

# **Sequences, series and limits with the HP49G**

**(and with the HP48)**

By Nick Karagiaouoglou

The programs that come with this document are changed several times according to the descriptions in the document. As only the newest versions of the programs are supplied each time a new part of the marathon is posted, you might not get the errors that earlier versions gave. Also, some of the programs of the first parts are no more necessary and thus are not part of the zip archive. The functionality of such programs is implemented in other programs, so that you have no functionality loss. However the code of old programs which were replaced and the code of old versions of programs which were updated is described in the document for study purposes.

Before you start working you should set your flags.

Enter the list { #A003008D8103F0h #0h #190101402000028h #0h } and press STOF

# Sequences, series and limits with the HP49G - Part 1

Hi all out there!

After a long fantastic vacation this is the start of the sequences, series & limits marathon, which promises to bring us more insight about the capabilities of the HP49G. I tried to collect many ideas during a long trek with Trabakoulas and for the very very first time in my life I made a to do list, so that I don't forget anything. Note that the programs written at the start of this document are not the same programs that come along with the document. The programs in the files for the HP49G are given at the end. Through the document we are going to change the programs and see how they get better (though not perfect).

We are going to examine thoroughly the powers (and flaws) of the HP49G regarding sequences, series & limits. What we do here will help us when examining limits and derivatives and integrals.

Let's bring in our minds some very elementary things, that will help us understand what a condensation point is. First we need to remember what a point set is. For the time being it suffices to accept that such a set is given, when we can prove for any point of some space, if this point is member of the set or not. Notice that "space" and "point" both don't need to be the space in which we live and its points. We could take as space for example all integers, and prove if some given integer (the point) is member of some given set or not.

Such a set, for which we can prove what are its members, is bounded, when a number exists that is bigger than the distance between any two points of the set. Often we measure the distance of such points by means of their coordinates. It suffices to consider such point sets on a straight line, on which we bring the coordinates of points. We can use more than one coordinates if necessary, that is when our set has more than one dimensions. For some one-dimensional set, if we denote the coordinates of two points with  $x_i$  and  $x_k$ , then the distance of the two points is given by  $|x_i - x_k|$ .

Can we prove if an one-dimensional set is bounded with the HP49G? Let's see. We should check if the biggest possible distance  $|x_i - x_k|$

between the points with coordinates  $x_i$  and  $x_k$  is a finite number. for

example, we consider the set of points  $P_n$  with coordinates  $\frac{1}{n}$ ,  $n$  going from 1 to  $+\infty$ . In this case we see that the point  $P_1$  with  $x_1 = 1$  is the biggest possible while the smallest possible is the point 0 as  $n$  approaches infinity. So the biggest possible distance between the points of the set is 1 and hence the set is bounded. But how should the HP49G know what the biggest and what the smallest possible points are? We can't of course evaluate  $\frac{1}{n}$  for any possible value of  $n$ . And

we also can't assume that the biggest coordinate is for  $n = 0$  and the smallest for  $n = \infty$ . Consider for example  $\frac{n}{(n-2)^2 + 3}$ . Here the

biggest possible coordinate is  $\frac{3}{4}$  for  $n = 3$  and the smallest 0 for

$n = 0$ . But there is hope on the horizon. We have the command TABVAR which can help us. First of all let's see what it does. It takes a univariate function of the current variable VX and returns a variation

table. Enter for example  $\frac{1}{X-2}$  and press TABVAR. (The command is well hidden in menu [SYMB] **GRAPH** [NXT].) The results are the function itself on stack level 3, a list that contains two sub lists in stack level 2 and a GROB on stack level 1 that contains a graphical representation of the variation table along with the function and its first and second derivatives. If you press the key [arrow-down] you can take a look at this GROB. On its lower part it contains the variation

table, an array which looks like

-	-	2	-	+	X
0				0	F

If we



take a look at the function plot then we see what the variation table means. Starting at  $X = -$  the function has the value 0. Then we have a minus sign and an arrow downwards, which tell us that the function



# Sequences, series and limits with the HP49G - Part 1

falls when we walk from  $X = -$  to  $X = 2$ . At  $X = 2$  the function goes to infinity. And then when we go from  $X = 2$  to  $X = +$  the function falls again from infinity to 0. Press [CANCEL] and [BACKSPACE] to get rid of the GROB. Now you have the variation table as a list on stack level 1. This list can be used to find possible extremal absolute values of a function. But there are also some things to be aware of. Our set  $\frac{1}{n}$  was *discrete*,  $n$  could be 1,2,3, and so on.

In the function  $\frac{1}{X}$  the variable  $X$  can take any value, like 3.5 or -1.8. That means, when an extremal value is found by TABVAR, this doesn't have to be also the extremal value of coordinates of the set.

For example consider  $\frac{X}{(X-2)^2+3}$ . TABVAR returns:

$$\begin{array}{ccccccc} - & - & -\sqrt{7} & + & \sqrt{7} & - & + & X \\ 1 & - & \frac{-2+\sqrt{7}}{6} & & \frac{2+\sqrt{7}}{6} & & 0 & F \end{array}$$

We see that for  $X = \sqrt{7}$  the function goes through a maximum  $\frac{2+\sqrt{7}}{6}$  and for  $X = -\sqrt{7}$  the function goes through a minimum

$-\frac{-2+\sqrt{7}}{6}$ . But in  $\frac{n}{(n-2)^2+3}$   $n$  can only have only integer values,

so it can't be  $\sqrt{7}$  or  $-\sqrt{7}$ . That means, if we use TABVAR for such a purpose in a program, we must check if the returned values for  $X$  are integers and if they are not, then we check what the members of the set are, that have integer coordinates adjacent to those returned. In this

example we have  $X = \sqrt{7} \approx 2.65$  which means that we take  $n = 2$

and  $n = 3$  and find the values  $\frac{2}{3}$  and  $\frac{3}{4}$  as potential extremal values of

$\frac{n}{(n-2)^2+3}$ . Another thing that we must do is to filter out those

coordinates and corresponding values that are not part of the set. TABVAR has returned  $-$ ,  $-\sqrt{7}$ ,  $\sqrt{7}$  and  $+$  as x-coordinates, but our set has  $n$  going from 1 to  $+$ . So we must throw away  $-$  and

$-\sqrt{7}$  and the corresponding values of  $\frac{X}{(X-2)^2+3}$ . And we must

also throw away all symbols like  $+$ ,  $-$ , and that TABVAR returns in order to show what happens between the x-coordinates. Additional work that must be done is to exchange  $n$  (or whatever is used as index) with the current variable  $VX$  when TABVAR is used and make the backwards substitution if needed later. That's because TABVAR works only with  $VX$  (which often is  $X$ ). (You see here the misery of having commands that want a special variable in order to work.) And there is yet another thing that we must be aware of. Suppose you want

to examine the set  $\frac{1}{n}$  for  $n = 1, 2, 3, \dots$ . Using TABVAR on  $\frac{1}{X}$  will

return the variation table  $\begin{array}{ccccccc} - & - & 0 & - & + & X \\ 0 & & & & 0 & F \end{array}$ . The set  $\frac{1}{n}$  with

$n = 1, 2, 3, \dots$  has a maximum value at  $n = 1$ . But this isn't included in

the variation table, because TABVAR examines  $\frac{1}{X}$  from  $-$  to  $+$

and in this interval nothing special happens at  $X = 1$ . So we should add our start and end coordinates in the list that is returned by TABVAR. When we have all special points, we can find the maximum and the minimum and then the absolute value of their difference. Comparing that with  $+$  we can find if the set is bounded or not. Let's make a first version of such a program. We denote a set with a list of the form:

{memberGeneralForm{variablestart end}}

# Sequences, series and limits with the HP49G - Part 1

For example the set  $\frac{1}{n}$  for  $n = 1, 2, 3, \dots$  would be denoted as

$$\frac{1}{n} \{n \mid 1 + \dots\}$$

The program should take such a set as a list and return a 1 if the set is bounded or a 0 if not. It should also return the bounds themselves as a list.

```
<<
OBJ-> DROP
OBJ-> DROP @explode set denoted as list
EVAL SWAP @Evaluate eventual infinities
EVAL SWAP
-> genMemb var lo hi @store in local variables
<<
"Calc. variation table"
1 DISP @Show message
genMemb var RCLVX = @Substitute with VX and
SUBST
IFERR @If error occurs when we
TABVAR @calculate the variation table
THEN
IF
ERRN #DE65h == @If we the function is const.
THEN
lo hi 2 ->LIST @Make list {lo hi}
OVER DUP 2 ->LIST @and {genMemb genMemb}
2 ->LIST 0.
ELSE
? DUP 2 ->LIST @else make list {? ?}
0.
END
ELSE
DROP @Drop returned GROB
OBJ-> @Transform the variation table
<< 2 ->LIST >> @to a list of lists with pairs
```

```
DOLIST @coordinate, member.
"Filtering out +,-, , "
1 DISP @Show message
1
<< @Throw away all elements
IF @in even positions
NSUB 2 MOD NOT @i.e. those with +,-, ,
THEN
DROP
END
>>
DOSUBS
"Adding low bound" @Show message
1 DISP @We use lim and not simply
lo PICK3 @EVAL to avoid errors for
RCLVX lo = lim @n=+ and similar cases.
2 ->LIST 1 ->LIST +@Add pair to list of variation
"Adding high bound"@Show message
1 DISP @Same again for max. n
hi PICK3
RCLVX hi = lim
2 ->LIST 1 ->LIST +@Add pair to list of variation
"Transforming non-
integer to integer" 1 DISP @Show message
1
<<
IF @If coordinate is algebraic
DUP HEAD DUPDUP @but not + or -
TYPE 9 ==
SWAP ABS
AND
THEN @Then add coordinates of
NIP ->NUM DUP @adjacent integers and
FLOOR R->I DUP @corresponding members
var SWAP =
genMemb SWAP
SUBST EXPAND
2 ->LIST
```

# Sequences, series and limits with the HP49G - Part 1

```

      SWAP CEIL
      R->I DUP
      var SWAP =
      genMemb SWAP
      SUBST EXPAND
      2 ->LIST
    ELSE
      DROP
    END
  >>
DOSUBS
"Filtering out < min"  @Message
1 DISP
1
<<
      @Throw away coordinates < nmin
      IF
      @and corresponding members
      DUP HEAD lo <
      THEN
      DROP
      END
    >>
DOSUBS
"Filtering out > max"  @Message
1 DISP
1
<<
      @Throw away coordinates > nmin
      IF
      @and corresponding members
      DUP HEAD hi >
      THEN
      DROP
      END
    >>
DOSUBS
1
      @Throw away coordinates
      <<
      2 GET
    >> DOSUBS
"Searching min"  @Message

```

```

1 DISP
      @Find minimum member
DUP
<<
      MIN EXPAND
    >>
STREAM
SWAP
"Searching max"  @Message
1 DISP
      @Find maximum member
DUP
<<
      MAX EXPAND
    >>
STREAM
DUP2 2 ->LIST
      @Build up list with min.
UNROT
      @and max. member
- ABS
      @Compare abs. diff. with
    END
  >>
>>

```

STOre the program in 'BOUNDS1'. (Or just use the programs which come with this document.) Before testing it let's explain some of its not so obvious parts. You certainly noticed that at the start of the program we EVALuate the starting and ending coordinates. Why do we need that? Well, the answer has to do with infinity as it is implemented on the HP49G. Remember that we enter the coordinates in a list. When we enter infinity in a list, then the resulting object does not contain something that can be used in algebraic objects. To see that better, enter { }, press [HEAD] to get the first element of the list on the stack. The infinity that now stands on stack level 1, cannot be used in algebraics. For example press [X] [SWAP] and then [=]. This should create the equation  $X =$  , should it? But instead of this you get a "Bad Argument Type Error". Enter again { }, and press [HEAD]. Press [DUP] to make a copy on stack level 2. Now press [TYPE]. The result is 14., the object type of a built-in CAS-command. Commands are not allowed in algebraic objects, at least not using

# Sequences, series and limits with the HP49G - Part 1

"standard" methods. DROP the 14. from stack level 1, and press EVAL. Now you have + and if you press [TYPE], you get a 9., the object type of algebraic objects. Such objects can be used to build up other algebraics. When you start typing a list, then you can see that the annunciator PRG lights up on the top right of the screen. What you enter now, will not be EVALuated but only written as a command in the list. It is the EVALuation of the command that puts the algebraic

infinity on the stack. When you simply press [blue-shift] and then [0] all this happens automatically, that is the command gets EVALuated first and the result, the algebraic infinity, is put on the stack.

In the program we use often the command STREAM. This command takes a list and some program from the stack, that uses two arguments. It then applies the program to the first two elements of the list and replaces the used elements with the result. This program is applied then to the new two first elements of the list, that is, the result of the first run and the third element of the original list. The process is repeated until all elements are consumed and a single result is returned. If for example we enter the list {1 2 3 4}, then the program << + >> and we press [STREAM], then the process shown on at the left takes place.

Note that for the sum of all list elements it is easier to use the command LIST and for the product it is easier to use LIST. STREAM can be used for more complicated things than simple addition of multiplication.

The last thing to explain is why we do not simply use << MAX >> and << MIN >> when we find the biggest and smallest members of the

set. This is because sometimes the algebraic result MAX(a,b) would be returned unevaluated. So we EVALuate to make sure that an evaluated result is returned.

Let's test the program. Enter  $\frac{1}{n}$  {n 1} and press [BOUNDS1]. In about 9,5 seconds the HP49G returns {0 1} and 1. The 1 means that the set is bounded, and its bounds are 0 and 1.

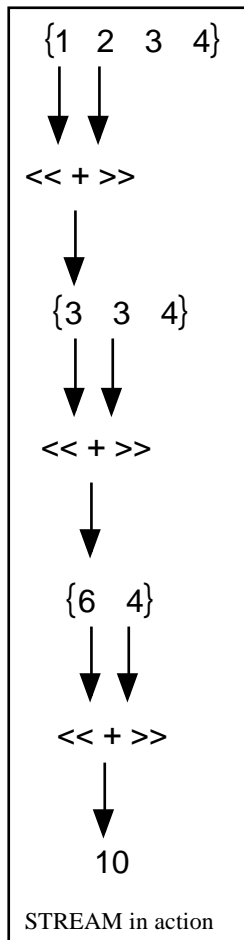
Enter  $\frac{n^2}{(n-2)^2+3}$  {n -} and press [BOUNDS1]. The results 0  $\frac{16}{7}$  and 1 are returned in about 31 seconds.

Now another example of a set that isn't bounded. Enter  $\frac{n^2-3}{n-1}$  {n 0} and press [BOUNDS1] to get the results {3 +} and 0 (for a set that isn't bounded) in 19 seconds.

But the program isn't successful in all cases. For example it will not work for the set  $\{e^{-(n^2)} \{n -\}\}$ , which is bounded. The reason is that TABVAR protests with "Not reducible to a rational expression", *though the HP49G has all necessary things to make a variation table of the corresponding function  $e^{-(x^2)}$ .*

If you use TABVAR with SIN(X) then you get the variation table:

-	?	-	+0	-	$-\frac{1}{2}$	+	0	+	$\frac{1}{2}$	-	-0	?	+	X
?	?	0			-1	0		1	0		?	?		F



# Sequences, series and limits with the HP49G - Part 1

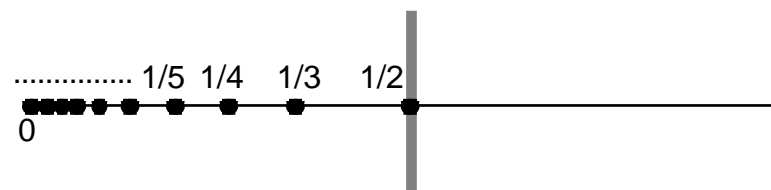
It shows that TABVAR has found what happens between  $-\infty$  and  $+\infty$ . It also shows that TABVAR can't tell you what happens outside this interval. (Or does this result mean "repeat what found between  $-\infty$  and  $+\infty$  endlessly?") Anyway, at least we know for sure what happens between  $-\infty$  and  $+\infty$ . If you enter the set  $\{\sin(n) \mid n \in \mathbb{N}\}$  and press [BOUNDS1] then at the moment when the program tries to find the limit of  $\sin(X)$  for  $X = +\infty$ , you first get the error "Bounded var error". At this point you must press [OK] to continue the calculation and get another error "SERIES remainder is  $O(1)$  at order 3". Press again [OK] to continue. These two errors that occur are somewhat unusual in the sense that they can't be caught with an IFERR THEN ELSE END clause! They are some kind of warnings which are displayed to you, so that you know what happened, but otherwise don't necessarily stop the calculation. The result of this operation is a  $?$ , which on the HP49G means an undefined result. When the program BOUNDS ends, it returns  $\{? \}$  and  $?$  to tell you that it can't determine if the set is bounded. Now, if we knew what the question marks in the variation table want to tell us, if we knew what the warnings want to tell us, then perhaps we could face such cases and handle them appropriately. But while the makers of the CAS keep silent we must just accept that we can't use the whole power of the HP49G, because the makers feel more comfortable answering one question at a time and telling us the truth only when we ask.

Let's go on with sets. An one-dimensional set is bounded when an upper bound  $K$  and a lower bound  $k$  can be given, such that for the coordinate  $x_i$  of an arbitrary point of the set the relation holds:  $k \leq x_i \leq K$ . Any number greater than  $K$  is also an upper bound of the set, and any number less than  $k$  is also a lower bound of the set.

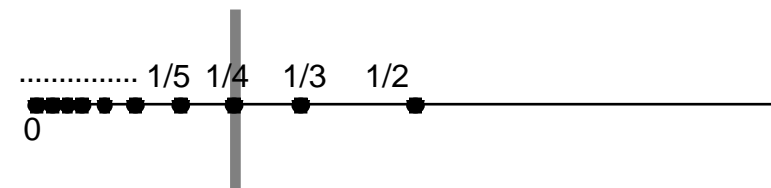
For example all points between 0 and 1 on your ruler are such a (continuous) bounded set. Note that bounds are not limits (while limits are bounds).

Let's now take a look at condensation points. Consider a set with

points  $P_n$  having the coordinates  $\frac{1}{n}$ , with  $n = 1, 2, 3, \dots$ . This set is bounded. The upper bound is  $G = 1$  while the lower bound is  $g = 0$ . This set is bounded but endless, it has infinite number of members but they all fit between 0 and 1. (So I can't be a member of this set since my physical dimensions don't allow me to fit in such a small interval. ;-)) If we chop this interval in two equal parts, then at the left part we have again an infinite number of these points.



If we cut the left part again in two equal parts, then again the left part has an infinite number of points. We can go on forever dividing each new left part into two equal subparts. The left part will always have an



infinite number of points. Each of these parts, is an interval that begins with 0 and has a length of  $\frac{1}{2^k}$  after  $k$  chops and it contains an infinite number of points of our set. By means of these nested intervals we just catch the condensation point of our set. Its name implies its meaning. In any arbitrary small neighbourhood of the condensation point, a neighbourhood given by an arbitrary small positive number  $\epsilon$ , there is an infinite number of points of the set. That means, go as near to the condensation point as you like (without reaching it), you will still find an infinite number of members of the considered set, lying

# Sequences, series and limits with the HP49G - Part 1

between you and the condensation point itself. (Condensation points are also not limits, we'll see the difference some pages later.) In this example we also see that though 0 is a condensation point, it does not belong to our set.

