Use ode called from sum to make a list of points y(x(i)) and then plot the list.

In this example, the ode key solves an ordinary differential equation (initial value problem). We call the ode function repeatedly from the sum function and make a list of points on the solution curve. Then we use the list plot function to graph the solution curve for the differential equation.

The initial value problem is

$$y'' = -y, \quad y(0) = 1, \quad y'(0) = 0$$

This problem has a known solution, $y(x) = \cos(x)$, allowing us to check the accuracy of the ode key.

The sum key will be used to run a loop where x = 0.02, 0.04, 0.06, ..., 1.00. We won't use the summation value returned from the sum key — we just need it to loop over the x values.

For each of these 50 values of x, we use ode to solve from $(x - 0.02)$ to x, and then store that $y(x)$ in register $20 + 50x$. That will create a list of values for the solution curve in registers 21, 22, ..., 70.

The input to the ode key is a bit more complicated than other similar functions such as integrate or sum. Four values are entered on the stack:

k, the order of the differential equation (k is 1, 2, or 3)
a, the initial value of x at which to start solving the d.e.
b, the end value of x at which to stop
n, the function number that defines the right-hand-side of the d.e.

The general form of the differential equation for the ode key is

$$\frac{d^k}{dx^k}\, y = f_n(x, y, y', \ldots)$$

from $x = a$ to b. k can be 1, 2, or 3. f_n takes its x-argument from the stack, y from register 0, y' from register 1 (if k $\geq$ 2), y'' from register 2 (if k = 3).

The initial conditions are:

$y(a)$ in register 0,

$y'(a)$ in register 1 (if k $\geq$ 2),

$y''(a)$ in register 2 (if k = 3).

For the example here, $y'' = -y, \quad y(0) = 1, \quad y'(0) = 0$, we have

$k = 2, \quad a = 0, \quad b = 1, \quad n = 0, \quad$ register 0 = 1, $\quad$ register 1 = 0.

After the ode function finishes, the final values for $y(b)$ and $y'(b)$ will be left in registers 0 and 1. This is convenient for making a list of y values at x = 0.02, 0.04, 0.06, ..., 1.00.

The reason for doing a list plot here is because each point in the plot is the result of an ode solution, so it is slow. A full plot command would need over 10 times as many points, and the points might not be evaluated in order of increasing x, forcing us to start each ode call from $x = 0$ for each point in the plot.

The sum function will call ode repeatedly, first solving from $a = 0.00$ to $b = 0.02$, then from $a = 0.02$ to $b = 0.04$, etc. After each call to ode, the final values of $y(x)$ and $y'(x)$ left in register 0 and register 1 will automatically become the initial conditions for the next call to ode when the next term in the sum is evaluated.

Here are the functions we will need:

f0: 0, rcl, chs

f1: 7, func, 11, sto, 2, x↔y, 0.02, −, 11, rcl, 0, ode, 11, rcl, 50, *, 20, +, sto

f2: 50, *, 20, +, rcl

f3: 7, func, 1, enter, 0, sto, 0, enter, 1, sto, 1, enter, 20, sto, 0.02, enter, 1, enter, 0.02, enter, 1, sum, 6, func, 0, enter, 1, enter, 0.02, enter, 2

f0 is the right-hand-side of the d.e., $f(x, y, y') = -y$

f1 is the function that the sum key will call with $x = 0.02, 0.04, 0.06, ..., 1.00$.
It stores the current x in register 11, then sets up the arguments to call ode.
2 is the order (k), $x - 0.02$ is the starting point (a), x (11, rcl) is the stopping point (b),
0 is the right-hand-side function number (n).
Then it calls ode, which leaves $y(x)$ in the display (x-register), and computes $50x + 20$ as the register number in which this $y(x)$ will be stored as part of the list of y values.

f2(x) returns the value in register number $50x + 20$.
This translates between the x values 0.02, 0.04, 0.06, ..., 1.00, and the register numbers in which the corresponding $y(x)$ values are stored, and returns that $y(x)$ value. This can be used in a list plot:
0, enter, 1, enter, 0.02, enter, 2, lplot

f3 sets up the initial conditions for the d.e., calls sum, and sets up the parameters for a list plot.

