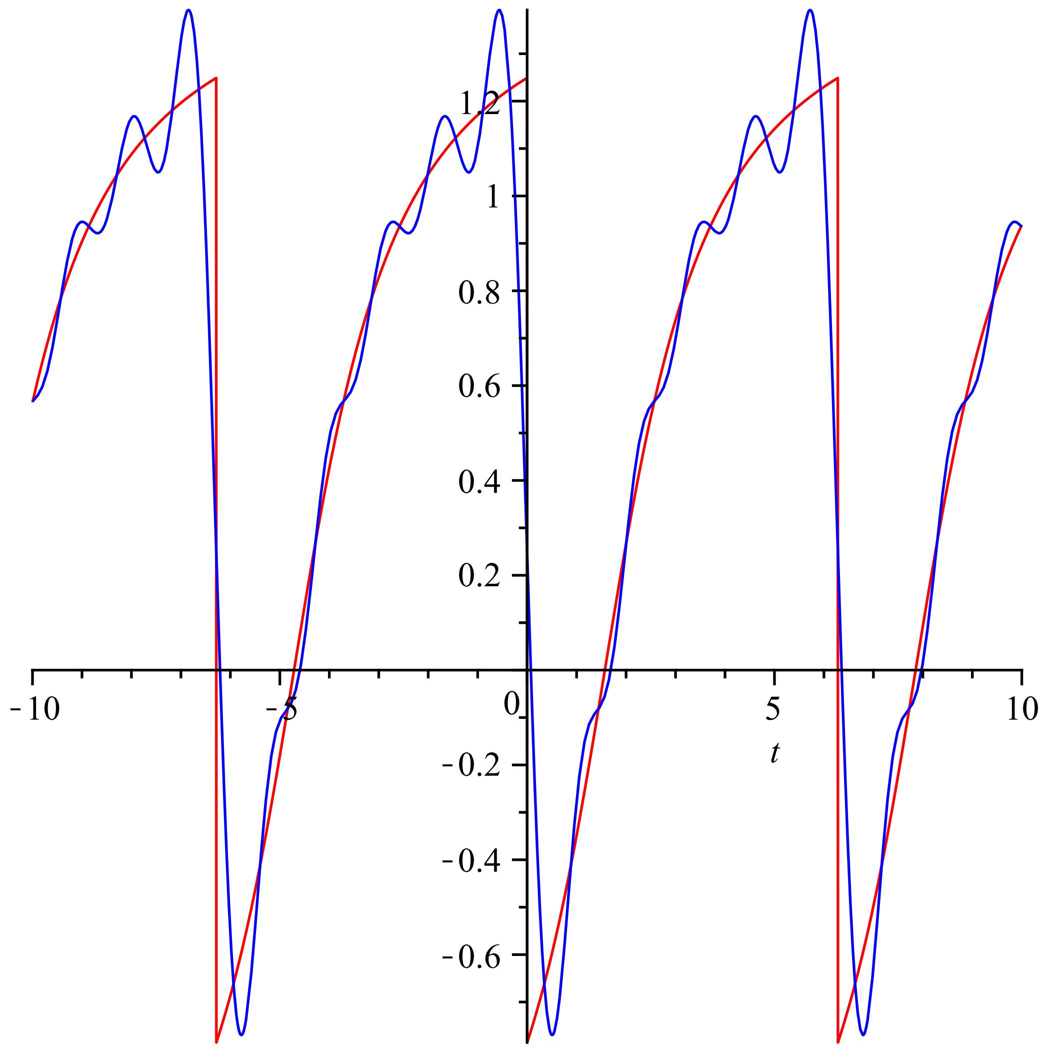
```
> Psum := N -> evalf(a(0)/2 + add(a(k)*cos(k*t) + b(k)*sin(k*
  t),k=1..N));
```

$$Psum := N \rightarrow evalf\left(\frac{1}{2}\, a(0) + add(a(k)\cos(k\,t) + b(k)\sin(k\,t), k=1..N)\right)$$

```
> Psum(5);
```