Let's have an additional example. Consider the set  $M$  with points  $P_n$  with a coordinate  $x_n$  given by  $(-1)^n \frac{n+3}{2n}$ , where  $n = 1, 2, 3, \dots$ . The set is infinite, it has an endless number of elements, but it is bounded. It has an upper bound  $G = \frac{5}{4}$  and a lower bound  $g = -2$ , as the following table implies. (This is not a prove. We use it only for guessing.)

n	1	2	3	4	5	6	7	8	9	10
$x_n$	-2	$\frac{5}{4}$	-1	$\frac{7}{8}$	$-\frac{4}{5}$	$\frac{3}{4}$	$-\frac{5}{7}$	$\frac{11}{16}$	$-\frac{2}{3}$	$\frac{13}{20}$

How could the HP49G help us here? If we enter the set  $(-1)^n \frac{n+3}{2n} \{n \geq 1\}$  and press BOUNDS, then the HP49G errors out with "Parameters not allowed" when it tries to find the variation table of the corresponding function  $(-1)^x \frac{X+3}{2X}$ . Now the question is where the HP49G sees parameters. This is a single variable function with no parameters at all. Even if we assume  $X \geq 1$  first, the same error happens again. What can we do to handle such cases? Well, we consider the factor  $(-1)^n$ , which alternates between -1 and 1 if  $n$  is integer going from 1 to  $+\infty$ . The lower bound is assumed to be -2 for  $n=1$ . If all of the negative members prove to be greater than or equal to -2, then we are half the way through. In real mode enter  $n$  and then UNASSUME (to start with a "clean"  $n$ ). Enter  $n \geq 1$  and then ASSUME. Enter  $-1 \frac{n+3}{2n} - 2$  and EXPAND. We use -1 instead of

$(-1)^n$  because we check if all *negative* coordinates are greater than or equal to -2. The result is 1, which shows that the inequality

$(-1)^n \frac{n+3}{2n} - 2$  is true for any  $n \geq 1$  that turns  $(-1)^n$  to -1. Enter  $n$

and UNASSUME again. Now, the upper bound is assumed to be  $\frac{5}{4}$  for  $n=2$ . If all of the positive members prove to be less than or equal to  $\frac{5}{4}$ , then we won. Enter  $n \geq 2$  and then ASSUME. Enter  $1 \frac{n+3}{2n} - \frac{5}{4}$  and press EXPAND. The result is again 1. But is that the proof? Not exactly. We just found that *one* lower bound in -2 and *one* upper bound is  $\frac{5}{4}$ . We don't know if there is some lower bound greater than

-2 and we also don't know if there is some upper bound less than  $\frac{5}{4}$ .

If we could prove this, then we could say that these are the bounds of the set. But wait! Couldn't we use the program BOUNDS separately for each possible outcome of  $(-1)^n \frac{n+3}{2n}$ , that is for the sets

$-1 \frac{n+3}{2n}$  and  $1 \frac{n+3}{2n}$ ? Let's try. Enter  $-1 \frac{n+3}{2n} \{n \geq 1\}$

and press BOUNDS. The results are  $-2 \frac{-1}{2}$  and 1. Oh nice! we

just got the lower bound -2. Now enter  $1 \frac{n+3}{2n} \{n \geq 1\}$  and

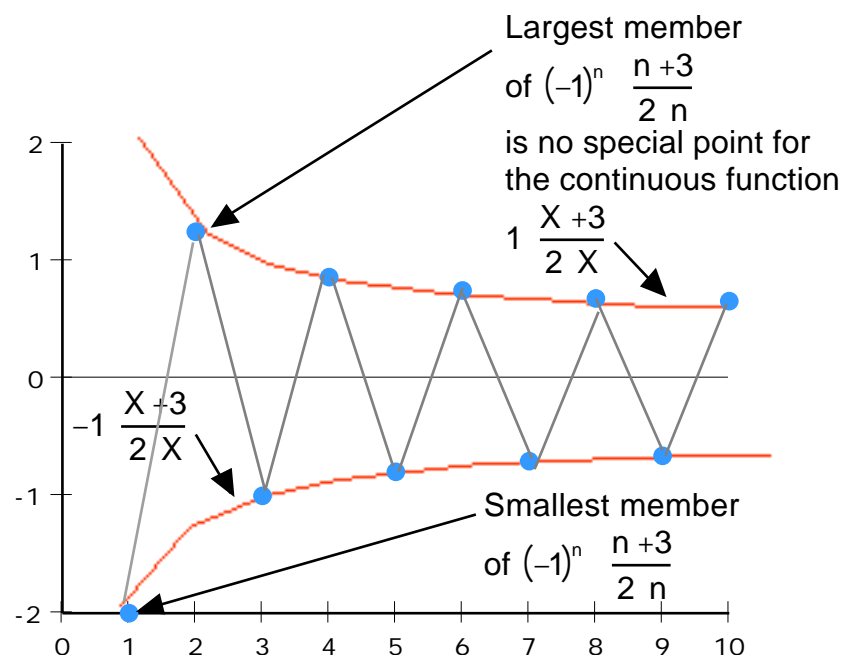
press BOUNDS. The results are  $-2 \frac{-1}{2}$  and 1. Oh no! Where is

$\frac{5}{4}$ ? Do, you see what happened (?). The set  $-1 \frac{n+3}{2n} \{n \geq 1\}$

is "alternating". Because of this, it has a "positive peak" at  $n = 2$ . But

# Sequences, series and limits with the HP49G - Part 1

the two sets  $-1 \frac{n+3}{2n} \{n \geq 1\}$  and  $1 \frac{n+3}{2n} \{n \geq 1\}$  are "smooth". Because of this the "peak" at  $n=2$  disappears. The picture below demonstrates this.



If you remember, we wrote code in the program BOUNDS1 that adds the starting and ending coordinates and members of the set to the variation table. Now, the second time we used the set

$1 \frac{n+3}{2n} \{n \geq 1\}$ , the program added  $\{1 \ 2\}$  for  $n=1$  (start),

while the original set  $(-1)^n \frac{n+3}{2n} \{n \geq 1\}$  has its first (and biggest) positive member for  $n=2$ . This member is  $\frac{5}{4}$ . Use DROP2

to get rid of the two last results, enter  $1 \frac{n+3}{2n} \{n \geq 2\}$  and press BOUNDS again. Now you get the results  $\frac{1}{2}$   $\frac{5}{4}$  and 1.

Hurrah! Let's put all that in a program. We must:

- 1) Find any possible outcome of such alternating factors like  $(-1)^n$
- 2) Build up partial cases of the set for each of the possible outcomes.
- 3) Find what the value of  $n$  is for the first element of each partial case.
- 4) Use BOUNDS for each partial case.
- 5) Get the smallest of the smallest and the greatest of the greatest bounds.

For the sub task (1), we can use the following procedure. We transform the whole general member from an algebraic object to a list using  $\rightarrow$ LST. Then, we check each object for being a  $\wedge$  (the power function). If it isn't we just evaluate. But if it is, we check to see if the exponent depends linearly on the coordinate, and if the base is 1 or -1. If these conditions aren't true, then we simply evaluate. If the conditions are true, we find all possible outcomes and build up lists with replacement patterns that will be used to build up all special sub cases.

For the sub task (2), we can use MATCH to build all possible sub cases of the set.

For the sub task (3), we can use the following: When we found all possible outcomes of the alternating factor, we found also the corresponding values of  $n$ . If we hold these values we can use them pair wise for each special sub case, when we use BOUNDS later.

Let's make code for all these things. After the program listing we'll see it's workings more detailed in every part using an example.

## Sequences, series and limits with the HP49G - Part 1

```
<<
OBJ-> DROP
OBJ-> DROP                @explode set denoted as list
EVAL SWAP                 @Evaluate eventual infinities
EVAL SWAP {} {}
PUSH                      @Save current flags
-> genMemb var lo hi      @store in local variables
explist spclist
<<
    "Checking alternating
exponents" 1 DISP        @Message
    genMemb ->LST         @Turn algebraic to RPL list.
    -> gmlst              @store in local variable
    <<
        gmlst 1           @We'll use DOSUBS (again ;-))
        <<
            IF             @If the current command is ^
                {^} OVER POS
            THEN            @then
                IF
                    OVER var @If the exponent is linear
                    LININ    @in coordinate
                    4 PICK   @and the base is
                    1 ==     @either 1
                    5 PICK   @or -1
                    -1 ==
                OR
                IF           @Convert result if it is alg.
                    DUP TYPE @to a value of 0
                    9 ==     @for FALSE (Why can we
                THEN         @do that?)
                    DROP 0
                END
                AND
            THEN             @All conditions met, so
                3 DUPN EVAL
                DUP 3 DISP   @Display what was found.
                'explist' SWAP @add it to local explist
```

```

STO+
END
END
EVAL @Evaluate the command
>> DOSUBS @Do to all objects of 'gmlst'
HEAD 'genMemb' STO @Store eval. gmlst in genMemb
-103 SF @Go to complex mode
"Solving for m

" 1 DISP @Another message
IF
  explist SIZE @If explist contains anything
THEN
  explist 1 @Another DOSUBS
  <<
    DUPDUP var DUP @Substitute var=var+m in
    'm' + = SUBST = @alternating factor.
    var lo = SUBST @Substitute var=lo
    DUP 2 DISP @Display what will be solved
    'm' SOLVE @Solve for m
    EQ-> NIP EXPAND @Get right hand side of sol.
    IF @If the result was m=? which
      DUP ? SAME @means that the base was 1,
    THEN @then drop it and put a 1 on
      DROP 1 @stack.
    END
    DUP 0
    -> expr evexpr @Store in locals
    intval
    <<
      WHILE
        evexpr TYPE @While the eval. expression
        28 @is no integer
      REPEAT @repeat putting incrementing
        expr 'n1' @values of n1 in expr,
        'intval' INCR @expanding expr for this value
        = DUP 3 DISP @of n1
        SUBST EXPAND @and putting the result in

```



# Sequences, series and limits with the HP49G - Part 1

```

    'evexpr' STO    @evexpr
END                @Build up list containing
lo DUP evexpr +    @expr low and upper values
1 -
3 ->LIST 1 ->LIST
'splist' SWAP      @add to specials list
STO+
>>
>>
DOSUBS
END
IF
    spclist SIZE      @If spclist contains something
THEN
    "Building list of
special replacements
"
    1 DISP            @Message
    spclist 1
    << HEAD >> DOSUBS  @Get heads of spclist elements
    {}
    -> exprlist replist @Store in locals
    <<
        lo spclist 1
        << 3 GET >> DOSUBS @Get 3rd. elem. of spclist
                                @elements
    IF
        DUP SIZE 1 >      @If resulting list has more
    THEN
        @than 1 elements, then
        << LCM >> STREAM @Find LCM of elements
    ELSE
        HEAD              @Else take first (and only)
    END
    FOR I
        exprlist 1        @Make list with elements
        <<
            DUP var I = SUBST @and expressions with
            2 ->LIST EXPAND  @which they will be
                                @matched.

```

```

>> DOSUBS
I 2 ->LIST 1 ->LIST @Add the corresponding
                        @low value
    'replist' SWAP STO+ @Add to list of replace.
NEXT
replist
>>
1
<<
    "Replacing, case " NSUB
    + 1 DISP            @Message
    -> repcase
    <<
        genMemb 1 repcase
        HEAD SIZE      @Do for all elements
        FOR I          @in repcase
            repcase HEAD @Display the list for
            I GET DUP 2 DISP @the match and do a match
            MATCH DROP
        NEXT
        var repcase 2 GET @Make a set with matched
        hi 3 ->LIST      @alternating factors.
        2 ->LIST
    >>
>> DOSUBS
ELSE
    genMemb var lo hi    @If spclist is empty
    3 ->LIST 2 ->LIST    @Build up original set
                        @and wrap it in a list.
END
>> POP                  @Restore flags
>>

```

STOre this in SPCASES. We're going to use the code in a bigger program later on. Now the explanations.

Suppose we have the set  $\left((-1)^{n+1} + (-1)^{n+3+2}\right) \frac{n+2}{3n}$ , with

# Sequences, series and limits with the HP49G - Part 1

$n = 1, 2, 3, \dots$ . We enter this set as a list, that is

$$\left( (-1)^{n+1} + (-1)^{n^3+2} \right) \frac{n+2}{3^n} \{n \ 1 \ \}$$

The first part of the program

```
OBJ-> DROP
OBJ-> DROP
EVAL SWAP
EVAL SWAP {} {}
PUSH
-> genMemb var lo hi
explist spclist
```

explodes the list that represents the set and evaluates the start and end value of  $n$ . It also enters two empty lists that we need later for accumulating such factors like  $(-1)^n$  and the special sub expressions that we are going to replace. It also uses the command PUSH, which stores all current flags and the current working directory in the list ENVSTACK in CASDIR. We do that because we are going to change settings later in the program and we should restore the user's settings when the program finishes. Then the part

```
"Checking alternating
exponents" 1 DISP
genMemb ->LST
-> gmlst
```

displays a message and converts the algebraic object that represents the general member of the set to a sequence of RPL objects enclosed in a list. We do that because we want to check each object for being a  $^$  and act accordingly if it is. At this point the stack contains the list:

$$\{1 \ - \ n \ 1 \ + \ ^ \ \dots \ / \ \}$$

Notice that the - at the second place in the list is not the operator for subtraction but rather the negation NEG. This list is stored in the local

variable gmlst. Then we have:

```
gmlst 1
<<
....
>> DOSUBS
```

The part between << and >> is what will be applied to each object of the list. First we check if the object is the power function  $^$ .

```
IF
{^} OVER POS
THEN
...
```

If it is, then we do:

```
IF
OVER
LININ
4 PICK
1 ==
5 PICK
-1 ==
OR
```

That is we check if the variable, the index of the set is only linearly present in the exponent. We do that because if the exponent isn't linear in the variable of the set, then, well then it can be hard to determine how many different values the term can have. For example  $(-1)^n$  with  $n = 1, 2, \dots$  can be either 1 or -1. But for  $(-1)^{\frac{n^2-3n}{2}}$  it gets harder or even impossible to say how many different values this can have. (Infinite? Are there any other exponents that do the same? And what about periodic functions? We will see later on ;-))

We also check if the base is 1 or -1, because for other values it

# Sequences, series and limits with the HP49G - Part 1

doesn't make big sense to start checking how many and what outcomes the sub expression may have. Consider for example  $\frac{3}{2}^n$ .

Then comes the part:

```
IF
  DUP TYPE
  9 ==
THEN
  DROP 0
END
AND
```

Why do we determine if the result of the comparisons of the base with 1 and -1 is of type 9 (algebraic)? Well, consider the case where the base contains some formal variable (variable with no value). Then the comparisons with 1 and -1 would return `base == 1` and `base == -1` because the base couldn't be evaluated in order to be compared with the two numbers. The following OR would return `base == 1 OR base == -1`. This can't be used for the THEN-part of the IF-THEN-END clause. If you for example try the code `..IF X 1 == THEN "OK" END..` and there is nothing stored in variable X, then the HP49G will say to you: "THEN Error: Undefined Name" meaning that it can't determine if `X == 1` because it doesn't know what the value of X is. If we get an algebraic as result of the comparisons and the following OR, we can be sure that the base was neither 1 nor -1, so we DROP this result and put 0 on the stack. If the result was not an algebraic, then we keep it. At this point there can be two possible results, a 1 for "Yes, the base was 1 or -1" or a 0 for "No, the base is something else". We AND this result with the result regarding linear dependency of the exponent on the set index.

```
THEN
  3 DUPN EVAL
  DUP 3 DISP
  'explist' SWAP
  STO+
END
END
```

If all conditions were met, then we make a copy of the three objects in stack levels 1,2 and three using the command DUPN. These are:

```
3: theBase
2: theExponent
1: theFunction ^
```

Then we EVAL and we get the Object `theBasetheExponent` on stack level 1, which we DUPLICATE and display on line 3 of the screen, so that the user says "Hurrah, we found something to work with". We then put the 'explist' on the stack, SWAP and use STO+ to add the found algebraics  $(-1)^{n+2}$  and  $(-1)^{3+2}$  to explist. After that we:

```
EVAL
>> DOSUBS
```

so that the partial RPL sequence gets transformed to the corresponding algebraic object, which will be used by the next pass of DOSUBS eventually adding another object of the form  $1^{\text{linearFunctionOfn}}$  or  $(-1)^{\text{linearFunctionOfn}}$  and return another algebraic object, and so on until the whole RPL sequence has been completely examined.

The next thing that we do is:

```
HEAD 'genMemb' STO
```

and you might ask yourself why we store the completely evaluated RPL sequence in 'genMemb', since it looks the same like what we already have stored there. Well, the reason for doing that is that we

# Sequences, series and limits with the HP49G - Part 1

will use this later on as argument for `MATCH` and sometimes this commands shows an unexpected behaviour. To understand this better let's have an example. Enter  $\text{SIN}(X)$ , then  $\text{COS}(X)$ , then press `[+]`, then enter 1 and press `[+]` again. You have now  $\text{SIN}(X) + \text{COS}(X) + 1$  on the stack. Now enter the list  $\{\text{COS}(X) + 1 \text{ A}\}$  and use `MATCH` to match the pattern  $\text{COS}(X) + 1$  with A. Would you expect to get  $\text{SIN}(X) + \text{A}$ ? I would! But instead of this you get the expression  $\text{SIN}(X) + \text{COS}(X) + 1$  and a 0 on stack level 1, which tells that the pattern  $\text{COS}(X) + 1$  isn't contained in  $\text{SIN}(X) + \text{COS}(X) + 1$ . Do the same again, enter  $\text{SIN}(X)$ , then  $\text{COS}(X)$ , then press `[+]`, then enter 1 and press `[+]`. Now press `EXPAND`. The algebraic  $\text{SIN}(X) + \text{COS}(X) + 1$  just looks the same like before. Re-enter  $\{\text{COS}(X) + 1 \text{ A}\}$ , press `MATCH` and see:  $\text{SIN}(X) + \text{A}$ . Strange? What is the difference between the `EXPANDED` and the not `EXPANDED` form of  $\text{SIN}(X) + \text{COS}(X) + 1$ ? They look the same. Well, let's take a closer look. Set flag -53. This is the flag that let's the HP49G show all existing parentheses in algebraic objects, when they are not shown in pretty print. Now do the same a third time, enter  $\text{SIN}(X)$ , then  $\text{COS}(X)$ , then press `[+]`, then enter 1 and press `[+]`. (It's getting monotonous but soon the monotony will break.) Press `[blue-shift]` and then `[arrow-down]` to get the algebraic  $\text{SIN}(X) + \text{COS}(X) + 1$  in the command line editor. You see that the HP49G put  $\text{SIN}(X) + \text{COS}(X)$  in a pair of invisible parentheses, so that the whole object is in reality  $(\text{SIN}(X) + \text{COS}(X)) + 1$ . You see now why the first `MATCH` didn't work. Though the object *looked* like  $\text{SIN}(X) + \text{COS}(X) + 1$  in reality it was  $(\text{SIN}(X) + \text{COS}(X)) + 1$ . In this object there is no exact pattern  $\text{COS}(X) + 1$ . Press `[ENTER]`, `EXPAND` and then press `[blue-shift]` and `[arrow-down]` to get the algebraic on the command line again. Now you see  $\text{SIN}(X) + (\text{COS}(X) + 1)$ . The HP49G liked more to have  $\text{COS}(X) + 1$  in parentheses, so the second `MATCH` worked. This phenomenon is the reason for saving the completely evaluated RPL sequence of our

general set member in 'genMemb' again. Since we can't know how the user will enter some algebraic and we also can't know where the HP49G will put parentheses, we can't be sure that using the command

`MATCH` later in the program will give us always the right results.