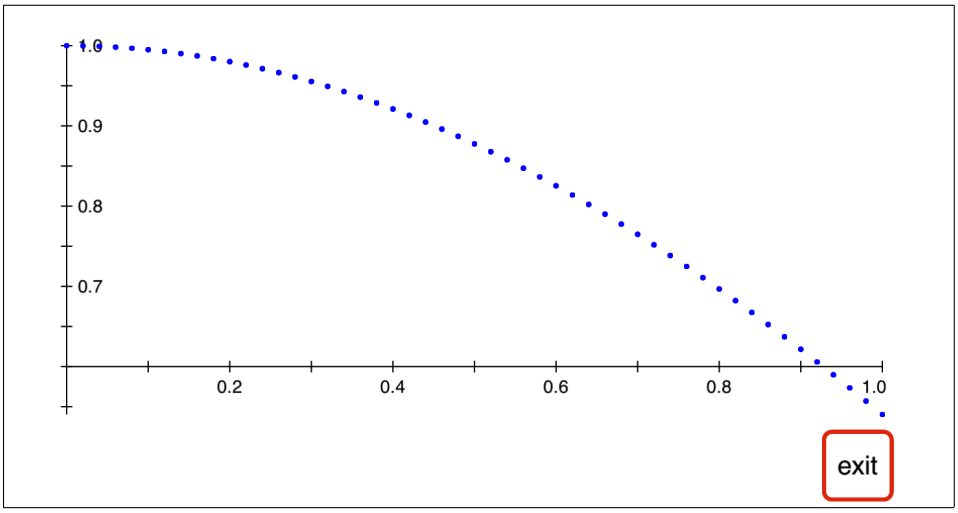
1, enter, 0, sto	($y(0) = 1$)
0, enter, 1, sto	($y'(0) = 0$)
1, enter, 20, sto	(put $y(0) = 1$ into the list of y values)
0.02, enter, 1, enter, 0.02, enter, 1, sum	(sum f1 from $x = 0.02$ to 1 with steps of 0.02)
0, enter, 1, enter, 0.02, enter, 2	(set up for a list plot of f2 from $x = 0$ to 1 by 0.02)

The plotting functions cannot be entered into user function definitions, since they are meant to be interactive, with graphs being displayed and possible zoom in or zoom out re-drawing of the graph. User functions are meant to run automatically, without stopping to interact with the user. So function f3 does everything except the final lplot function.

Using a step size of 0.02 might cause a problem with normal computer arithmetic, since 0.02 is not exactly representable in base 2. Stepping from 0 to 1 by adding 0.02 for each step might not hit 1.0 exactly because of rounding errors, possibly not producing exactly 50 steps. That is not a problem with Calc-50, because the calculator is using a power of 10 base. Commonly used steps like 0.1, 0.01, 0.02, etc., are exact and no rounding errors perturb the x_i values.

Now to solve the ode and do a list plot of the solution,

7, func, 3, f_n, lplot

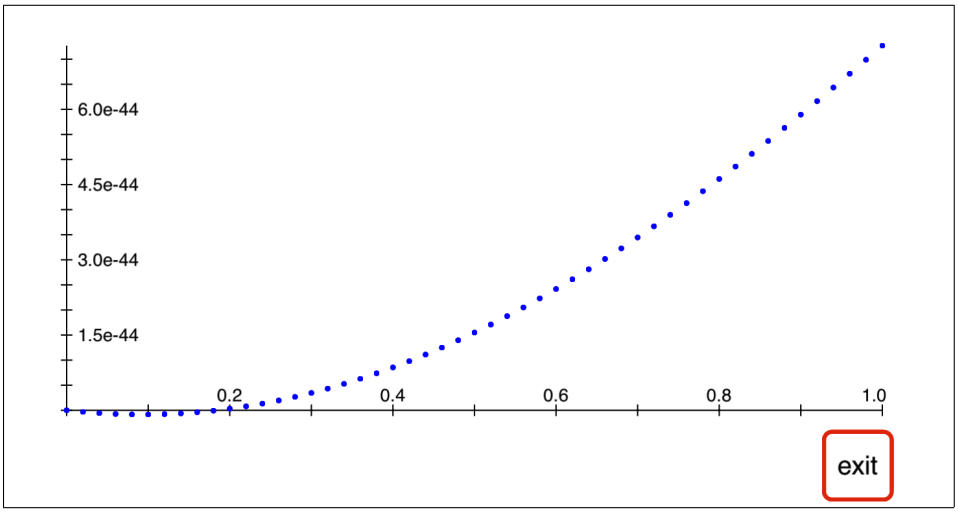


As a check of the ode key, plot the errors of each of the points in the list. f2 will return the computed points from ode, and the exact solution to this d.e. is $\cos(x)$, so define function f4 to give the computed solution minus the true solution at $x = 0.00, 0.02, 0.04, 0.06, ..., 1.00$. $f4(x) = f2(x) - \cos(x)$

f4: 1, func, 14, sto, 2, f_n, 14, rcl, cos, $-$

Then plot the errors

7, func, 0, enter, 1, enter, 0.02, enter, 4, lplot



This shows the ode results are all correct to about 43 or 44 digits.