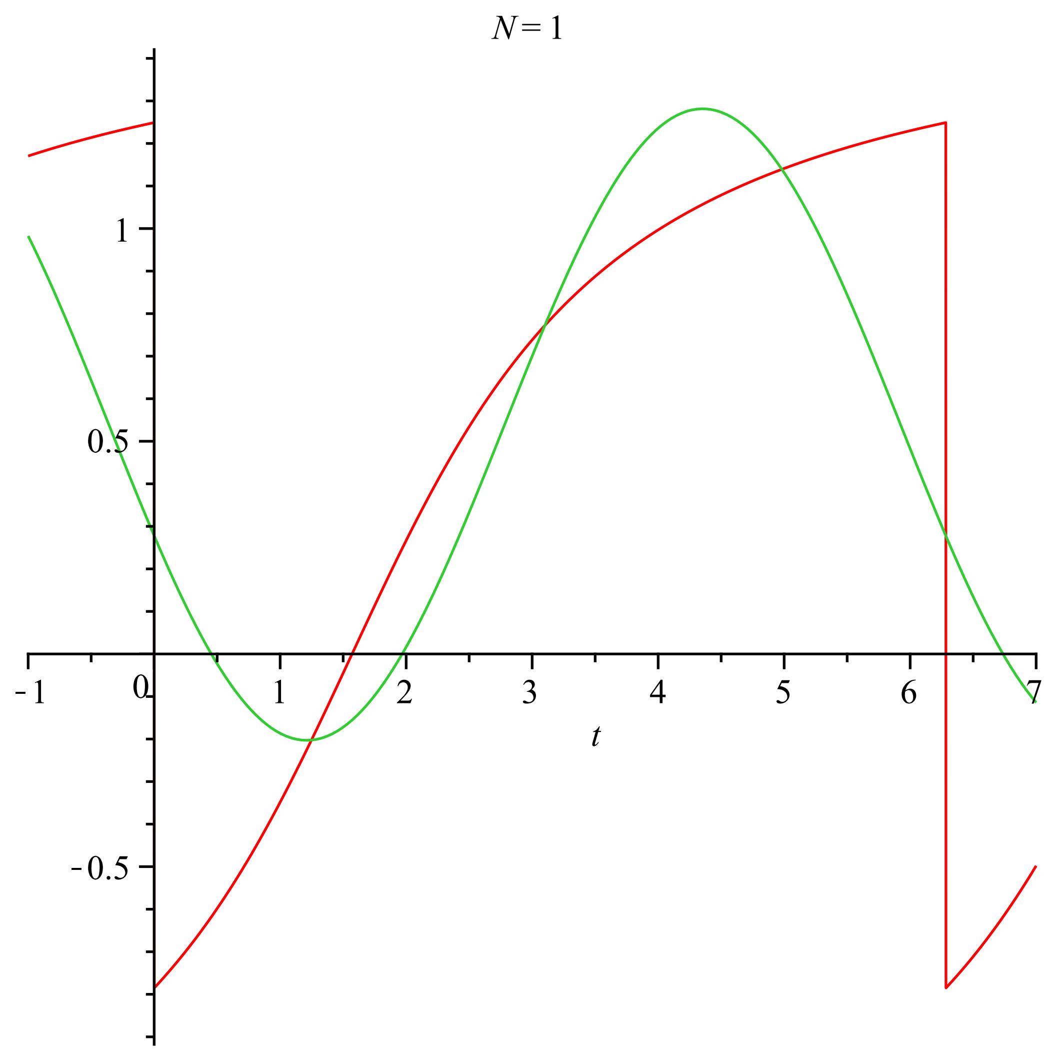
$$0.5392550490 - 0.2613646026\cos(t) - 0.6946934775\sin(t) - 0.01778157083\cos(2.\,t) - 0.3534766624\sin(2.\,t) - 0.005502151119\cos(3.\,t) - 0.2184358835\sin(3.\,t) - 0.004907326798\cos(4.\,t) - 0.1625398224\sin(4.\,t) - 0.003256263089\cos(5.\,t) - 0.1300901623\sin(5.\,t)$$

```
> plot([f1(t),Psum(5)],t=-10..10,colour=[red,blue]);
```



Here's an animation to see how well the approximation goes as N increases. I'm using **display(..., insequence=true)** rather than **animate** to avoid premature evaluation problems.

```
> with(plots):
  display([seq(plot([f1(t),Psum(N)],t=-1..7,title=('N'=N)),N=1.
  .20)],insequence=true);
```