But we have sub expressions  $(-1)^{n+2}$  and  $(-1)^{n^3+2}$  which were made through sequential evaluation of the corresponding commands and we have the complete general member which was also formed by evaluating all participating RPL objects in the same sequence. Since the evaluation sequence was the same, the "inner form" of the resulting objects will also be the same. And so we make sure that matching will work.

On with our program.

```
-103 SF
```

We set flag -103 for a switch to complex mode, because we are going to solve equations that the HP49G can solve only in complex mode.

Then:

```
"Solving for m
" 1 DISP
```

we just tell the user that the program starts solving some equations.

```
IF
  explist SIZE
```

We then check if `explist` contains anything and if it does:

# Sequences, series and limits with the HP49G - Part 1

```
THEN
  explist 1
  <<
    ..procedure for each object
  >>
  DOSUBS
END
```

we apply a procedure to each object of explist. The objects of explist are all the objects of the form  $1^{\text{linearFunctionOf } n}$  or  $(-1)^{\text{linearFunctionOf } n}$  that we found. Let's see what this procedure does. First

```
DUPDUP var DUP
'm' + = SUBST =
```

we make two copies of the object using DUPDUP. We then built up the equation  $\text{var} = \text{var} + m$ . For example, if the set index was  $n$ , then this equation will be  $n = n + m$ . (Note that because we use  $m$ , the general set member shouldn't contain any  $m$ .) We use SUBST to substitute  $n$  with  $n + m$ . For example the object  $(-1)^{n^{3+2}}$  will be transformed to  $(-1)^{(n+m)^{3+2}}$ . Then we use  $=$  again to form the equation  $(-1)^{n^{3+2}} = (-1)^{(n+m)^{3+2}}$ . Next we do:

```
var lo = SUBST
```

that is we substitute var with the starting value of var. With a starting value of 1 for  $n$  the above equation now would be:

$(-1)^{1^{3+2}} = (-1)^{(1+m)^{3+2}}$ . You see now what we want. If we solve this for  $m$ , then it is like getting the answer to the question, "For what value of the exponent does  $(-1)^{\text{exponent}}$  equal to  $(-1)^{\text{startValue}}$ ?" Or "How many values exist for  $n$ , which cover all possible outcomes of  $(-1)^{n^{3+2}}$ , when  $n$  starts at, say 1, and goes on to infinity?"

On with the program:

```
DUP 2 DISP
'm' SOLVE
```

We show a copy of the equation to be solved to the user and solve it for  $m$ . Then

```
EQ-> NIP EXPAND
```

we split the solution ( $m = \text{something}$ ) to its left and right hand side and throw away the left hand side and expand the right hand side. Then

```
IF
  DUP ? SAME
THEN
  DROP 1
END
```

we check if the solution was a ? (undefined) which means that we solved something like  $1^1 = 1^{1+m}$  with base being equal to 1. This equation has infinite number of solutions, that is  $m$  can be any integer. So we drop the ? and put a 1 on stack. If the solution wasn't ?, then we keep it.

```
DUP 0
```

We make a copy of this and put a 0 on the stack, which will be used as a counting variable. Now,

```
-> expr evexpr intval
```

we store in local variables and

# Sequences, series and limits with the HP49G - Part 1

```
WHILE
  evexpr TYPE
  28
REPEAT
```

keep doing while evexpr (evaluated expression) is not integer. At this point we must look at the expressions of the form  $1^{\text{linearFunctionOfn}}$  or  $(-1)^{\text{linearFunctionOfn}}$ . For example we have detected the existence of  $(-1)^{n^{3+2}}$ . We solve  $(-1)^{1^{3+2}} = (-1)^{(1+m)^{3+2}}$  for  $m$  and find the solution  $m = \frac{2}{3} n1$ . This result is what the HP49G returns, as it

wants to tell us that any quotient of the form  $\frac{2}{3} n1$  will satisfy the equation,  $n1$  being also an integer. (Notice that  $n1$  is returned as a part of the result, so your initial general set member also shouldn't contain  $n1$ .) Now, that means that  $m$  can be  $\frac{2}{3}$  or  $\frac{4}{3}$  or 2 and so on. But we also want  $m$  to be integer itself, because we started with a general set member that contained  $(-1)^{n^{3+2}}$ , and  $n$  was also integer, so the whole exponent has to be integer.

```
REPEAT
  expr 'n1'
  'intval' INCR
  = DUP 3 DISP
  SUBST EXPAND
  'evexpr' STO
END
```

For this reason we start putting integer values in  $n1$  (starting at the start index of the set, and continue until we detect an integer. Notice that INCR is handy for this purpose, as it not only increments 'intval' but also leaves a copy of the incremented value on stack, which we use for creating the equation that is used as one argument of SUBST. Notice

also that we can use STO or INCR for local variables just like we do for global variables. After the WHILE-REPEAT-END loop we have:

```
lo DUP evexpr +
1 -
3 ->LIST
'splist' SWAP
STO+
```

We calculate the start and end of the index range of var, that covers all possible outcomes of objects like  $(-1)^{n^{3+2}}$ . We then make a list consisting of the object, the starting and the found ending index. This list is added to 'splist' so that we can use these data later. We wrap the list of three elements in another list, because adding a list like {a b c} to a list like {otherContents} would simply put the elements of the first list in the second, resulting in {otherContents a b c} and this would require additional work later because we lose the information that a, b and c belong together. But adding a list like {{a b c}} will result in {otherContents {a b c}} which preserves the information that a, b and c belong together.

Now we do

```
IF
  splist SIZE
THEN
```

which checks if splist contains something and if it does

```
"Building list of
special replacements
"
1 DISP
```

# Sequences, series and limits with the HP49G - Part 1

we start making a list that contains what we need to MATCH the general set member with any possible outcome that it can have.

```
spclist 1
<< HEAD >> DOSUBS
{ }
```

We make a new list consisting of the first elements of each sub list of spclist, in our case  $(-1)^{n+2}$  and  $(-1)^{n+3+2}$  and put also an empty list on the stack.

```
-> exprlist replist
```

We store them in local variables and start the local variables procedure.

```
lo spclist 1
<< 3 GET >> DOSUBS
```

We calculate the start and end values of a FOR-NEXT loop that comes a little bit later on. The starting value is in the local variable 'lo', it is the starting index of the set. For the end value we must work a bit more. First we make a list of all third elements of the sub lists of spclist. These are the found values for the possible outcomes of  $(-1)^{n+2}$  and  $(-1)^{n+3+2}$ . For the first algebraic we have a 2 and for the second another 2. That means that  $(-1)^{n+2}$  will take all possible values if we only calculate it for  $n = 1$  and  $n = -1$ . Same for  $(-1)^{n+3+2}$ . So now we have a list {2 2} on the stack.

```
IF
  DUP SIZE 1 >
THEN
  << LCM >> STREAM
ELSE
  HEAD
END
```

Now, if the list contains more than one elements then we must somehow find all possible combinations of results. This is achieved by STREAMing LCM, the function that returns the least common multiple of two integers, over the elements in the above list. If for example we had the two objects  $(-1)^{2n}$  and  $(-1)^n$  then the list would be {1 2}. So we would have 2 possible combinations:

n	$(-1)^{2n}$	$(-1)^n$
1	1	-1
2	1	1

because the third would already be identical to the first. If the list contains only one element then this is already what we want to have. Having the start and end for the loop we do:

```
FOR I
  exprlist 1
  <<
    DUP var I = SUBST EXPAND
    2 ->LIST
  >> DOSUBS
  I 2 ->LIST 1 ->LIST

  'replist' SWAP STO+
NEXT .
```

We apply a DOSUBS procedure to each element of exprlist. This procedure DUPLICates the element and substitutes all values for n, in our case  $n = 1$  and  $n = 2$ . Then it makes a list out of the two objects. In our case when DOSUBS is ready on the first pass of the FOR-NEXT loop we have the list  $\left\{ \left\{ (-1)^{n+2} \quad -1 \right\} \quad \left\{ (-1)^{n+3+2} \quad -1 \right\} \right\}$  which we use to make the list  $\left\{ \left\{ \left\{ (-1)^{n+2} \quad -1 \right\} \quad \left\{ (-1)^{n+3+2} \quad -1 \right\} \right\} \quad 1 \right\}$ , which contains the arguments for MATCH and the corresponding value of

# Sequences, series and limits with the HP49G - Part 1

n. The second pass gives  $\left\{ \left\{ \left\{ (-1)^{n+2} \quad 1 \right\} \left\{ (-1)^{n+3+2} \quad 1 \right\} \right\} \quad 2 \right\}$ . Both lists are added to the local variable 'replist'. When ready we

```
replist
```

return replist to the stack.

```
1
```

After that we put a 1 on the stack that is going to be used by the next DOSUBS. The next DOSUBS procedure starts with

```
"Replacing, case " NSUB
+ 1 DISP
```

which shows messages counting the replacements that are made.

```
-> repcase
```

we store each element of replist, in our case the first list  $\left\{ \left\{ \left\{ (-1)^{n+2} \quad -1 \right\} \left\{ (-1)^{n+3+2} \quad -1 \right\} \right\} \quad 1 \right\}$ , and the second list  $\left\{ \left\{ \left\{ (-1)^{n+2} \quad 1 \right\} \left\{ (-1)^{n+3+2} \quad 1 \right\} \right\} \quad 2 \right\}$  in the local repcase.

```
genMemb 1 repcase
HEAD SIZE
```

We put the general set member on the stack and then we do a FOR-NEXT loop as many times as there are elements in the first sub list of the above lists, in our case two times.

```
FOR I
  repcase HEAD
  I GET DUP 2 DISP
  MATCH DROP
NEXT
```

Each pass takes the  $I^{\text{th}}$  element of the first element of repcase, displays a copy of it on the screen and MATCHES. The flag on stack level 1 is dropped.

```
var repcase 2 GET
hi 3 ->LIST
2 ->LIST
```

Then we make a list out of the MATCHED general set member, the set index and the start and end values. When done we have on the stack:

$$(-1+ -1) \frac{n+2}{3} \{n \quad 1 \quad + \} \quad (1+1) \frac{n+2}{3} \{n \quad 2 \quad + \}$$

Next part is

```
genMemb var lo hi
3 ->LIST 2 ->LIST
```

which runs if the list spclist was empty. It simply reconstructs the original set and puts it in a list.

At the end we use

```
POP
```

to restore the user's settings.



# Sequences, series and limits with the HP49G - Part 1

Now we have the two possible outcomes of  $\left((-1)^{n+1} + (-1)^{n+3+2}\right) \frac{n+2}{3n}$  along with the right variable starting and ending values.

Let's test it. Enter  $\left((-1)^{n+1} + (-1)^{n+3+2}\right) \frac{n+2}{3n} \{n \ 1 \ \}$  and press SPCASES. The result is returned in 30 seconds:

$$(1+(-1)) \frac{n+2}{3n} \{n \ 1 \ + \} \quad (-1+1) \frac{n+2}{3n} \{n \ 2 \ + \}$$

Enter  $(-1)^n (-1)^{\frac{3n-2}{5}} \frac{n-2}{3} \{n \ 1 \ \}$  and press SPCASES.

The result is returned in 70 seconds and is a big list containing all possible outcomes.

We can use BOUNDS1 and SPCASES in another program to find bounds of such sets.

```
<<
SPCASES {} {}
-> setcases extrmlst flaglst
<<
1 setcases SIZE
FOR I
  setcases I GET
  BOUNDS1 'flaglst' SWAP STO+
  'extrmlst' SWAP STO+
NEXT
extrmlst
<< MIN >>
STREAM
extrmlst
<< MAX >>
```

```
STREAM
2 ->LIST
flaglst 1 + @We put an additional 1 in the list
<< AND >> @flaglst so that STREAMing works even
STREAM @if there was only one 1 or 0 in the
>> @list
```

Store this in BOUNDS and test it. Enter again

$$\left((-1)^{n+1} + (-1)^{n+3+2}\right) \frac{n+2}{3n} \{n \ 1 \ \}$$

and press BOUNDS. In about 67 seconds you get the results  $-2 \frac{4}{3}$  and a 1.

Enter  $(-1)^n \frac{n-2}{3n^2} \{n \ 0 \ \}$ . Press BOUNDS to get the results

$$\frac{-1}{24} + \text{and } 0 \text{ in 63 seconds.}$$

We move on now to condensation points. These thingies can be erroneously taken for limits, but they aren't. If we consider again

$$\left((-1)^{n+1} + (-1)^{n+3+2}\right) \frac{n+2}{3n} \text{ with } n = 1, 2, 3, \dots \text{ then we see the}$$

difference. When  $n$  grows to  $+$  the set members alternate between  $-2 \frac{n+2}{3n}$  and  $2 \frac{n+2}{3n}$ . The first approaches  $-\frac{2}{3}$ , the second  $\frac{2}{3}$ .

But we see also the similarities. If we consider alone the first possible outcome of  $\left((-1)^{n+1} + (-1)^{n+3+2}\right) \frac{n+2}{3n}$ , that is  $-2 \frac{n+2}{3n}$ , then we

see that  $\lim_{n \rightarrow +\infty} -2 \frac{n+2}{3n} = -\frac{2}{3}$ . If we consider the second we see that

# Sequences, series and limits with the HP49G - Part 1

$\lim_{n \rightarrow \infty} 2 \frac{n+2}{3n} = \frac{2}{3}$ . So the idea arises to use the same technique as we did for BOUNDS. Find all possible outcomes and then use the command lim to find condensation points. The next program does this. It is very similar to the already programmed things. It searches for special sub expressions that lead to more than one possible outcomes and it finds a separate set for each one of these possible outcomes. Then it uses lim to find the limit of each possible outcome, which is a condensation point for the original set. Because of the similarity we don't comment the program. But here we make an additional improvement (which you can use to improve the code in SPCASES).

Consider for example  $\frac{1}{n} \cos(n)$  where  $n = 1, 2, 3, \dots$ . The periodic factor  $\cos(n)$  can be  $-1$ ,  $0$  or  $1$ , that is we have again three possible outcomes for  $\frac{1}{n} \cos(n)$  which are:  $-\frac{1}{n}$ ,  $0$ ,  $\frac{1}{n}$ . In the next program code is added to handle such periodic functions like COS, SIN or TAN, that "branch" the original set to a finite number of cases. We also added code that filters out eventually duplicates of condensation points. The additional code is bold will be explained after the program listing.

```
<<
  PUSH
  OBJ-> DROP OBJ-> DROP
  SWAP EVAL SWAP EVAL
  {} {} {} RCLVX
  -> genMemb var low high explist perlist spclist xvar
  <<
    "Checking exponents
containing " var + 1 DISP
    genMemb ->LST
    -> genMemblst
  <<
    genMemblst 1
  <<
    IF
```

```
    {^} OVER POS
  THEN
    IF
      OVER var LININ
      4 PICK 1 ==
      5 PICK -1 == OR
    IF
      DUP TYPE 9 ==
    THEN
      DROP 0
    END
    AND
  THEN
    3 DUPN EVAL
    DUP 3 DISP
    'explist' SWAP STO+
  END
  END
  EVAL
>>
DOSUBS
"Checking periodics
containing " var + 1 DISP
OBJ-> DROP ->LST 1
<<
  IF
    {COS SIN TAN}
    OVER POS DUP
  THEN
    IF
      OVER LNAME
    IF
      DUP TYPE 29 ==
    THEN
      AXL
    END
    var POS
  THEN
```

# Sequences, series and limits with the HP49G - Part 1

```

OVER EVAL DUP
3 DISP
'perlist' SWAP STO+
ELSE
DROP
END
END
EVAL
>>
DOSUBS
HEAD 'genMemb' STO
>>
-103 SF
"Solving for m" 1 DISP
IF
  explist SIZE
THEN
  explist 1
  <<
    DUPDUP var DUP 'm' + =
    SUBST = var low = SUBST DUP 2
    DISP 'm' SOLVE EQ-> NIP EXPAND
    DUP 0
    -> expr evexpr intval
    <<
      WHILE
        evexpr TYPE 28
      REPEAT
        expr 'n1' 'intval' INCR = DUP
        3 DISP SUBST EXPAND 'evexpr' STO
      END
      low DUP evexpr + 1 - 3 ->LIST 1 ->LIST
      'spclist' SWAP STO+
    >>
  >>
DOSUBS
END
IF

```

```

perlist SIZE
THEN
  "Solving for m" 1 DISP
  perlist 1
  <<
    DUP OBJ-> DROP2 DUP var low = SUBST
    'm' *
    IF
      4 PICK OBJ->
      UNROT DROP2
      {COS SIN} SWAP POS
    THEN
      2 *
    END
    + = DUP 2 DISP var SOLVE
    EQ-> NIP EXPAND DUP 0
    -> expr evexpr intval
    <<
      WHILE
        evexpr TYPE 28
      REPEAT
        expr 'm' 'intval' INCR = DUP
        3 DISP SUBST EXPAND 'evexpr' STO
      END
      low DUP evexpr + 2 - 3 ->LIST
      1 ->LIST 'spclist' SWAP STO+
    >>
  >>
DOSUBS
END
IF
  spclist SIZE
THEN
  "Building list of
special replacements" 1 DISP
  spclist 1
  << HEAD >> DOSUBS {}
  -> exprlist replist

```

# Sequences, series and limits with the HP49G - Part 1

```

<<
  low spclist 1
  << 3 GET >> DOSUBS
  IF
    DUP SIZE 1 >
  THEN
    << LCM >> STREAM
  ELSE
    HEAD
  END
FOR I
  exprlist 1
  <<
    DUP var I = SUBST 2 ->LIST
  >>
  DOSUBS
  1 ->LIST 'replist' SWAP STO+
NEXT
replist
>>
1
<<
  "Replacing, case " NSUB + 1 DISP
  -> repcase
  <<
    genMemb 1 repcase SIZE
    FOR I
      repcase I GET
      DUP 2 DISP MATCH DROP
    NEXT
  >>
  >>
  DOSUBS
ELSE
  genMemb
END
var high EVAL =
"Finding limits for

```

```

" OVER + 1 DISP
lim POP
xvar STOVX @Store previous vx, as lim can change it!
"Searching duplicates" 1 DISP
IF
  DUP TYPE 5
THEN
  1 ->LIST
END
IF
  DUP SIZE 1 >
THEN
  {}
  -> lst
  <<
    1
    <<
      IF
        EXPAND lst OVER
        POS NOT
      THEN
        'lst' OVER
        STO+
      END
    >>
    DOSUBS
    DROP
    lst
  >>
END POP
>>

```

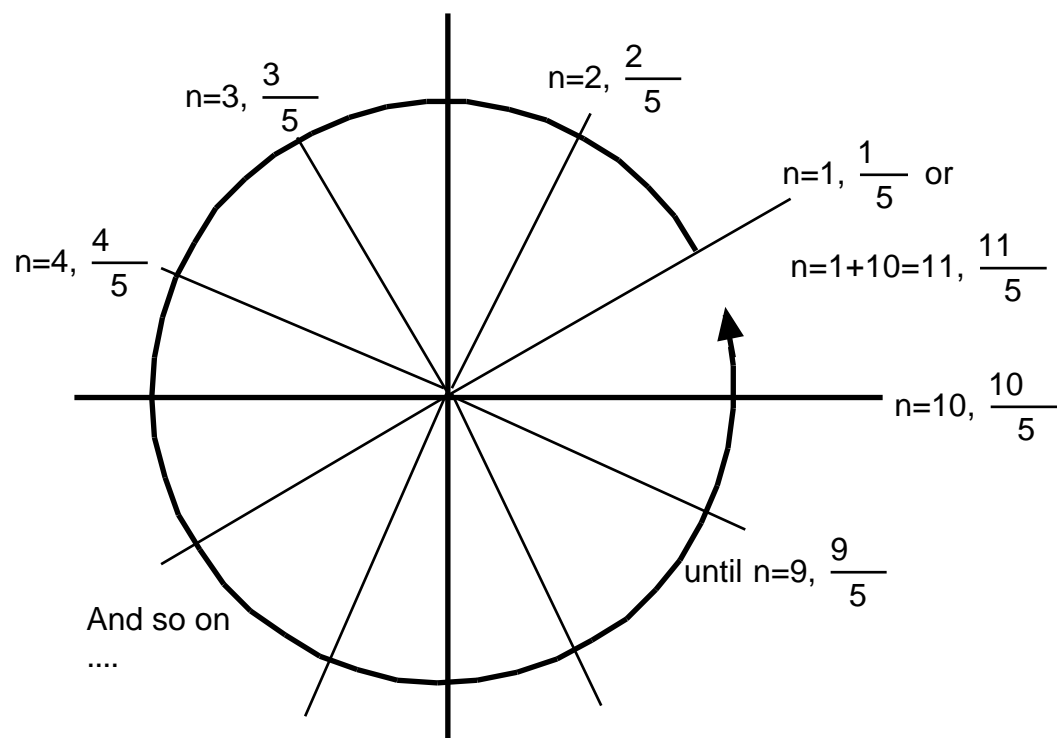
And now the explanation of the additional code. Imagine that we have found  $\cos \frac{n}{5}$  and we want to know how many different values of  $n$  exist, such that starting at  $n = \text{low}$  we get all possible outcomes of this trigonometric expression. It is sufficient to solve

# Sequences, series and limits with the HP49G - Part 1

$$\frac{n}{5} = \frac{\text{low}}{5} + 2m \quad \text{for } n, \text{ where } m \text{ is just another integer.}$$

Solving this is like answering the question: "What can  $n$  be different that  $\text{low}$ , in order for  $\cos \frac{n}{5}$  to be equal to  $\cos \frac{\text{low}}{5}$ ?"

Look at the next picture to understand this better.



That is, because the cosine (and the sine) have a period of  $2\pi$ , there are 10 possible outcomes for  $\cos \frac{n}{5}$ . Starting at  $n = 1$  we have

$\cos \frac{1}{5}$ ,  $\cos \frac{2}{5}$ , ..., until  $\cos \frac{10}{5}$ . The outcome

$\cos \frac{11}{5}$  is already equal to  $\cos \frac{1}{5}$ . We found that the

solution was  $n = \text{low} + 10m$ . The value  $\text{low}$  was 1 and so we have

$n = 1 + 10m$ . Then we start substituting  $m = 1$ ,  $m = 2$  and so on, until  $n$  becomes an integer because  $n$  can't have non-integer values. In this example  $m = 1$  results in  $n = 11$  which is already integer. But

there are cases like for example  $\sin \frac{n}{7} + \frac{1}{8}$

for which we get the solution  $n = \frac{14m+3}{3}$ . Here  $n$  becomes integer for  $m = 3$ .

As already said we sort out eventual duplicates of the found condensation points. The code for this is the second block in bold in the above program. We use some extra features of the HP49G to do this. One of these features has to do with the command **SORT** which sorts lists. This command can also sort lists of lists, provided that the first element in the sub lists is something that can be sorted. For example if you enter the list  $\{\{2 \ 3 \ 1\} \{1 \ 2\} \{4 \ 3\}\}$  and press **SORT**, the result will be  $\{\{1 \ 2\} \{2 \ 3 \ 1\} \{4 \ 3\}\}$ . To get the command **SORT** on a menu press [blue-shift], [SYMB] and then [F3].

Another nice thing that we use is the command **LNAME** which returns a vector of all variables in an algebraic on stack level 1 and the algebraic itself on stack level 2. If the algebraic contains no variables then the command returns an empty list. (Why not empty vector?)

# Sequences, series and limits with the HP49G - Part 1

And a remark about saving the variable VX and restoring it later. Sometimes the function lim will replace your current VX with the variable for which you find a limit. Because we are trying to make programs that leave the user environment untouched, and because we know that lim may change VX, we recall it using RCLVX, do our job, and the restore it using STOVX.

And now here cometh the time of the truth. We test the program. Store it in CONDENSPT and let's go. Enter again our old friend

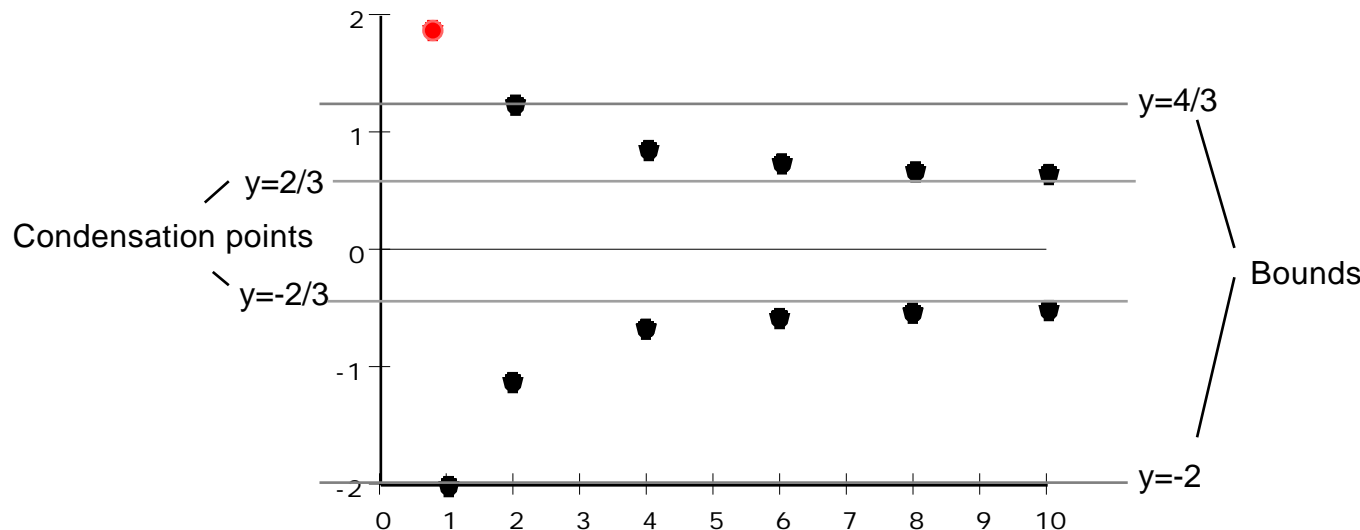
$$\left((-1)^{n+1} + (-1)^{n+3+2}\right) \frac{n+2}{3n} \{n \ 1 \ }$$

and press CONDENSPT.

In about 39 seconds you get  $-\frac{2}{3} \ \frac{2}{3}$ . So, the old friend is a

bounded set with bounds  $-2 \ \frac{4}{3}$  and has the two condensation

points  $-\frac{2}{3} \ \frac{2}{3}$ . Interesting to see this on a picture. Enter the set again and press SPCASES. The result is the list:



$$(-1+ -1) \frac{n+2}{3n} \{n \ 1 \ } \quad (1+1) \frac{n+2}{3n} \{n \ 2 \ }$$

Do << HEAD >> DOSUBS to get  $(-1+ -1) \frac{n+2}{3n} \quad (1+1) \frac{n+2}{3n}$

and then STEQ to store the two equations in EQ for plotting. Now, I have heard many times complains about the missing capability of the HP49G to plot only points of some sequence without curves and lines. But the available plot types are so flexible that we can use them for plots quite different than we do the usual way. Press [blue-shift], hold it pressed and press [F4]. In the PLOT-SETUP screen choose plot type Function, enter n as Indep: and uncheck the option \_Connect, so that you get only points but no lines between them. Now press [blue-shift], hold it pressed and press [F2] to go to the PLOT WINDOW - FUNCTION. Enter H-View from -1 to 10 and V-View from -2 to 2. Enter 1 for Low, 10 for High, 1 for Step and uncheck the option \_Pixels. These settings will let the HP49G draw points starting at n=1 and going up to n=10 in steps of 1. Press [F6] and wait until the plot is done. The screen now must be like the picture below, of course

without the red point. This point doesn't belong to the plot and it appears only because we also plotted the second case

$$(1+1) \frac{n+2}{3n} \text{ from 1 to 10,}$$

though it should be from 2 to 10. You see how the set

# Sequences, series and limits with the HP49G - Part 1

$\left((-1)^{n+1} + (-1)^{n^3+2}\right) \frac{n+2}{3n} \{n \geq 1\}$  behaves. All points are between  $-2$  and  $\frac{4}{3}$ . There are infinite points getting closer and closer to  $\frac{2}{3}$  and infinite points getting closer and closer to  $-\frac{2}{3}$  as  $n$  goes to  $+\infty$ .

Another example. Enter:

$$\frac{(-1)^n (2n-3) \cos \frac{n}{3} - \frac{7}{5}}{5n} \{n \geq 1\}$$

and press CONDESPT. The HP49G rattles and roles and after 65 seconds it wins the rock'n'condence dancing competition ending its performance with:

$$-\frac{2 \cos \frac{4}{21}}{5} - \frac{2 \cos \frac{32}{21}}{5} - \frac{2 \cos \frac{6}{7}}{5}$$

The interested reader may now add such code to SPCASES and BOUNDS1 that they also handle such trigonometric expressions. This is already done in the programs that come with this documents, so lucky guys you don't need to type. But - there's always a big but - there are many shortcomings in these programs. What if the index isn't in the exponent of 1 or -1? Will the programs work or error? What if the trigonometric expressions just don't branch to a finite number of special subsets? What will happen if we have  $\sin(n^2)$  in our set? How can we then prevent the programs from going insane while they are trying to win the dancing competition? Think about that in your spare time and search for solutions. Study the code of the programs

BOUNDS, BOUNDS1, SPCASES, CONDENSPT and exercise critics and new ideas. The HP49G has all needed commands to make the programs better and more robust. And when you are ready, post your excellent thoughts, to help Trabakoulas find the condensation points of the cows. ;-)

One remark about debugging. When you want to debug the programs then there is a potential problem about such commands like DOSUBS, DOLIST and so on. Consider the following code:

```
.....
2
<</>>
DOSUBS
....
```

When you debug this pressing SST, you reach at some point the number 2. Pressing SST puts this number on stack level 1. Pressing SST again puts the DOSUBS procedure <</>> on the stack. Pressing SST again, applies <</>> to all pairs of the list at once. If the contains for example a 0, then the DOSUBS will error but you will not be able to find what was wrong, especially if the list was huge and the DOSUBS procedure complicated. So, if you want to watch how the DOSUBS procedure is applied to the arguments in the list, you should edit the program and add a HALT at the very beginning of the procedure, which then would be for example << HALT / >>. Now, when you press SST and the procedure is applied to its arguments, then it halts right at the start and subsequent presses of SST just evaluate the commands contained in the procedure one by one for each argument of the list, without running the whole thing at once. So you can find exactly where/when the error occurs inside the DOSUBS procedure.

Let's continue with condensation points. Bolzano and Weierstraß, two great mathematicians have found out that:

**An infinite bounded set has at least one condensation point.**

# Sequences, series and limits with the HP49G - Part 1

The above could be of importance for us. Why? Because if we want to check some set for existence of condensation points, then we could run into troubles if lim can't find a limit. But we can check if it is infinite and at the same time bounded. Then we know that there is at least one condensation point. However this doesn't have to mean that an infinite unbounded set has no condensation points. A simple

example like  $\frac{1}{n} \{n \ 0 \}$  shows that though the set is infinite

and unbounded it does have the condensation point 0. So if we find that a set is infinite and bounded we can immediately say that it has at least one condensation point. If we find that it is infinite and unbounded then we have to try to find its condensation points if there are some.

How could we check if some set is infinite, that is has an infinite number of members (points)?. Our notation of sets could make us think: "Look if n comes from or goes to  $-\infty$  or  $+\infty$  and if so, then the set is infinite." But this doesn't have to be always true. Consider for example  $\{\cos(n) \mid n \in \mathbb{Z}\}$ . It has only two points which are  $-1$  and  $1$ . But we can use the program SPCASES, when we have sets containing such expressions that SPCASES can handle. If there is a finite number of special outcomes, then the code will hopefully find them and replace them with expressions that are evaluable to numbers. (If there is an infinite number of cases then the code will continue happily to find one after the other without noticing that it has just started a job which it can never end. This is the weak point of the programs, but let's hope that someone out there will have an idea.)

When SPCASES finishes, it returns a list of sets that represent the special cases. We can temporarily switch to numeric mode and EVALuate each of these cases. The evaluation will return either an algebraic expression or a number. If it returns an algebraic expression which contains the index of the set, then we check if the index domain contains at least one infinity. If it does, then we know that the set represented by the current special case of the original set is infinite, while if the domain is finite then we know that the set is also finite. If the evaluation it returns a number then the set is finite. When we

finished examining each case, then we can say that the original set is infinite if some of the sets represented by the special cases proved to be infinite.

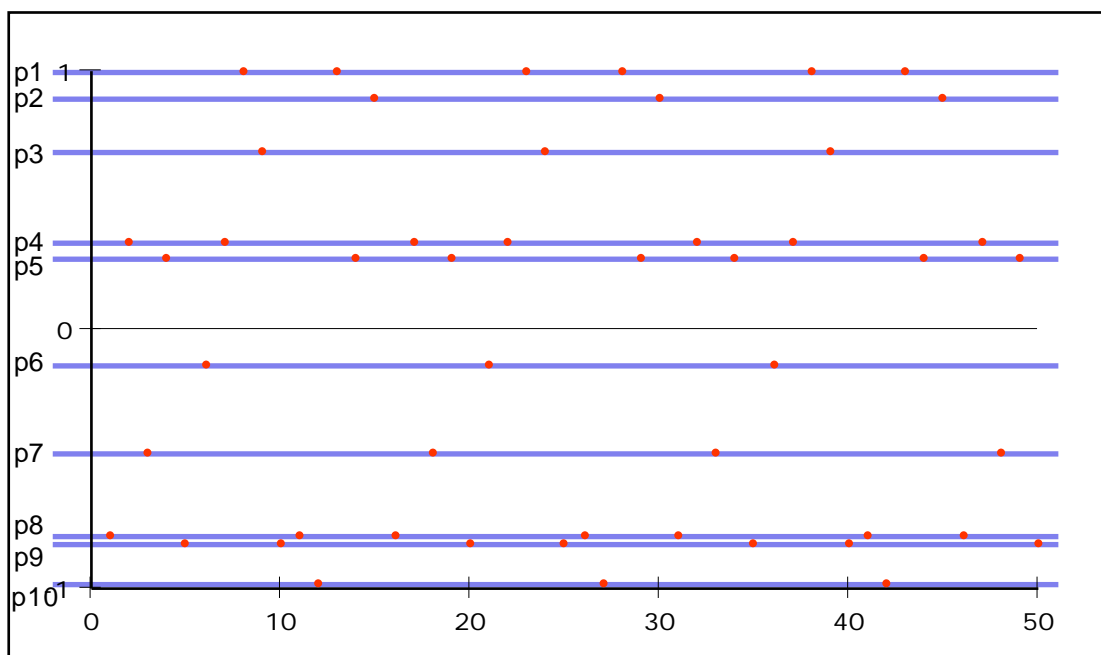
```
<<
  PUSH
  "Checking existence
of special cases" 1 DISP
  SPCASES
  "Checking existence
and domain of index" 1 DISP
  1
  <<
    LCD->          @Return current display as GROB
    {#0h #Eh}
    PICK 3
    0 ->GROB       @Make GROB of the set in pretty print
    REPL ->LCD     @Replace part of the display GROB
                    @with the GROB of the set
    OBJ-> DROP      @explode set
    SWAP -105 SF EVAL @Evaluate in numeric mode
    LNAME -105 CF NIP @Find names and switch to exact
    IF
      DUP TYPE 5    @If result isn't list
    THEN
      OBJ-> HEAD ->LIST @the convert it to list
    END
    OVER HEAD POS   @Find if index is in member
    SWAP OBJ-> 4 ROLL
    DROP 2 SWAP -    @Find how big index domain is
                    @and compare it to
  >>
  DOSUBS            @Do to all special cases
  0 +               @Add a 0 in list of results
  << OR >>         @so that streamed OR always
  STREAM           @works OK
  POP
  >>
```



# Sequences, series and limits with the HP49G - Part 1

Store that in 'ISINF?'. The program takes a set from the stack and returns a 1 if it is infinite or a 0 otherwise. Notice the usage of the commands LCD->, ->GROB and ->LCD for showing messages to Trabakoulas the impatient who wants to know if the set of sheep is infinite. ;-) LCD-> returns what is currently displayed on screen as a GROB on the stack. ->GROB takes something from stack level 2, and a number from stack level 1, and reruns a GROB of that something. If the number on stack level 1 is a 0, then the GROB is in pretty print for algebraic objects. The command ->LCD just takes a GROB from the stack and makes it to the current display on the screen.

Now let's test the program. Enter  $\{n^2 \{n \ 1 \} \}$  and press ISINF?. The calc says 1. (yes) after some seconds. Enter  $\{n^2 \{n \ 1 \ 10\} \}$  and press ISINF?. This time the calc says 0. (no) because the index runs only from 1 to 10.



And a crazier one. Enter  $\text{COS } \frac{n^2}{3} + \frac{3n}{5} - \frac{1}{7} \{n \ 1 \}$

and press ISINF? When I tried this the first time, I thought that it had to be infinite. I expected the COS function to send the quadratic argument to different points as n goes from 1 to infinity. But the HP49G proved me wrong. It said that the set is finite! Let's see if that is true. Enter

$\text{COS } \frac{n^2}{3} + \frac{3n}{5} - \frac{1}{7}$  and press STEQ. Set 'n' as the

independent variable of the plot, set H-View from 0 to 50, set V-View from -1.3 to 1.3 and set Step to 1 unit (not pixel) so that only the points n=1,2,... will be plotted. Also disable the option \_Connect. Let that draw and you see the unexpected. The set has a finite number of points (10) which are repeated as n grows from 1 to infinity! So our methods are not perfect but also not so bad at all, it seems.

Now we can make another program, that checks if the set has any condensation points without actually finding them.

```
<<
-> set
<<
  IF
    set ISINF?
  THEN
    IF
      set BOUNDS NIP
    THEN
      1.
    ELSE
      @In this case there might
      ? @be condensation points.
    END
      @We must use CONDENSPT
    ELSE
      0. @Set is finite, no condensation
    END
      @points
  >>
>>
```

# Sequences, series and limits with the HP49G - Part 1

Store it in HASCNDSPT?. Let's check it in combination with the other program CONDENSPT which calculates the condensation points.