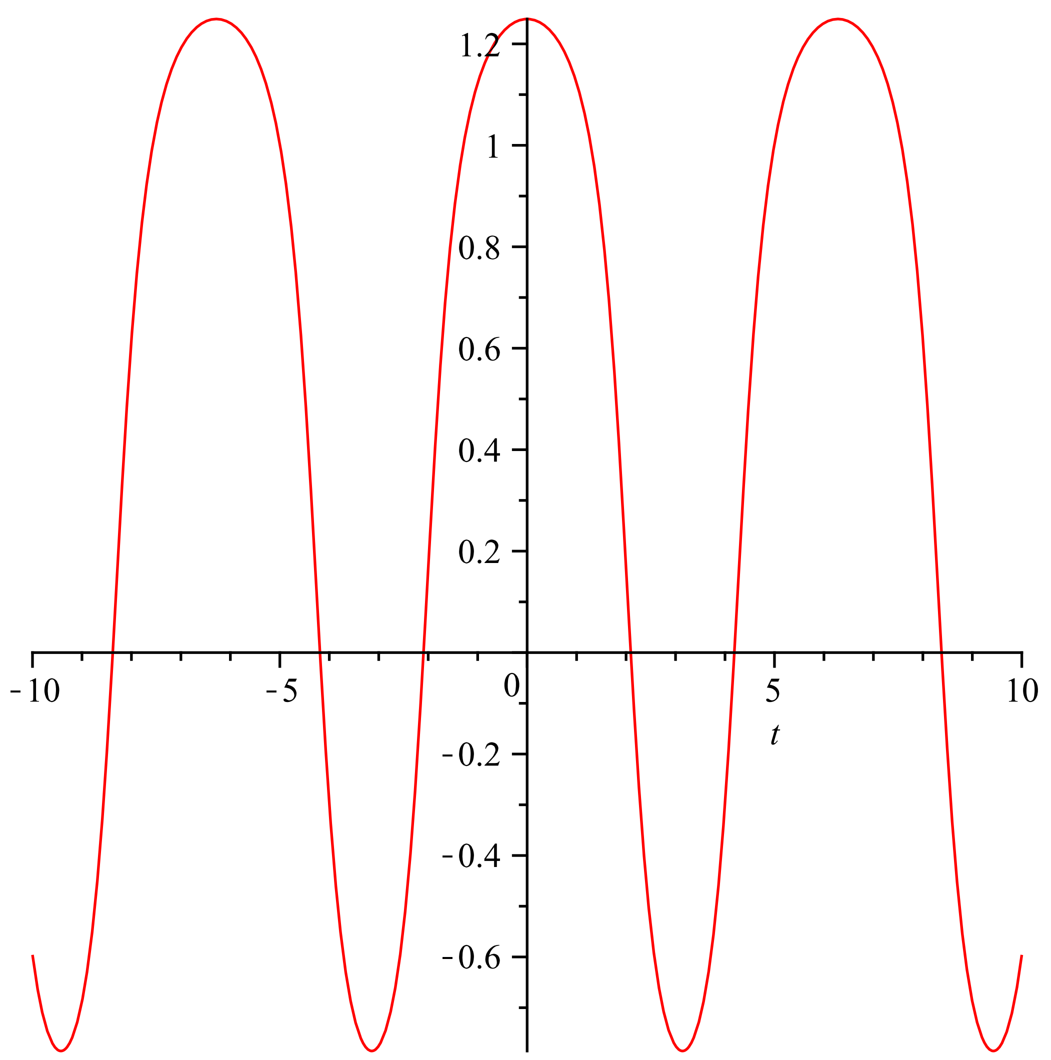


The approximation is rather bad near the jumps at 0 and 2π. That's inevitable: the partial sum is a continuous function, so it can't stay close to the discontinuous function f_1 when that takes a jump. Somewhat surprising, perhaps, is the fact that the partial sum "overshoots" the jump, and that the size of the "overshoot", i.e. the difference between the maximum of Psum(N) and the maximum of f_1, doesn't go to 0 as N increases. This is called the **Gibbs phenomenon**.