Enter the set  $(-1)^n \sin \frac{2n}{3} - \frac{n-1}{3n+1} \{n-1\}$ , make a

copy of it on stack level 2 and press HASCNDSPT?. The HP49G shakes its brain, shows messages, rattles a lot, and after about 5 minutes (!) it returns a 1, which means that the set has at least one condensation point. Drop the 1 and press CONDENSPT. You just sent the HP49G to the next "Jump in the fire competition", which it accepts with enthusiasm and which it finishes in about 1 minute bringing you the prizes that it won, namely the condensation points

$$-\frac{\sqrt{3}}{6} \quad 0 \quad \frac{\sqrt{3}}{6}.$$

Another example: Enter the crazy little set

$$\cos \frac{n^2}{3} + \frac{3n}{5} - \frac{1}{7} \{n-1\} \text{ again and make a copy of}$$

it on stack level 2. Press HASCNDSPT?. The HP49G needs about 3 minutes to return a 0. The set has no condensation points because it is finite, as we already have seen before. Now, drop the 0 and press CONDENSPT. The result is given to you after 2.5 minutes. (It is a big list.) Our program finds incorrectly that the set has condensation points though it can't have them since it is finite. The problem is that the program CONDENSPT doesn't check to see if the calculated special cases really depend on the index and if the index domain is infinite. There are two ways to correct this. Either we add such code in CONDENSPT, or we add a check at the start of the program:

```
<<
IF
  DUP HASCNDSPT?
  IF
    DUP 1 ==
    SWAP ? SAME OR
  THEN
    @If HASCNDSPT? said
    @1 or ?
    @then put 1 on the stack
>>
```

```
1.
ELSE
0.
END
THEN
PUSH
OBJ-> DROP OBJ-> DROP
SWAP EVAL SWAP EVAL
{} {} {} RCLVX
-> genMemb var low high explist perlist spclist
xvar
<<
..blah blah rest of code of HASCNDSPT?
lst
>>
END POP

ELSE
DROP
{}
END
>>
```

The additional code is bold in the partial listing of CONDENSPT. Let's take a look at the programs until now, and what they do.

**SPCASES** Finds a finite number of special cases of sets that contain powers of 1 and -1 containing the index or periodic functions with arguments that contain the index. Returns a list in which the general member has no more such powers and periodics but the special outcomes instead.

**BOUNDS1** Finds bounds of a set and returns also a 1, if the set is bounded or 0 if it isn't.

**CONDENSPT** Finds condensation points of a set.

# Sequences, series and limits with the HP49G - Part 1

**BOUNDS** Finds bounds of a list of special cases of a set.

**ISINF?** Returns a 1 for infinite sets or a 0 for finite sets.

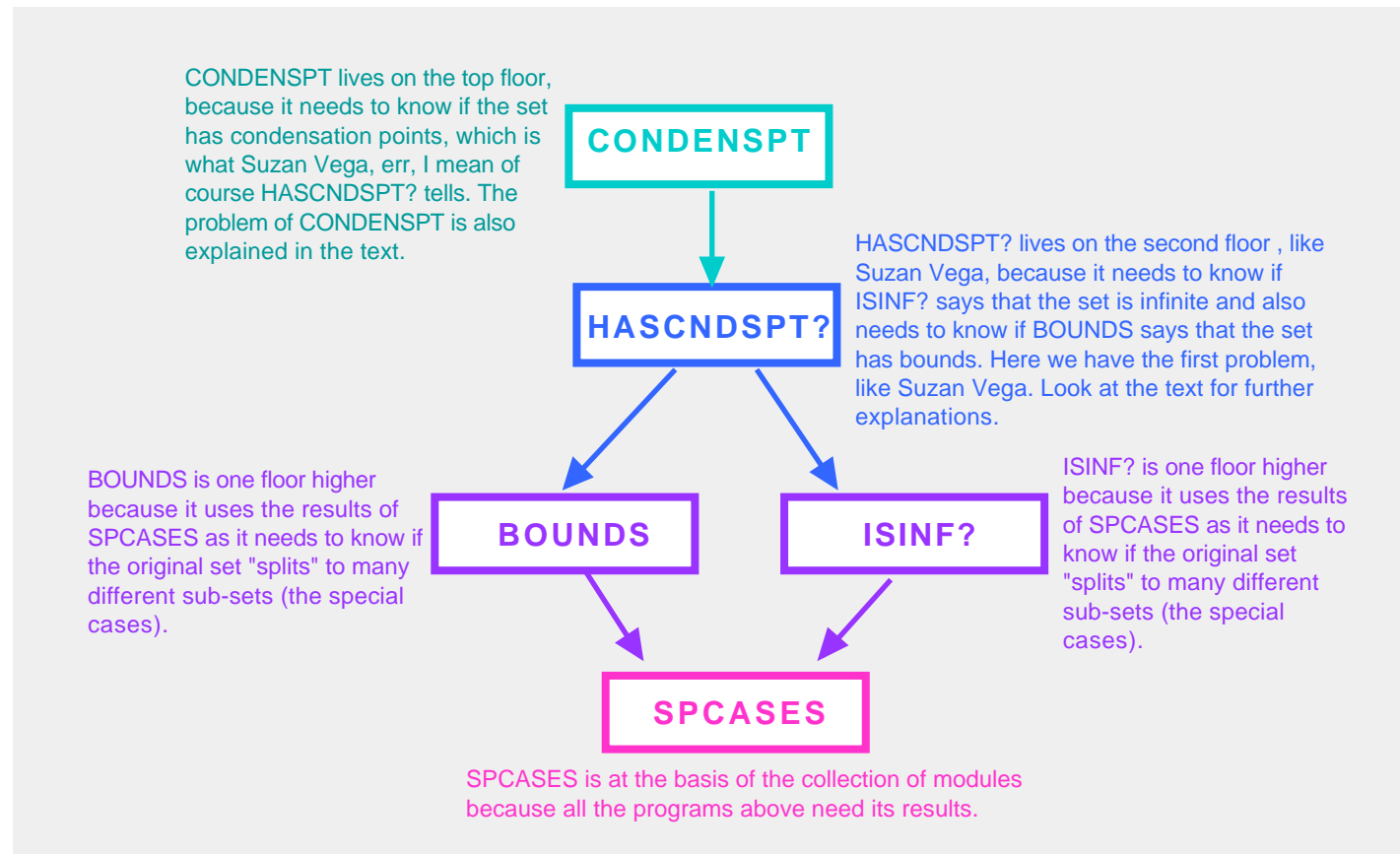
**HASCNDSPT?** Returns a 1 if a set has condensation points or a 0 if it doesn't without actually calculating the condensation points themselves always.

Notice however that these programs are not at all optimised. For example, CONDENSPT uses HASCNDSPT? to check if there are any condensation points and HASCNDSPT? uses SPCASES to find special case. But CONDENSPT itself contains also code that finds special cases. So let's optimise a little bit.

The first thing that we do is "mapping" our mathematical knowledge to a set of functions for the HP49G, trying to give the functions the roles of modules that can call other modules or be called from other modules. Take a look at the picture on the right to understand better how the modules will be organised. Start at the bottom and work your way to the top.

The first problem that we have, is that CONDENSPT as displayed on pages 1-19 to 1-21, contains itself the code of SPCASES. This is

of course not very efficient and wastes memory. If we replace the code with a call to SPCASES then we will save RAM. But if we do that, then the code of SPCASES will run three times when we let CONDENSPT run. This is because CONDENSPT should also contain a call to HASCNDSPT? which in turn contains a call to BOUNDS and a call to ISINF? and these two programs call SPCASES themselves. So let's go to the first floor, there where ISINF? and BOUNDS live, and take a look around. Up to the first floor we have no problems. But when HASCNDSPT? one floor higher calls ISINF? then ISINF? calls SPCASES. SPCASES gives a list with all special cases of the set.



# Sequences, series and limits with the HP49G - Part 1

Then HASCNDSPT? calls BOUNDS which also calls SPCASES and so the same code runs again. We can see that this second run of BOUNDS is not necessary when the programs are called from HASCNDSPT?. But the programs BOUNDS and ISINF? should work also as stand alone programs, which means that each of them must call SPCASES then. We are in a dilemma, it seems. Should both programs BOUNDS and ISINF? contain a call to SPCASES or not? If we only could tell for example BOUNDS "call SPCASES only when you are called by HASCNDSPT? but otherwise not." Well, how about passing some variable value from HASCNDSPT? to BOUNDS and letting BOUNDS determine if it should call SPCASES by checking what this value is? That's a possible method. Let's do that.

First of all, we don't need to modify SPCASES at all, it stays the way it is on pages 1-9 to 1-10.

Then we have ISINF? which should return only a 1 or a 0 if it is directly executed by the user, but in case it is called by Suzan Vega (or by HASCNDSPT?) it should also return the special cases found by SPCASES, because they are needed by Suzan (or by HASCNDSPT?) on the second floor. At this point we make a small excursion to the mysterious local variables which are very very etherious beings, existing like ghosts only when their procedures are running and disappearing otherwise. You know of course that the special syntax

```
-> a b
<<
  a b +
>>
```

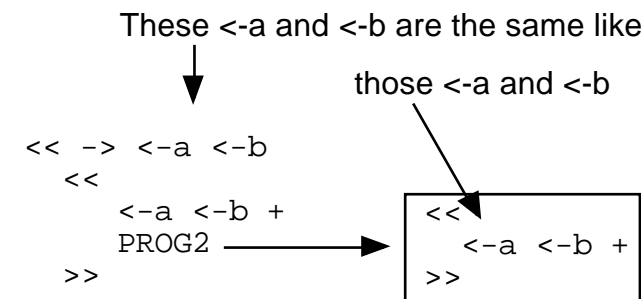
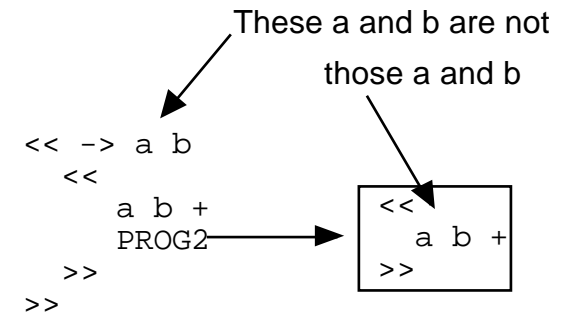
creates two local variables **a**, and **b** that exist only in the bold part of the program lines above. Outside this region no local variables **a** and **b** exist. The local variables are also distinct from any global variable **a** and **b** that might exist in the current path. That means that even if you have **a** and **b** as globals in the current path, evaluating **a** and **b** in the bold part of the program will give you the local and not the global variables. Another thing to be aware of when working with such ghosts is that if some program that defines local variable calls some

other program while the local variables exist, and if the called program contains the same names, then the variables in the called program are *not* the same like the variables in the caller. That means that where the locals exist is defined when the program that contains them is compiled, or in

other words when you press [ENTER] after having typed the program. The program PROG2 in the picture above, was ENTERed and thus compiled at some other time than the program containing the locals **a** and **b** and so the variables **a** and **b** in PROG2 are different entities than those in the other program. (Many different ghosts here, its like being in Great Britain ;-)) This makes us a bit sorrow because we would like to define a local variable in HASCNDSPT? that can be used in ISINF? or in BOUNDS. But to our biggest possible happiness, there are local

variables that do exactly this, they exist also in programs that are compiled to some other time than the compile time of the program that defined the local variables. This kind of local variable is called "compiled local variable".

A compiled local variable is a local variable whose name begins with a <- (arrow left, character 142), like for example the variables **a**, **b** or **Trabakoulas**. These variables exist only while programs



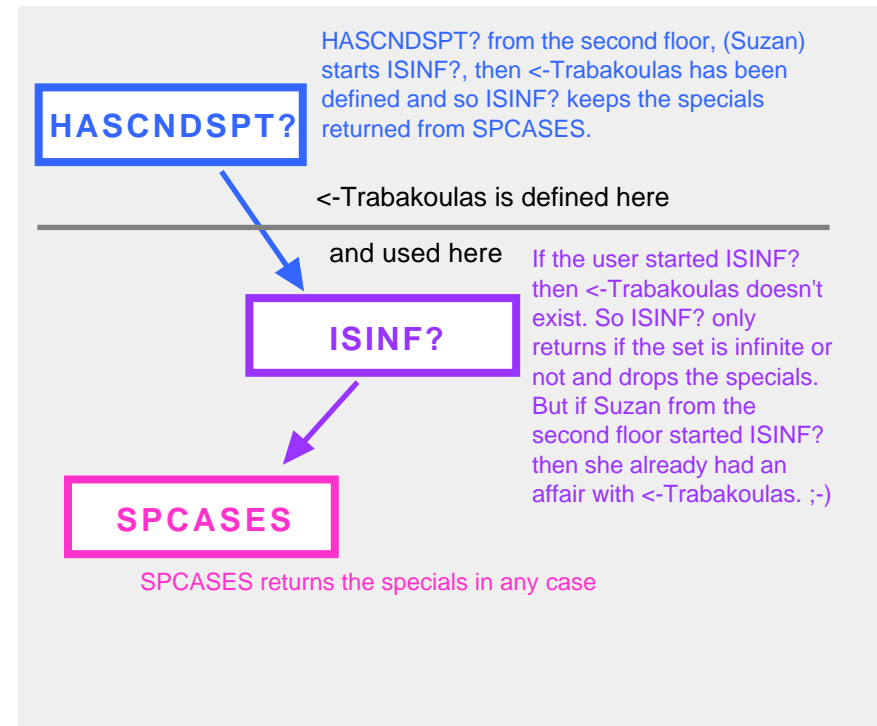
# Sequences, series and limits with the HP49G - Part 1

run and to no other times. But they exist *for all programs*. They exist everywhere and yet appear in no menu, that is they are in the twilight zone!

Now, we know what we should do, in order to let the programs ISINF? and BOUNDS run one way if the user presses their menu key, and another way if they are called from HASCNDSPT? Take a look at the code of the new ISINF?. Notice the difference at the end of the program

```
<<
  PUSH
  "Checking existence
of special cases" 1 DISP
  SPCASES
  "Checking existence
of index and domain of
index in

" 1 DISP DUP 1
<<
  LCD-> { # 0h # 15h}
  PICK3 0 ->GROB REPL
  ->LCD OBJ-> DROP SWAP
  -105 SF EVAL LNAME
  -105 CF NIP
  IF
    DUP TYPE 5
  THEN
    OBJ-> HEAD
    ->LIST
  END
  OVER HEAD POS
  SWAP OBJ-> 4 ROLL
  DROP2 SWAP - SAME
  AND
  >> DOSUBS 0 +
  << OR >> STREAM
  IFERR                                @If <-Trabakoulas doesn't exist
```



```
<-Trabakoulas      @that means that the user
THEN                @started the program and so nip
  NIP                @the special cases.
ELSE                @else
  DROP              @drop poor <-Trabakoulas.
END POP
>>
```

The picture above shows the situation. The same technique we use in BOUNDS. If <-Trabakoulas exists, then we don't call SPCASES. But if not, then we do call BOUNDS. The code of BOUNDS is now:

# Sequences, series and limits with the HP49G - Part 1

```

<<
PUSH
IFERR      @If <-Trabakoulas doesn't exist then
  <-Trabakoulas @the user started BOUNDS. That means
THEN       @call SPCASES
  SPCASES
ELSE       @But if he does exist then
  DROP2    @drop two objects
END
{ } { } -> setcases extrmlst flaglst
<<
  1 setcases SIZE
  FOR I
    setcases I
    GET OBJ-> DROP OBJ->
    DROP EVAL SWAP EVAL
    SWAP -> set var lo hi
    <<
      "Building var. table of
"
      1 DISP LCD-> { # 0h # 7h }
      set 0 ->GROB
      REPL ->LCD set var
      RCLVX = SUBST
      IFERR
        TABVAR
      THEN
        CASE
          ERRN # DE65h ==
          THEN
            lo hi 2 ->LIST OVER DUP 2
            ->LIST 2 ->LIST 0
          END
          ERRN # DE25h ==
          THEN
            -105 SF EVAL XQ TABVAR
          END
        END
      END
    END
  END
  ? DUP 2 ->LIST 0.
  END
  END DROP OBJ->
  << 2 ->LIST >> DOLIST
  "Filtering out +,-,|v,|^"
  1 DISP 1
  <<
    IF
      NSUB 2 MOD NOT
    THEN
      DROP
    END
  >> DOSUBS
  "Adding low" 1 DISP lo PICK3 RCLVX lo =
  lim 2 ->LIST 1 ->LIST +
  "Adding high" 1 DISP hi ROT RCLVX hi = lim
  2 ->LIST 1 ->LIST +
  "Transforming non-
integer to integer " var + 1 DISP 1
  <<
    IF
      DUP HEAD DUPDUP TYPE 9 == SWAP
      ABS      AND
    THEN
      NIP ->NUM DUP FLOOR R->I
      DUP var SWAP = set SWAP SUBST EXPAND 2
      ->LIST SWAP CEIL R->I
      DUP var SWAP = set
      SWAP SUBST EXPAND 2 ->LIST
    ELSE
      DROP
    END
  >> DOSUBS
  "Filtering out " RCLVX + "<" + var + "min
" + 1 DISP 1
  <<
    IF
      DUP HEAD lo <

```

# Sequences, series and limits with the HP49G - Part 1

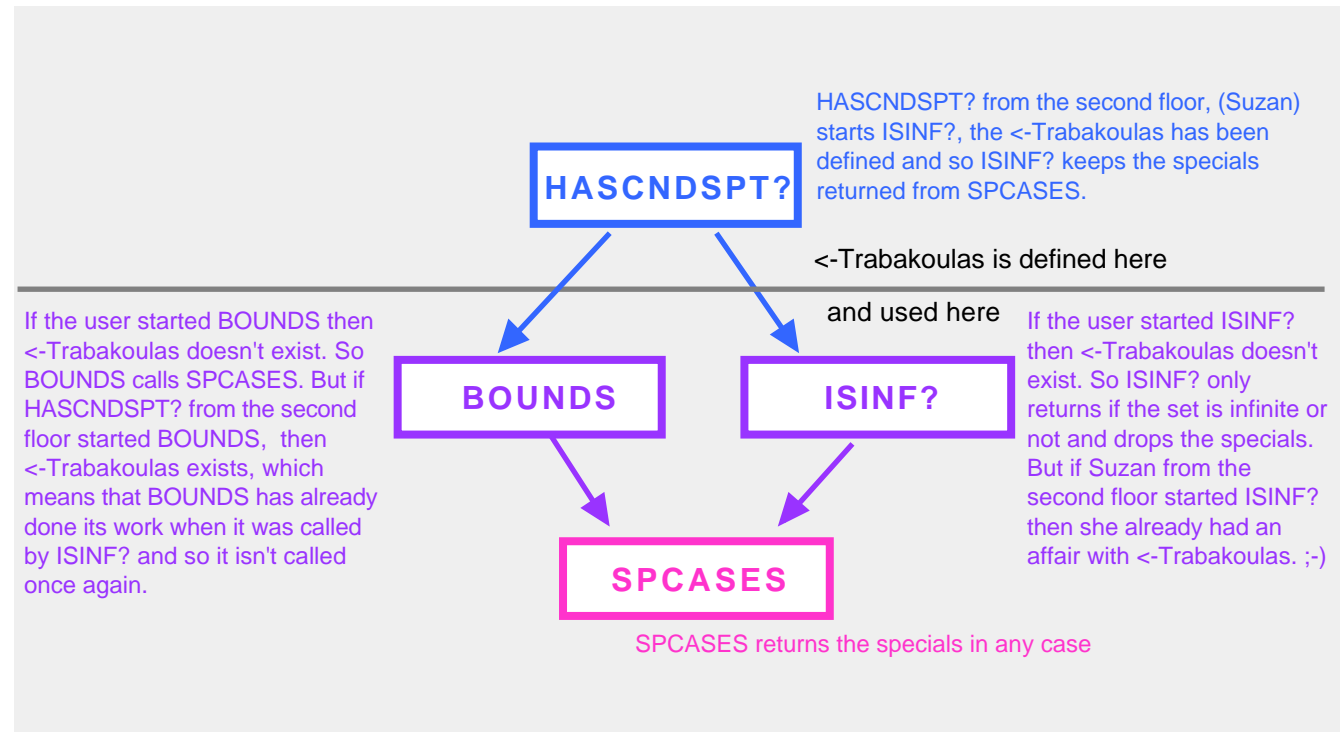
```

    THEN DROP
    END
  >> DOSUBS
    "Filtering out " RCLVX + ">" + var + "max
" + 1 DISP 1
  <<
    IF
      DUP HEAD hi >
    THEN
      DROP
    END
  >> DOSUBS 1
  << 2 GET >> DOSUBS
  "Searching min." 1 DISP DUP
  << MIN EXPAND >>
  STREAM SWAP
  "Searching max."
  1 DISP
  << MAX EXPAND >>
  STREAM DUP2
  2 ->LIST UNROT -
  ABS
  <
  >> 'flaglst'
  SWAP STO+ 'extrmlst'
  SWAP STO+
NEXT
extrmlst
<< MIN >> STREAM
extrmlst
<< MAX >> STREAM 2
->LIST
flaglst 1 +
<< AND >> STREAM
>> POP
>>

```

Trabakoulas is used here to decide if the user or HASCNDSPT? was the the one that started BOUNDS. If HASCNDSPT? started BOUNDS then Trabakoulas exists and so BOUNDS doesn't call SPCASES, because SPCASES has already been called by ISINF? before. Note also that the code of the previously made program BOUNDS1 is now included in BOUNDS and BOUNDS1 is not necessary any more.

The program HASCNDSPT? includes code, that defines the compiled local variable Trabakoulas, so that BOUNDS and ISINF? can distinguish between the user or HASCNDSPT? as callers:



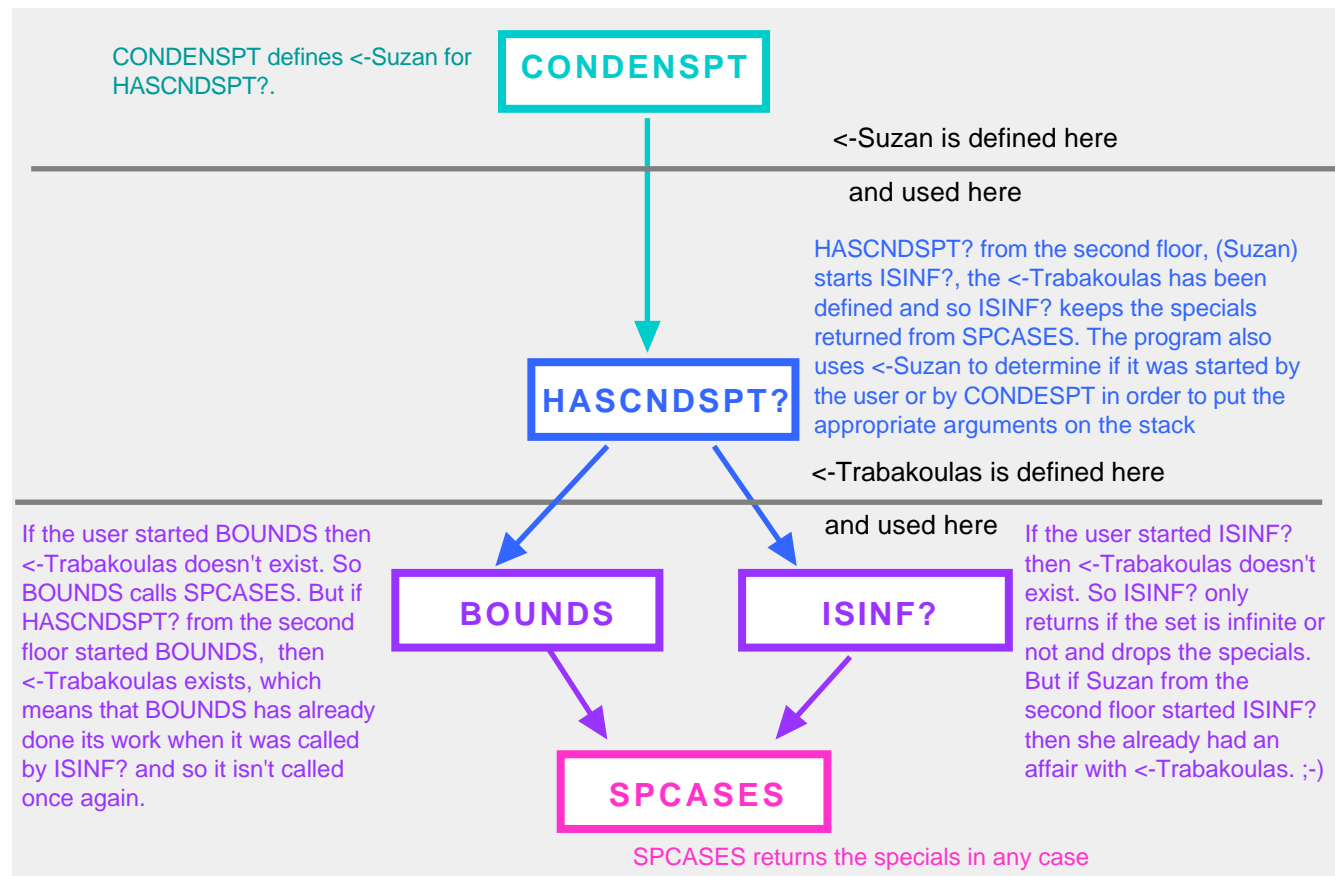
# Sequences, series and limits with the HP49G - Part 1

```
<<
PUSH 1 -> set <-Trabakoulas
<<
  IF
    set ISINF? @ISINF? sees now <-Trabakoulas defined
  THEN
    IFERR
      <-Suzan @If <-Suzan doesn't exist, then the
    THEN @the user has started the program, so
      DUP @make copy of stack level 1
    ELSE @else
      DROP @drop poor
      DUPDUP @<-Suzan
    END
  IF
    BOUNDS @BOUNDS sees
    NIP @<-Trabakoulas
  THEN @defined
    1.
  ELSE
    ?
  END
ELSE
  DROP 0.
END
>> POP
>>
```

As you can see HASCNDSPT? calls Trabakoulas into existence and so when the program calls ISINF? and BOUNDS these called programs will act accordingly.

HASCNDSPT? uses a new local compiled variable Suzan, to determine what arguments will be

put on the stack. Why? Because when the program is started by the user it should only return a 1 (yes) or 0 (no) as answer to the question if there are condensation points. But when CONDENSPT calls the program then HASCNDSPT? returns the special cases which are used further from CONDENSPT. On the next page we have the code of CONDENSPT:





# Sequences, series and limits with the HP49G - Part 1

```

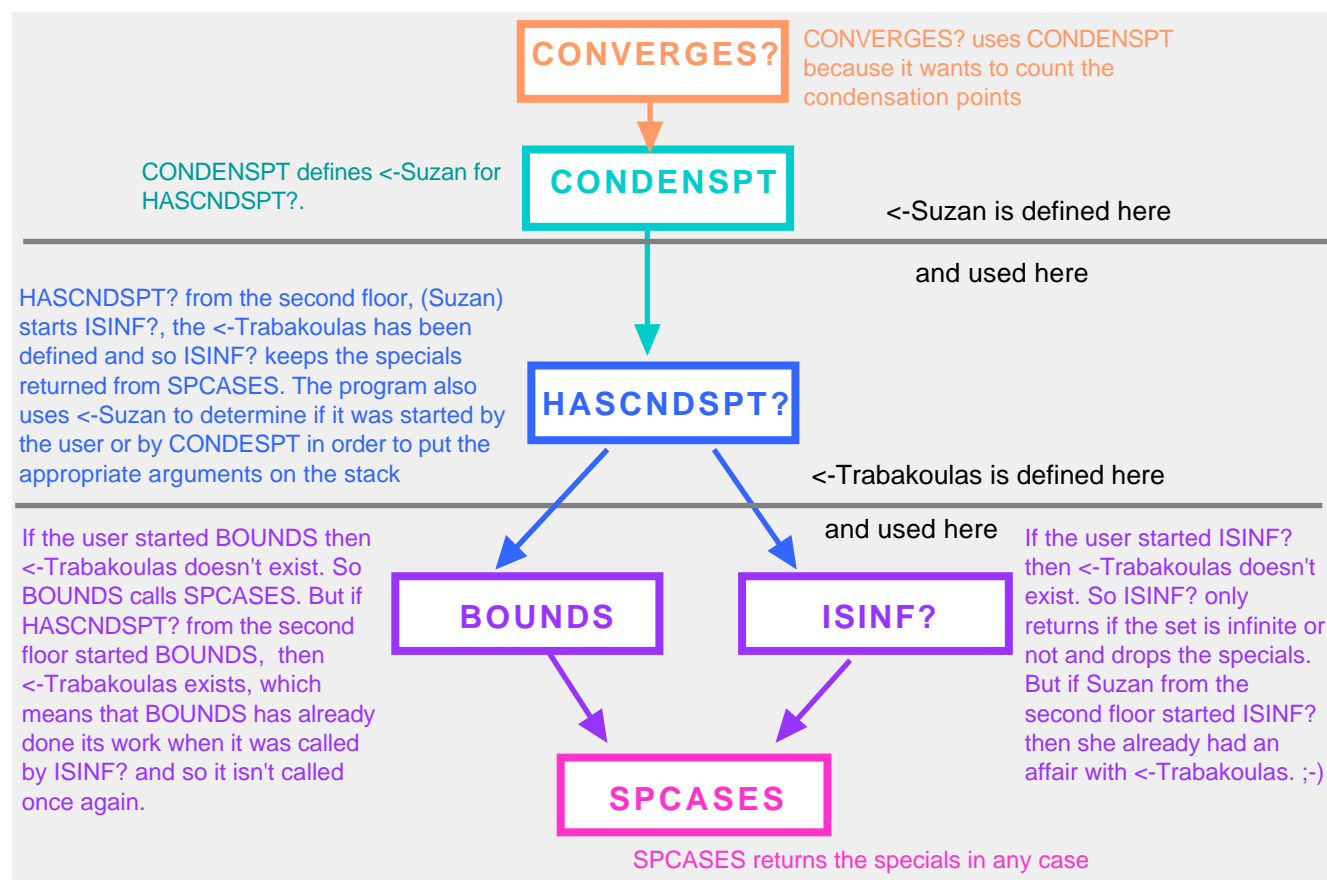
<< PUSH 1 -> <-Suzan
<<
  IF
    IF
      DUP HASCNDSPT? DUP 1. == SWAP ? SAME OR
    THEN
      1.
    ELSE
      0.
    END
  THEN
    SWAP OBJ-> DROP NIP
    OBJ-> DROP NIP EVAL
    RCLVX -> var high xvar
    << 1
      << HEAD >> DOSUBS
      var high EVAL =
      "Finding limits for
"
      OVER + "
"
      + 1 DISP lim xvar
      STOVX "Searching
duplicates" 1 DISP
      IF
        DUP TYPE 5
      THEN
        1 ->LIST
      END
      IF
        DUP SIZE 1 >
      THEN
        { }
        -> lst
        << 1
          <<
            IF
              EXPAND lst

```

```

OVER POS NOT
THEN
  'lst' OVER STO+
END
>> DOSUBS
DROP lst
>>
END
>>
ELSE

```



# Sequences, series and limits with the HP49G - Part 1

```

      DROP { }
    END
  >> POP
>>

```

Now the complete situation of Trabakoulas flirting Suzan looks like in the picture on the bottom of the right column on the previous page. Quite ideal stuff for a mathematical soap opera ;-)

We make another program that answers the question if a point set converges with a 1 (yes) or 0 (no). The mathematics behind this program say that if such a set has one and only one condensation point, then it converges and its condensation point is its limit.

```

<<
  IF
    CONDENSPT SIZE 1 ==
  THEN
    1.
  ELSE
    0.
  END
>>

```

Store it in CONVERGES? Ready for a test. Examine the point set

$(-1)^n \cos \frac{2n}{3} \frac{2n-3}{4n} \{n \ 1 \}$  with all programs.

SPCASES returns:

$$\begin{aligned}
 & -1 \frac{-1}{2} \frac{2n-3}{4n} \{n \ 1 \} \\
 & 1 \frac{-1}{2} \frac{2n-3}{4n} \{n \ 2 \} \\
 & -1 \frac{1}{2} \frac{2n-3}{4n} \{n \ 3 \} \\
 & 1 \frac{-1}{2} \frac{2n-3}{4n} \{n \ 4 \} \\
 & -1 \frac{-1}{2} \frac{2n-3}{4n} \{n \ 5 \} \\
 & 1 \frac{1}{2} \frac{2n-3}{4n} \{n \ 6 \}
 \end{aligned}$$

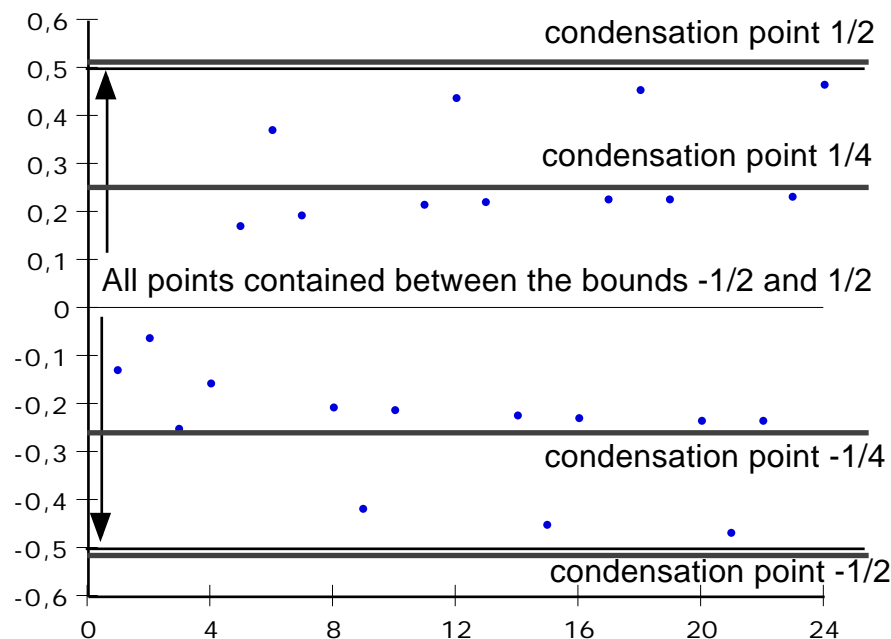
These are the special cases to which the factor  $(-1)^n \cos \frac{2n}{3}$  splits the set. (They are all given twice. Perhaps add code to filter out unnecessary copies?)

ISINF? returns 1 for yes. Indeed if you plot

$(-1)^n \cos \frac{2n}{3} \frac{2n-3}{4n}$  for  $n=1,2,3,\dots$  you see that the set

has an infinite number of points that approach  $\frac{1}{2}$  or  $-\frac{1}{2}$  or  $\frac{1}{4}$  or  $-\frac{1}{4}$  as  $n$  goes higher and higher. The picture on the top of the left column on the next page demonstrates this.

# Sequences, series and limits with the HP49G - Part 1



Of course the programs are way not perfect. It will crash in some cases, or even return wrong results in other cases. But I hope you had a good overview over some of the capabilities of the HP49G which are good when it comes to creating new things.

This was a very very difficult birth but now that we're through I think it was worth it. Next time we will continue with sequences and we'll see how our programs can help and what the HP49G offers for working with them. And of course we will see where the affair of Suzan and Trabakoulas ends ;-)

Infinite unbounded yet condensed greetings,  
Nick.

BOUNDS returns  $\frac{-1}{2}$   $\frac{1}{2}$  and a 1 because the set is bounded.

HASCNDSPT? returns a 1 (yes).

CONDENSPT returns  $\frac{-1}{2}$   $\frac{-1}{4}$   $\frac{1}{4}$   $\frac{1}{2}$ , the four condensation points of the set.

And finally CONVERGES returns 0. because the set has 4 condensation points.

# Sequences, series and limits with the HP49G - Part 2

Hi again and may the (con)sequences of your decision to buy the HP49G always lead your way.

In the second part of SESELIMA (SEquences, SEries and Limits MARathon, sounds quite swiss, almost Sesseli Matt, what Thomas?) we will extend the romance story of Trabakoulas and Suzan of the second floor to sequences. And see what the (con)sequences of the romance were. (Nick, you are repeating yourself! ;-))

What has been said and done until now about point sets are reasonably extendable for sets of numbers and number sequences. We only need to switch from the points to the coordinates. Any number sequence is then also a set of numbers. But from any given number set we can build up many different sequences, for example through choosing a special *sequence* in which the numbers will be ordered. The resulting ordering of numbers is then characterised by a number symbol, which is often an index.

We can then repeat the same sayings of part 1 of this marathon for number sequences.

- 1) A finite sequence has a finite number of elements (numbers), an infinite has an infinite numbers.
- 2) A sequence is bounded, if an upper bound  $K$  and a lower bound  $k$  exist such that for each number  $a$  of this sequence  $k \leq a \leq K$ .

A positive number  $K$  can then be given such that for the absolute values  $|a|$  of the numbers of the sequence:  $|a| \leq K$ .

- 3) If we denote the whole sequence with  $(a_n)$  and a particular member of the sequence  $a_n$  (with the index  $n$  for  $n^{\text{th}}$  element) then we can formulate the definition of a condensation point as follows: A number  $A$  is a condensation point of the sequence  $(a_n)$  if for an arbitrary positive number the inequality holds:

$|a_n - A| < \epsilon$  for an infinite number of indices  $n$  of the sequence.

For example, the sequence  $a_n = (-1)^n \frac{n+3}{2n}$  has the two condensation points  $\frac{1}{2}$  and  $-\frac{1}{2}$  because in an arbitrary close neighbourhood of  $\frac{1}{2}$  and of  $-\frac{1}{2}$  we can find infinite numbers  $a_n$  of the sequence.