Fourier becomes Chebyshev

The poor approximation of our function by partial sums of its Fourier series is due to the discontinuities. In general, the smoother a function is, the faster its Fourier series coefficients go to 0, and so the better approximation the partial sums of the Fourier series are for the function. The best case is for an analytic function, where the coefficients go to 0 exponentially as $n \rightarrow \infty$. So we might try replacing the saw function by something analytic that takes the interval $[0, 2\pi]$ to $[-1, 1]$. Well, the cosine function does that.

```
> plot(f0(cos(t)),t=-10..10);
```



Now we have a nice analytic function, that will be very well approximated by its Fourier series.

This has an additional, very important, benefit: $\cos(k\,t)$ is a polynomial in $\cos(t)$, so our approximation becomes a polynomial in x rather than a trigonometric function.

```
> seq(expand(cos(k*t)), k = 2 .. 10);
```

$$2\cos(t)^2-1,\ 4\cos(t)^3-3\cos(t),\ 8\cos(t)^4-8\cos(t)^2+1,\ 16\cos(t)^5-20\cos(t)^3+5\cos(t),\ 32\cos(t)^6-48\cos(t)^4+18\cos(t)^2-1,\ 64\cos(t)^7-112\cos(t)^5+56\cos(t)^3-7\cos(t),\ 128\cos(t)^8-256\cos(t)^6+160\cos(t)^4-32\cos(t)^2+1,\ 256\cos(t)^9-576\cos(t)^7+432\cos(t)^5-120\cos(t)^3+9\cos(t),\ 512\cos(t)^{10}-1280\cos(t)^8+1120\cos(t)^6-400\cos(t)^4+50\cos(t)^2-1$$

```
> fp := unapply(f0(cos(t)),t);
```

$$fp := t \rightarrow \arctan(2\cos(t)+1)$$

```
> for count from 0 to 10 do
    a[count]:= evalf(1/Pi * Int(fp(t)*cos(count*t),t=-Pi..Pi))
  end do:
```

```
> a[0]/2 + add(a[k]*cos(k*t), k = 1 .. 10);
```

$$0.4522784470+1.058171027\cos(t)-0.2720196494\cos(2\,t)-0.02881149458\cos(3\,t)$$

$+ 0.05817102725 \cos(4\,t) - 0.01794454247 \cos(5\,t) - 0.005730457513 \cos(6\,t) + 0.006960372166 \cos(7\,t) - 0.001644114298 \cos(8\,t) - 0.001121821048 \cos(9\,t) + 0.0009741971126 \cos(10\,t)$

```
> expand(%);
```

$$0.7680442382 \cos(t)^3 + 0.7855812697 - 1.551301055 \cos(t)^5 + 1.328619386 \cos(t)^6 - 1.457418934 \cos(t)^8 - 0.2871861883 \cos(t)^9 + 0.0876930459 \cos(t)^4 - 1.011234239 \cos(t)^2 + 1.091632742 \cos(t)^7 + 0.9960638037 \cos(t) + 0.4987889217 \cos(t)^{10} \tag{4.1}$$

```
> eval(%,cos(t)=x);
```

$$0.7680442382\, x^3 + 0.7855812697 - 1.551301055\, x^5 + 1.328619386\, x^6 - 1.457418934\, x^8 - 0.2871861883\, x^9 + 0.0876930459\, x^4 - 1.011234239\, x^2 + 1.091632742\, x^7 + 0.9960638037\, x + 0.4987889217\, x^{10} \tag{4.2}$$

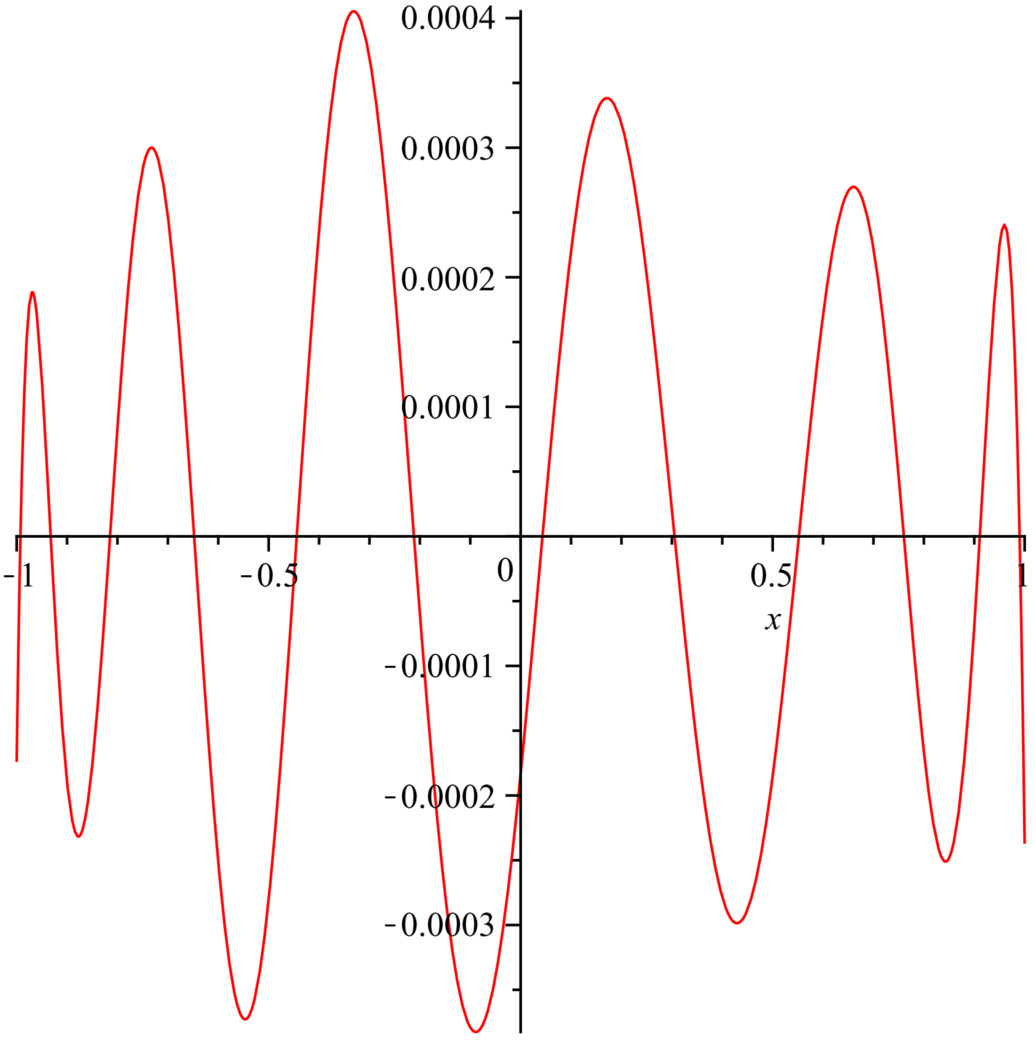
```
> ChebApp:= unapply(%,x);
```

$$ChebApp := x \rightarrow 0.7680442382\, x^3 + 0.7855812697 - 1.551301055\, x^5 + 1.328619386\, x^6 - 1.457418934\, x^8 - 0.2871861883\, x^9 + 0.0876930459\, x^4 - 1.011234239\, x^2 + 1.091632742\, x^7 + 0.9960638037\, x + 0.4987889217\, x^{10}$$

The **Cheb** is short for "Chebyshev". This is called a **Chebyshev series** approximation for $f_0(x)$ on the interval [-1,1]. The polynomials $T_n(s)$ such that $\cos(n\,t) = T_n(\cos(t))$ are called the **Chebyshev polynomials of the first kind**.

Let's see how well the Chebyshev series does at approximating $f_0(x)$.

```
> plot(f0(x)-ChebApp(x),x=-1..1);
```



Chebyshev approximations in Maple

Actually Maple has commands to produce Chebyshev series approximations: **chebpade** and **chebyshev** in the **numapprox** package. **chebpade** stands for Chebyshev-Padé, and the Padé part is because it can do something a bit more general than just a Chebyshev series.
We can use **chebpade** to produce an approximation to a given degree N.

```
> with(numapprox):
> chebpade(f0(x),x=-1..1,10);
```

$$-0.0288114945860669\ T(3,x)+0.0581710272714922\ T(4,x)$$
$$-0.0179445424708702\ T(5,x)-0.00573045751510578\ T(6,x)$$
$$+1.05817102727149\ T(1,x)-0.272019649514069\ T(2,x)$$
$$+0.452278447151191\ T(0,x)-0.00112182104832474\ T(9,x)$$
$$+0.000974197112900724\ T(10,x)+0.00696037216927651\ T(7,x)$$
$$-0.00164411429807496\ T(8,x)$$