A second example: The sequence given by:

$$1 + \frac{1}{2}, \quad 2 + \frac{1}{2}, \quad 3 + \frac{1}{2}, \quad 1 + \frac{1}{3}, \quad 2 + \frac{1}{3}, \quad 3 + \frac{1}{3}, \quad 1 + \frac{1}{4}, \\ 2 + \frac{1}{4}, \quad 3 + \frac{1}{4}, \dots$$

has the three condensation points 1, 2 and 3. The rule for building up the sequence is  $a_n = n - 3 \text{ FLOOR } \frac{n-1}{3} + \frac{1}{2 + \text{FLOOR } \frac{n-1}{3}}$

with  $n = 1, 2, 3, 4, \dots$ . The function FLOOR is in the third page of the menu MATH/REAL. It takes a real number or integer from the stack and returns the biggest integer that is less than or equal to the argument. This is one of the functions that are not very logical in their behaviour. If the result of some operation is an integer, then one would also expect the result to be of type integer, that is 28. But no, the result is an integer with type 0 (real number) that is with *decimal* point. And to make things even funnier, if you give FLOOR an integer number then suddenly the result is of type integer. Sometimes one way, sometimes the other way around. So that you always have to think about many things that can go wrong when you program and to waste time on things that wouldn't be necessary if care had been taken when

# Sequences, series and limits with the HP49G - Part 2

the HP49G was designed right from the start. But let's move on.

- 4) Any infinite and bounded sequence has at least one condensation point. (That was Bolzano and Weierstraß)
- 5) A sequence  $(a_n)$  converges against the limit  $a$ , when almost all numbers of the sequence are in an arbitrary close - neighbourhood of the limit. That means: You can choose  $\epsilon$  to be as small (but positive) as you like. Then an index  $N$  that depends on your choice of  $\epsilon$  can always be given, such that all sequence members  $a_n$  with index  $n > N$  are between  $a - \epsilon$  and  $a + \epsilon$ .

This means the same as that *the sequence  $(a_n)$  has one and only one condensation point, which then is the limit of the sequence.*

We denote this with  $\lim_{n \rightarrow \infty} a_n = a$ .

The difference to the condensation points is that beginning with some index  $n$  not only infinite numbers  $a_n$  but *all* numbers  $a_n$  must be in the arbitrary close -neighbourhood of  $a$ . (Rcobo this was for you ;-))

The opposite is also valid: One sequence with exactly one condensation point converges always.

The programs that we have up to now, work also with sequences because we have on the HP49G no distinction whatsoever between sets and sequences. Actually it would be better to keep the notation  $\{\text{generalMember} \{\text{indexVar} \text{ low } \text{high}\}\}$  for sequences, as sets can be much much more general. But the sequences are also more general than the above notation implies. So perhaps we should take a closer look at the construction plans of sequences. First of all, any *sequentially* ordered set of numbers is a number sequence. For

example the set  $a_1 = 7, a_2 = 3, a_3 = 13$  is a finite number

sequence. The set of even numbers  $a_1 = 2, a_2 = 4, a_3 = 6, \dots$  is one infinite number sequence, that is each member has its next.

A sequence  $(a_n)$  is given, if each natural number  $1, 2, 3, \dots$  can be mapped to exactly one number  $a_n$ , which is then called the  $n^{\text{th}}$  member of the sequence.

The recipe for building up the members is often analytic. For example the sequence  $a_n = \frac{1}{n}$  describes all members with an analytic formula. Such sequences are covered good by our notation. The above sequence for example would be denoted as  $\frac{1}{n} \{n \geq 1\}$ .

But there are also non analytic sequences. Let's first take a sequence that is non-analytic in the sense of the HP49G CAS. The sequence from the last page  $a_n = n - 3 \text{ FLOOR } \frac{n-1}{3} + \frac{1}{2 + \text{FLOOR } \frac{n-1}{3}}$  is such an example. The HP49G doesn't know the derivative of FLOOR and so this sequence is non analytic for it. We can denote this sequence as

$$n - 3 \text{ FLOOR } \frac{n-1}{3} + \frac{1}{2 + \text{FLOOR } \frac{n-1}{3}} \{n \geq 1\}, \text{ but}$$

how would our programs react when they see the function FLOOR? Will they find on which floor Suzan lives? ;-)) Let's try. Enter that monster and store it in some cave like 'NONAN' to have it handy whenever blood thirsty programs appear. ;-)) Now try SPCASES. We won, 1-0 for us. SPCASES did its work OK and returned the only

# Sequences, series and limits with the HP49G - Part 2

special case, that is the sequence itself, in a list:

$$n-3 \text{ FLOOR } \frac{n-1}{3} + \frac{1}{2+\text{FLOOR } \frac{n-1}{3}} \{n-1\}$$

Try ISINF?. After some seconds in suspense the result is 1. Oh yes, the sequence is infinite and the score 2-0 for us.

Now we go on with BOUNDS. And here the program crashes!. But it crashes because Nick has forgotten something, so we can blame it on him. All together now: "Think again, Nick!" Fortunately the correction isn't hard to do. Just add the bold code in BOUNDS.

```

.....
IFERR
  TABVAR
THEN
  CASE
    ERRN # DE65h ==
    THEN
      lo hi 2 ->LIST OVER DUP 2
      ->LIST 2 ->LIST 0
    END
    ERRN # DE25h ==
    THEN
      -105 SF EVAL XQ TABVAR
    END
    lo hi 2 ->LIST
    ? DUP 2 ->LIST
    2 ->LIST 0.
  END
.....

```

Now we can try again. This time it is the function lim that errors out with "Bad Argument Type". It can't deal with FLOOR. What are we

going to do? Well, let's try SERIES, which is more powerful in finding limits. Bring the sequence on the stack and use HEAD to get the head of the monster. Enter n, then RCLVX, press [=] and then

SUBST to get  $X-3 \text{ FLOOR } \frac{X-1}{3} + \frac{1}{2+\text{FLOOR } \frac{X-1}{3}}$  (if your

VX is X). Why do we change variables here? Press again RCLVX, enter and then press [=] to get  $X = +$  (again, if your VX is X) and then enter the order of the expansion, for example 1. Now press [SERIES]. The HP49G needs a couple of seconds and returns a result:

On stack level 1 you have  $h = \frac{1}{X}$  and on stack level 2 the list

Limit: Equiv:  $\frac{?}{X}$  Expans:0 Remain:  $\frac{?}{X}$  . Hurrah! SERIES

found a limit. The rest is not very helpful but since we are interested only for the limit, we decide to use SERIES in cases where lim fails. Second correction to BOUNDS:

```

.....
"Adding low" 1 DISP lo
PICK3 RCLVX
lo =
IFERR @Use SERIES if lim errors out
  lim
THEN
  1 SERIES
  DROP HEAD
END
2 ->LIST 1 ->LIST +
"Adding high" 1 DISP hi ROT RCLVX hi =
IFERR @Use SERIES if lim errors out
  lim
THEN
  1 SERIES
  DROP HEAD EVAL
END

```

# Sequences, series and limits with the HP49G - Part 2

2 ->LIST 1 ->LIST +

.....

Try again. This time BOUNDS doesn't crash but returns {? ?} and ?. It says "Dunno", so to speak. This "dunno" is the result of using TABVAR, which it errors in this case but because it is in an error trap,

the HP49G returns the variation table  $1 \frac{3}{2} \{+ \ ?\}$ . The

question mark ("dunno") in this table keeps appearing as the result of any subsequent operation.

But wait a minute! SERIES said that the limit for X of the monster is . This is wrong! The monster has no limit because it has *three* condensation points! It's good to have the code that puts the "dunno" in the variation table, so that the result at the end is also "dunno", which is better than "Yes, I know the wrong result"! We use SERIES here only as an eventual life saver that let's us proceed after lim crashes the program and/or perhaps a better command to find limits that lim can't find. Do you see how the sincerity of one and the robustness of the other give a better combination? It's like real life, man! ;-) Let's hope that SERIES won't cause any great crashes. And what about the score? Well, hard to say who wins this time. Leave it at 2-0 for us. (We win, so we can let the score unchanged ;-))

Now we try HASCNDSPT?. Since the corrections of BOUNDS also influence the fate of HASCNDSPT? we can be sure that if it crashes, it crashes for other reasons. And it really does. After several seconds you see a "THEN Error: Bad Argument Value" and a ? on stack level 1. Let's correct that. Replace the red code

```
ELSE      @else @drop poor <-Suzan
          DROP DUPDUP
END
IF
  BOUNDS NIP @BOUNDS sees <-Trabakoulas defined
THEN
  1.
ELSE
```

?

**END**

with the new bold code

```
ELSE      @else @drop poor <-Suzan
          DROP DUPDUP
END
BOUNDS NIP
CASE
  DUP ? SAME
  THEN
  END
  DUP
  THEN
    DROP 1.
  END
  ?
END
```

Now try again. It takes quite a long time but the program returns ?, also an honest answer. Score remains 2-0?

Last check. What does CONDENSPT with the monster? It needs several seconds until it crashes when it tries to find the limit of the monster for  $n =$  . If we use the correction again:

```
"Finding limits for
" OVER + "
" + 1 DISP
IFERR
  lim
THEN
  1 SERIES DROP HEAD EVAL
END
xvar STOVX
```

then CONDENSPT returns the dishonest answer {Limit: } . Would it be better to use the following correction?

# Sequences, series and limits with the HP49G - Part 2

```
"Finding limits for
" OVER + "
" + 1 DISP
IFERR
  lim
THEN
  DROP2 ?
END
xvar STOVX
```

I can't say. If we don't use SERIES we perhaps lose some results which SERIES can calculate but lim can't. On the other hand using SERIES can give wrong results. What can we do? Perhaps put a "dunno" and the result of SERIES with its label in a list, so that we know that SERIES was in work, so perhaps the found condensation point is OK but perhaps also not? Be it that way. We correct CONDENSPT:

```
"Finding limits for
" OVER + "
" + 1 DISP
IFERR
  lim
THEN
  1 SERIES DROP HEAD ? 2 ->LIST 1 ->LIST
END
xvar STOVX
```

With this correction CONDENSPT returns  $\{\{ \text{Limit: } ? \} \}$  as if to say, "hmm, is it correct what I found?". Now the score is 2-1 (too much sincerity can cause damage ;-)). But we have at least a more or less well behaved collection of modules.

Sequences can also be made recursively. The famous Fibonacci numbers for example are such a sequence. The recipe for them is  $a_1 = 0$ ,  $a_2 = 1$ ,  $a_n = a_{n-1} + a_{n-2}$ . The HP49G allows to define functions that calculate members of such recursively defined

sequences. The Fibonacci sequence for example can be defined just like its definition. You just go to the EQW and type:

$$\text{FIBO}(n) = \text{IFTE}(n == 1, 0, \text{IFTE}(n == 2, 1, \text{FIBO}(n-1) + \text{FIBO}(n-2)))$$

Then you press [ENTER] and [DEF]. The HP49G creates a user function FIBO that has all it needs to calculate Fibonacci numbers. If you recall it on the stack it looks like:

```
<<
-> n
' IFTE(n==0,1,IFTE(n==1,FIBO(n-1)+FIBO(n-2))) '
>>
```

This was one of the most amazing examples given in the manual of the old HP48SX! A function that calls itself until some break condition is true. No loops, no explicitly saving sum results in-between, just a clear and compact *mathematical* definition. By the way, this is the second way of user function definitions. A local variable structure including only one algebraic object directly after the definitions of local variables. Its general structure is:

```
<<
-> var1 [var2 [var3 [...]]]
'algebraic that uses the local variables'
>>
```

This type of user defined function takes 1 or more arguments from the stack and returns exactly one result. It can be used in other algebraic objects. For example you could put the function FIBO in another algebraic object:

$$\text{FIBO}(3) \quad X^2 - \text{FIBO}(2) \quad X.$$

If you write the above in the EQW you can use the soft key for FIBO from the menu VARS. When you press that key the name FIBO appear on the EQW already selected (inverse). You must then press



# Sequences, series and limits with the HP49G - Part 2

[BACKSPACE] (delete backwards) and then start putting the parentheses and the argument of the function. If you EXPAND the above you get  $X^2 - X$ . You can even use FIBO as a part of other definitions. For example DEFine the user function:

$$F(X,m) = FIBO(m) \cdot X^m$$

Then EVALuating  $F(U^2 - 1, 4)$  will return  $2 \cdot (U^2 - 1)^4$ . You can enter  $F(X, FIBO(5))$  and EVALuate it to get  $X^3$ . Let's take a look how

EVALuation takes place. If you give the function F two arguments, the name X and one integer, say 4, then a cascade of evaluations takes place. You are kept away from all this inner complications. You only have your clear mathematical definition and let the HP49G do the dirty work. In addition to putting such user functions in algebraics, you can also use them in RPL just like any other function. For example you could also enter X, then 4 and then press the soft key of the function F. The result is the same.

A bit of caution is required sometimes. If you for example enter a symbolic argument n and then press FIBO you get:

$$IFTE(n == 1, 0, IFTE(n == 2, 1, FIBO(n-1) + FIBO(n-2))) \Big|_{n=n}$$

This is logically correct. EVALuating again you get:

$$IFTE \ n == 1, 0, IFTE \ n == 2, 1, IFTE \ (n-1 == 1, 0, IFTE \ (n == 2, 1, FIBO(n-1) + FIBO(n-2))) \Big|_{n=n-1} + IFTE \ (n-2 == 1, 1, IFTE \ (n == 2, 1, FIBO(n-1) + FIBO(n-2))) \Big|_{n=n-2}$$

You see what happens? The function has no way to determine if the symbolic argument n has one of the special values 1 or 2 which break the recurrence. You could evaluate this again and again and you would get always more complicated results. The disadvantage of such a user

function is that it can't test the type of the argument. IFTE that is allowed in algebraics is of course a big help, but it can be only used for tests that are allowed in algebraics. A hypothetical algebraic definition like for example:

$$TYPE(n) == 6, APPLY(FIBO, n),$$

$$FIBO(n) = IFTE \ n == 1, 0, IFTE \ n == 2, 1, FIBO(n-1) + FIBO(n-2)$$

is not possible because TYPE isn't allowed in algebraic objects. The above hypothetical definition brings a couple of questions. First of all, what is (officially) allowed in algebraic objects? To answer this, we must make a small excursion to the types of objects of the HP49G. As you already know, the HP49G has a great variety of different objects, which are characterised by their object type. The objects that are allowed in algebraics are of type 18. (the old "analytic functions"), of type 14. (the new CAS-functions), of type 6. (global names) and of type 7. (local names). Other types are not allowed. TYPE is of type 19. (command) so you can't put in in an algebraic object.

Another question that the above hypothetical definition throws is, what is the function APPLY?. Well, when APPLY(function, argument) gets evaluated it just returns the function function to the argument argument and returns function(argument) *holding the evaluation at this step*, even if it could be further carried out. For example, enter

$F(X) = X^2$  and press DEF. If you now evaluate  $F(2)$ , you get of course 4. But if you evaluate  $APPLY(F, 2)$  you get  $F(2)$  *though the evaluation*

*could be further carried out to return 4!*

So, let's return to the above "impossible" definition. In fact this definition can be made on the HP49G, but not the algebraic way. You have to enter it as follows:

# Sequences, series and limits with the HP49G - Part 2

```
<<
-> n
<<
CASE
  { 6. 7. 9. }    @If the argument is a local or
  n TYPE POS      @global name or algebraic
  THEN            @
  'APPLY(FIBO,n)   @Just apply FIBO
  EVAL            @on n and return FIBO(n)
END
  n 1 ==          @If n is equal to 1
  THEN            @return 0
  0
END
  n 2 ==          @If n is equal to 2
  THEN            @return 1
  1
END
  n 1 - FIBO       @If nothing of the above is
  n 2 - FIBO       @true, then calculate
  +               @FIBO(n-1)+FIBO(n-2)
END
>>
>>
```

Store this in FIBO. It is also a user function, but one with a program as definition. If you evaluate now  $FIBO(5) + FIBO(4)$ , then you get 5. But evaluating  $FIBO(5) + FIBO(X-2)$  returns  $FIBO(X-2) + 3$ . Evaluating the last result simply re-returns  $FIBO(X-2) + 3$ . So we're just out of the endless repeating evaluation. For Mathematica users: Does this somehow reminds you of Hold[]? ;-)

Now, the question is, how do we use APPLY when we want it in RPL? Here we must be careful a little bit. Though almost all functions if used in RPL syntax expect their arguments to be on the stack in the same sequence like in algebraic syntax, APPLY behaves differently. It expects the function to be applied on stack level 1, and arguments of

that function as a list on stack level 2.

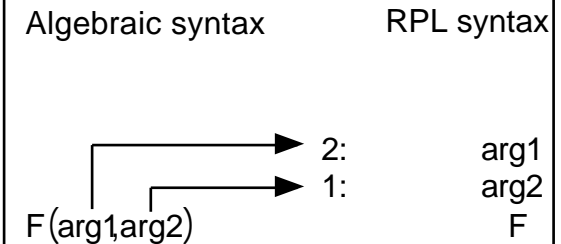
And another question is, how do we use APPLY when some function has to be applied on more than one arguments? Perhaps the answer is clear for RPL syntax. We just put a list of all arguments on stack level 2. For example enter {X Y} then F and use APPLY to get  $F(X,Y)$ . But what do we do when we want algebraic syntax? We can't put a list in an algebraic object. The answer is a bit frustrating. From the command

line we enter the arguments after the function, separated with commas.

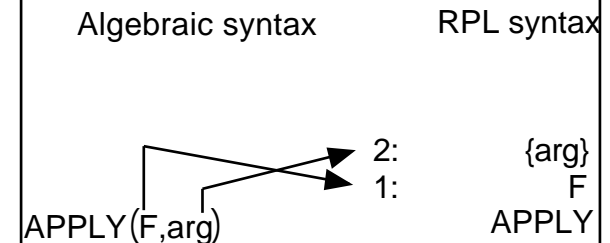
The command line contains then for example 'APPLY(F,X,Y)'. When you press [ENTER] to send this to the stack, then 'InvalidExpression' appears on stack level 1. *Captain Jean-Yves!*

*You said to me in the group that such "illegal" expressions should not be made using ->ALG, and we should only use the "officially supported" syntax. But, as you can see, the "officially supported" syntax creates itself "illegal" objects! You can't expect from the users to know when a simple "officially supported" command line will create an 'InvalidExpression'. It is not that I say that this should error out with "Invalid Syntax", not at all. The contradiction and inconsistency of your argumentation is what I find frustrating. Anyway, for us it is a blessing to know that the 'InvalidExpression' can*

Most functions behave like this:



But APPLY behaves like this:



# Sequences, series and limits with the HP49G - Part 2

be evaluated. Press [EVAL] to get  $F(X,Y)$ . And to make things even more frustrating. Up to today I just have not found any possible method to get an algebraic object that contains APPLY() in the EQW. Can somebody find a method?

The above recursive definition of the Fibonacci sequence has the advantage of clarity but it also has a disadvantage. It is very slow. Each time FIBO is evaluated with some argument  $n$ , it calculates all members of the Fibonacci sequence from  $n$  down to 0. This doesn't have to be. Consider for example:

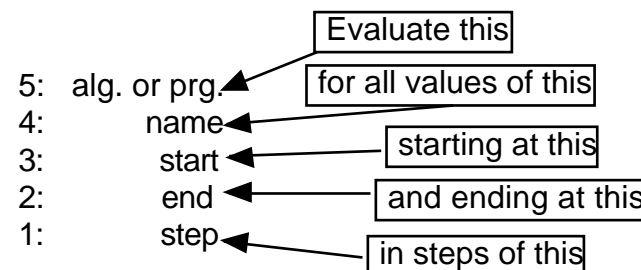
```
<< -> n
<<
CASE
  { 6. 7. 9. }    @If the argument is a local or
  n TYPE POS      @global name or algebraic
THEN              @
  'APPLY(FIBO,n)   @Just apply FIBO
  EVAL            @on n and return FIBO(n)
END
  n 1 ==          @If n is equal to 1
THEN
  0               @return 0
END
  n 2 ==          @If n is equal to 2
THEN
  1               @return 1
END
  0 1 3 n         @If nothing of the above is
  START          @true, calculate FIBO
  DUP ROT +
  NEXT
  NIP
END
>>
>>
```

This FIBO works much much faster than the other one. But the clarity of the definition is gone!