The result is given in terms of **T(n,x)**, which stand for the Chebyshev polynomials. Notice that the coefficient of each $T(n,x)$ is the coefficient of $\cos(n\,t)$ in our Fourier series. The actual definition of $T(n, x)$ as Chebyshev polynomials is contained in the **orthopoly** package. I'm not going to load that package, but instead use it this way:

```
> eval(%, T = orthopoly[T]);
```

$$0.996063804055484\, x + 0.768044238351589\, x^3 - 1.55130105536918\, x^5 + 1.32861938627262\, x^6 - 1.45741893466652\, x^8 - 0.287186188371133\, x^9 + 0.0876930460447320\, x^4 - 1.01123423928854\, x^2 + 1.09163274266875\, x^7 + 0.498788921805171\, x^{10} + 0.785581270040882$$

Notice that this is our ChebApp(x) (up to round-off error affecting the last digit of some coefficients).

```
> % - ChebApp(x);
```

$$3.55484086611568\ 10^{-10}\, x + 1.51588852581597\ 10^{-10}\, x^3 - 3.69178687620320\ 10^{-10}\, x^5 + 2.72615929830522\ 10^{-10}\, x^6 - 6.66521504655293\ 10^{-10}\, x^8 - 7.11329883884559\ 10^{-11}\, x^9 + 1.44731990281421\ 10^{-10}\, x^4 - 2.88544965698634\ 10^{-10}\, x^2 + 6.68745725462827\ 10^{-10}\, x^7 + 1.05170538944321\ 10^{-10}\, x^{10} + 3.40881989302488\ 10^{-10}$$

The maximum error in ChebApp(x) was about $4\ 10^{-4}$, still more than our $\varepsilon = 10^{-8}$. To attempt an approximation with error at most ε, we can try **chebyshev**. We give **chebyshev** our desired ε, and it will produce an approximation of whatever degree it needs.

```
> chebyshev(f0(x),x=-1..1,10^(-8));
```

$$-0.0000757640505946219\, T(18, x) + 0.0000296610644775531\, T(19, x) - 0.0288114945860669\, T(3, x) + 9.10669348976225\ 10^{-7}\, T(20, x) + 0.0581710272714922\, T(4, x) - 0.0179445424708701\, T(5, x) + 4.24644368876864\ 10^{-8}\, T(25, x) + 5.66240774303582\ 10^{-8}\, T(26, x) + 2.49134438606034\ 10^{-9}\, T(32, x) - 4.16389150932986\ 10^{-8}\, T(27, x) + 2.43382924141505\ 10^{-7}\, T(23, x) + 0.000142833053852007\, T(13, x) - 0.00573045751510579\, T(6, x) + 1.05817102727149\, T(1, x) - 0.272019649514069\, T(2, x) + 0.452278447151191\, T(0, x) - 0.00112182104832474\, T(9, x) + 2.45005887196057\ 10^{-9}\, T(28, x) + 1.21978131469271\ 10^{-8}\, T(29, x) + 0.000974197112900738\, T(10, x) - 2.43009422324076\ 10^{-7}\, T(24, x) + 0.00696037216927648\, T(7, x) - 0.00164411429807495\, T(8, x) - 0.00000137447961971678\, T(21, x) + 3.88083734932629\ 10^{-7}\, T(22, x) + 0.0000209405094970062\, T(16, x) - 0.000137741903096520\, T(11, x) - 0.000215989116121847\, T(12, x) + 0.00000233904648048977\, T(17, x) - 0.00000413079194211789\, T(14, x) - 0.0000408504603294581\, T(15, x) - 6.85347846873698\ 10^{-9}\, T(30, x)$$

The degree here will be 32.

Maple objects introduced in this lesson:

numapprox package
chebpade (in **numapprox** package)
orthopoly[T]
chebyshev (in **numapprox** package)