Now that we have a faster FIBO let's calculate some Fibonacci numbers, say the first 20. We don't have to write another program and put FIBO in a loop, because we have the command SEQ. From the EQW enter FIBO( $n$ ). Then enter  $n$ , 1, 20, 1 and press SEQ. (The command is in the second page of the menu PRG/LIST/PROC.) In about 11.5 seconds you have a list of the first 29 Fibonacci numbers.

SEQ expects an algebraic object or program on stack level 5, a name on stack level 4, a start and an end value on levels 3 and 2 and the step on stack level 1. It gives the variable of stack level 4 different values starting at start and ending at end, stepping with the step value each time. It evaluates the algebraic or program of stack level 5 using the values of the variable on stack level 4. The results are then wrapped in a list. The above example is equivalent to:

```
<<
  1 20
  FOR I
    I FIBO
  1
  STEP
  20 ->LIST
>>
```



SEQ can of course also be used to get sequences of algebraic objects. For example entering CYCLOTOMIC( $n$ ),  $n$ , 2, 10, 2 and pressing SEQ will give you a list with the first 5 even cyclotomic polynomials.

## Sequences, series and limits with the HP49G - Part 2

There are also sequences, for which we still don't have any recipe, be it analytic, by recurrence or whatsoever, to calculate their members. For example the sequence 3,1,4,1,5,9,2,6,5,... is known as the sequence of the digits of the decimal representation of  $\pi$ . Nobody can give an analytic recipe to calculate these digits. (But we do have algorithmic methods to do that.) Such a sequence is also the sequence of the prime numbers. Thanks to the flexibility of the HP49G it is also possible to get such non-analytic sequences. For example the program:

```
<<
-> n
<<
0
0 n
START
  NEXTPRIME
  DUP
NEXT
DROP n ->LIST
>>
>>
```

will take a number  $n$  from the stack and give you the first  $n$  prime numbers in a list. But can you think about how we can do that using SEQ? Take a look at the following program:

```
<<
0
<< NEXTPRIME DUP >>
n 1
5 ROLL
1 SEQ
REVLIST TAIL REVLIST
+
>>
```

The program `<< NEXTPRIME DUP >>` used by SEQ has no appearance of the variable  $n$  in it. This variable is in this case only a

place holder, a dummy, to just count from 1 to the number of primes that we want. The same program also duplicates the last found prime number so that the next evaluation of NEXTPRIME finds its argument on the stack. The 0 at the beginning of the program is for giving the very first evaluation of NEXTPRIME something to work with. (NEXTPRIME takes a number from the stack and returns the next prime greater than the given number.) We could also have written:

```
<<
<<
  IF
    n 0 ==
  THEN
    0
  END
  NEXTPRIME DUP
>>
n 0
4 ROLL 1 -
1 SEQ
REVLIST TAIL REVLIST
+
>>
```

In this case it is the program used by SEQ that puts the 0 on the stack, when SEQ runs the first pass.

We see: Though the HP49G has many many quirks, when it comes to flexibility there is no other calc on this earth that comes even close to it. The lonesome king of flexibility *is* the HP49G. This is the most important thing for a calculator with a CAS. To be flexible enough so that we, the users, can formulate our problems. In other words, how should we describe the word "red" to somebody who has no idea what a colour is? ("Somebody" could stand for TI this time ;-))

As you can see we have more than enough power in our hands, to create finite parts of non analytic sequences. But this is only a part of the story. The non analytic sequences can't be handled at all by our

# Sequences, series and limits with the HP49G - Part 2

programs. Of course we could denote a the Fibonacci sequence as  
or as

$$\{ \text{FIBO}(n) = \text{IFTE}(n=2, n-1, \text{FIBO}(n-1) + \text{FIBO}(n-2)) \mid n=1 \}$$

$$\{ \text{FIBO}(n) = \text{IFTE}(n=1, 0, \text{IFTE}(n=2, 1, \text{FIBO}(n-1) + \text{FIBO}(n-2))) \mid n=1 \}$$

but what's the use of it? Our programs can't do much with it. They can only work with analytic sequences. There could be two principal ways to handle also such non analytic sequences. One would be a module that can find an analytic expression out of the recurrence by recursively solving for the general member form  $a_n$ , and working with the analytic form instead. The other would be some algorithm that can work directly with the recurrence and for example find if the sequence is infinite, has bounds and so on.

Perhaps you don't believe me now, but it *is* possible in principle to convert recurrences to analytic closed forms. I for myself am still quite astonished about it. But this doesn't change anything to proven facts. The question is where to begin when explaining such astonishing things. Hmm, I see Trabakoulas smiling with his well known smile that he always put on, when something is so tricky. He says, "My son, start at the beginning!". And so I do.

You perhaps have the impression that all maths about sequences are just counting a little bit, adding a little bit, making sums and the like. But as always, when we search better, we find more things, beautiful things, extraordinary mysterious things.

Let's take the simple sequence  $A(n) = A(n-1) + 1$ , with  $A(0) = 0$  and  $n = 0, 1, 2, 3, \dots$ , which is defined by recurrence. It says that you start at the first member, which is  $A(0) = 0$ . Then, to find  $A(1)$  you simply add 1 to the previous member. So you have  $A(1) = A(0) + 1 = 0 + 1 = 1$ . And so it goes on.

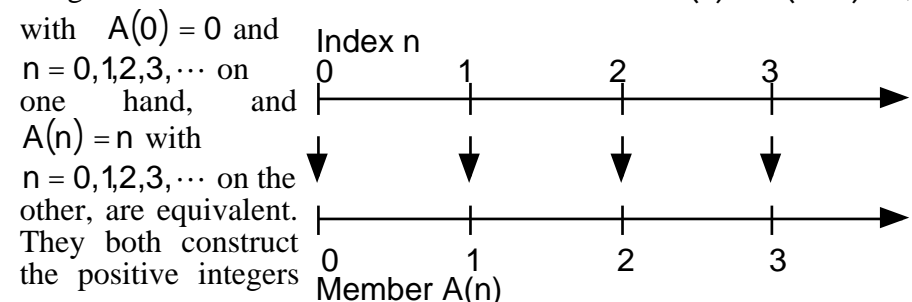
$$\begin{aligned} A(0) &= 0 \\ A(1) &= A(0) + 1 = 0 + 1 = 1 \\ A(2) &= A(1) + 1 = 1 + 1 = 2 \\ A(3) &= A(2) + 1 = 2 + 1 = 3 \\ A(4) &= A(3) + 1 = 3 + 1 = 4 \\ &\dots\dots\dots \\ &\dots\dots\dots \\ A(n) &= A(n-1) + 1 = ??? \end{aligned}$$

Did you already notice that the members are the positive integers themselves? What does this mean? Is the whole set of the natural numbers a recursively defined sequence? One could interpret this fact exactly this way. Or even say that the recipe for constructing these numbers is to just start at 0, adding 1 to the just passed number every time.

Did you also notice something else? Instead of the definition  $A(n) = A(n-1) + 1$ , with  $A(0) = 0$  and  $n = 0, 1, 2, 3, \dots$ , we can simply write  $A(n) = n$  with  $n = 0, 1, 2, 3, \dots$ . This is an *analytic closed form*.

It has a *tremendous* advantage, that exists in any analytic closed form. When you have such a form, you are able to know anything that there is to know about something. (That's why these forms are Nick's favourites. ;-))

What we actually did with this sequence, is that we mapped positive integers onto themselves. So the two definitions  $A(n) = A(n-1) + 1$ ,



# Sequences, series and limits with the HP49G - Part 2

out of the positive integers. Big deal, you may say, but wait 'cause you ain't seen nothing yet!

Before we go further, a small visit from the future. The differential equations marathon hasn't taken place yet, but for those already familiar with differential equations, did you notice how the definition  $A(n) = A(n-1) + 1$ , with  $A(0) = 0$  and  $n = 0, 1, 2, 3, \dots$ , resembles a diffeq? No? Then look:  $A(n) = A(n-1) + 1$   $A(n) - A(n-1) = 1$ . The left hand side of the equation is now the discrete analogon of a the differential quotient  $\frac{dA(n)}{dn}$ . Instead of an infinite small change in

$A(n)$  you have a finite change, a *difference*. The equation

$A(n) - A(n-1) = 1$  is the discrete version of  $\frac{dA(n)}{dn} = 1$ . The later is a

differential equation whose solution is  $A(n) = n + C$ , and  $C$  is the integration constant, which varies from case to case according to some initial or boundary conditions. The similarities are even more. If we interpret the second part of the recurrence  $A(0) = 0$  as an initial or boundary condition, then we can find the particular solution of the differential equation:  $A(0) = 0$   $0 + C = 0$   $C = 0$ . And so:

$$A(n) = n + C \quad C = 0 \quad A(n) = n + 0 \quad A(n) = n, \text{ which looks}$$

exactly like the closed analytic form of the recurrence. Of course both things aren't exactly the same. For example  $A(n)$  and  $n$  are continuous in the case of the differential equation, but discrete in the case of the sequence. On the one side we have a function and its variable, on the other side we have a sequence and its index.

Let's get back to the present. We make another easy example,  $B(m) = B(m-1) + 2$ , with  $B(0) = 0$  and  $m = 0, 1, 2, 3, \dots$ . Its analytic closed form is  $B(m) = 2 \cdot m$  with  $m = 0, 1, 2, 3, \dots$ . You see again the advantage of the analytic closed form? You don't really have to carry the additions. For example if some teacher (who wants to punish you)

tells you that you must find  $B(12345678901234567890)$ , then you could of course sit a couple of years at your desk and carry out 12345678901234567890 additions. Or you could buy an HP49G and let the program run:

```
<<
  0 SWAP 1 SWAP
  START
    2 +
  END
>>
```

It takes any number  $n$  from the stack and returns the  $n^{\text{th}}$  member of the sequence, so also the 12345678901234567890<sup>th</sup> member. This is **SCATA**. What, you don't know what **SCATA** is? That is **Super Calculators Against Teacher's Authority**.<sup>1</sup> ;-) But then you would just cause a psychic problem of the HP49G. And in the year 2010 you will say, "Poor little HP49G calculates since a decade!"

Even better it would be to program:

```
<<
  2 *
>>
```

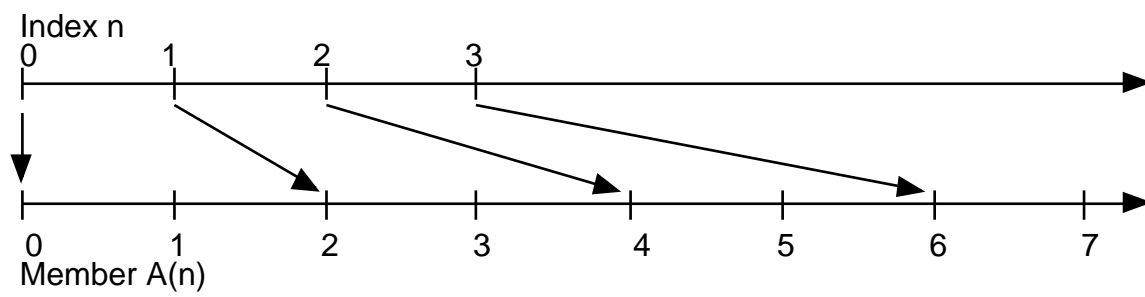
which does exactly the same as above in a time which will give the punishing teacher a heart attack. This is **MATH**. Oh no, no not mathematics, but **Mathematics Against Teacher's Health**<sup>2</sup> ;-)

It is also interesting to see what this sequence does, what it maps the positive integers onto. Turn page to take a look at the construction of the even numbers.

<sup>1</sup> Just ask a Greek it means in greek language ;-)

<sup>2</sup> Interesting to know that the word *mathematics* in greek language, gave its root "*math*" to the verb *mathaino*, "To learn".

# Sequences, series and limits with the HP49G - Part 2



For the last example split  $2 D(k-1)$  to  $D(k-1) + D(k-1)$  and then find the difference  $D(k) - D(k-1)$ .

I know what you think now. What is with more complex recurrences? For example, let's suppose you know that  $P(j) = P(j-1) - \frac{3}{8} j^2 - \frac{5}{6} j + \frac{1}{3}$ , with  $P(0) = 0$  and  $j = 0, 1, 2, 3, \dots$ . What are then

Applying the recipe  $B(m) = B(m-1) + 2$ , with  $B(0) = 0$  and  $m = 0, 1, 2, 3, \dots$ , or equivalently torturing our teacher with  $B(m) = 2m$  with  $m = 0, 1, 2, 3, \dots$ , is the same like making the even numbers.

Now, can you say what the recipe for the odd numbers is? Right, it is  $C(l) = C(l-1) + 2$ , but now with  $C(0) = 1$  and  $m = 0, 1, 2, 3, \dots$ , or in its analytic closed form  $C(l) = 2l + 1$  with  $m = 0, 1, 2, 3, \dots$ . Perhaps unexpected but after some thought understandable. The recurrence recipe for constructing the odd numbers is the same like the recipe for constructing the even numbers. Only the initial condition changes, that is we start at 0 for even numbers but at 1 for odd numbers. Later we do the same for both. Just add 2 and go on until infinity. (or until the teacher surrenders ;-))

Next example:  $D(k) = 2 D(k-1)$ , with  $D(0) = 1$  and  $k = 0, 1, 2, 3, \dots$ . This sequence just creates our beloved numbers 1, 2, 4, 8, 16, 32, 64, 128, ... and so on. Its analytic closed form is  $D(k) = 2^k$  with  $k = 0, 1, 2, 3, \dots$ . (Computer basics come from mathematics? It seems to be this way!)

A note for the eager inpatient people who can't wait until the differential equations marathon: Try to find the corresponding differential equations for the above examples. Try to solve them using the given initial conditions. Do you find any interesting things? (Hint:

the numbers  $P(j)$ ? What does  $P(1000)$  look like? Did the teacher win after all? Are we lost?

But here cometh the HP49G our hero and life saver, the intellectual child of the Professor and the Captain, the one and only. (Butter them up, so that they have grace and give us the HP5000GX in future ;-)). One of its not so widely known functions is the rarely used SIGMA. What this function does, will amaze you now! It takes a difference, like for example the above  $A(n) - A(n-1)$  (which was 1) and the a variable (the index) and returns the general analytic closed form of the sequence. Don't believe it? Then enter a 1, then 'n', and then press SIGMA. (It is on the second page of menu CALC/DERIV). What do you see? Yes, n the analytic closed form of  $A(n) - A(n-1) = 1$ .

Second example was  $B(m) - B(m-1) = 2$ . So enter 2, 'm' and press SIGMA again. Result is  $2m$ , the analytic closed form of the even numbers. Now I hear the rebellion boiling. "For the odd numbers we have also  $C(l) - C(l-1) = 2$ , but entering 2, then l and pressing SIGMA gives us  $2l$ , the even numbers!" Yes, yes, my rebels, but don't forget the boundaries. You remember of course the integration variable which varies from case to case according to the initial/boundary condition? The point is that SIGMA returns the solution without consideration of boundaries. (How could it, when we don't give it any?) So when you get a solution from SIGMA, you just add a constant, say C to it. For example, add C to the solution  $2l$ . Now you have  $2l + C$ . Press DUP to make a copy on stack level 2.

# Sequences, series and limits with the HP49G - Part 2

Now, we know that the initial condition, the start so to speak, is at  $C(0) = 1$ . That means, when the index  $l$  is 0, then  $C(l)$  is 1. So, enter  $l = 0$  and press SUBST. Now you have  $2 \cdot 0 + C$ . This is the member  $C(0)$ , which is equal to 1 according to our initial condition. Enter a 1 and press [=] to make the equation  $2 \cdot 0 + C = 1$ . Enter  $C$  and press SOLVE to get  $C = 1$ . Now, SUBSTitute this in the equation on stack level 2. You have now  $2 \cdot l + 1$ , the odd numbers, which come out when the recurrence  $C(l) = C(l-1) + 2$ ,  $m = 0, 1, 2, 3, \dots$  has the initial condition  $C(0) = 1$ . (For the even numbers we would find  $C = 0$  and so the solution given by SIGMA doesn't change.)

Actually SIGMA wants the difference  $F(n+1) - F(n)$  and not  $F(n) - F(n-1)$ . This makes no difference, when such differences don't depend on the index  $n$  but it does make difference when they do. Let's try  $P(j) = P(j-1) - \frac{3}{8}j^2 - \frac{5}{6}j + \frac{1}{3}$ , with  $P(0) = 0$  and  $j = 0, 1, 2, 3, \dots$ . We make first the definition by recurrence. Go to the EQW and enter:

$$\text{PRCR}(j) = \text{IFTE } j == 0, 0, \text{PRCR}(j-1) - \frac{3}{8}j^2 - \frac{5}{6}j + \frac{1}{3}$$

Then press ENTER to put this on the stack and press DEF. The HP49G creates the user defined function PRCR which you can find in the menu VAR. Let's try some points. Enter 'PRCR(n)', 'n', 0, 5, 1 and press SEQ. The calculation takes a moment and returns the first 6 numbers of this sequence wrapped in a list:

$$0 \quad \frac{-7}{8} \quad \frac{-89}{24} \quad \frac{-37}{4} \quad \frac{-73}{4} \quad \frac{-755}{24}$$

Now for the analytic definition. Enter  $-\frac{3}{8}j^2 - \frac{5}{6}j + \frac{1}{3}$  (this is the difference  $P(j+1) - P(j)$ ). Enter  $j$  (the index) and press SIGMA. The result is

$$-\frac{6j^3 + 11j^2 - 33j}{48}$$

Now, we must apply the initial condition but we must be a little bit careful. This result is meant as  $P(j+1) = -\frac{6j^3 + 11j^2 - 33j}{48} + C$ . We can't make such a

definition on the HP49G. The left hand side *must* contain a function with one or more names of variables separated with commas. This doesn't mean that we can't *calculate* things like  $P(j+1)$ ,  $P(j-k)$  or even  $P \sum_{n=0}^{10} n$ . It is only the *definition* that wouldn't work that way.

Because we can't define such a function on the HP49G, we simply shift the initial condition. The recursive definition contains the initial condition as the part  $\text{IFTE}(j == 0, 0, \dots)$ . We see that for  $j = 0$  the function will return 0. The index of the analytic expression is ahead of the index in the recurrence by 1. It is not  $P(j)$  but  $P(j+1)$  instead. That means that we must apply the initial condition  $P(1) = 0$  when we make the analytic definition. All clear? OK! First enter 'C' and press [+] to add the constant. Now press ENTER to duplicate. Enter 0 and press

$$[=] \text{ . Now, enter 'C' and SOLVE. The result is } C = \frac{-1}{3} \text{ . SUBSTitute this to the expression on stack level 2. Press EXPAND to get } -\frac{6j^3 + 11j^2 - 33j + 16}{48} \text{ . Now, because we shifted the index } j \text{ by}$$

1 for the initial condition, we must also do that for the expression itself, or else the results will be shifted. We must add 1 to  $j$  in the expression. Enter  $j = j + 1$  and SUBSTitute. Note that  $j = j + 1$  is an impossible mathematical equation, it doesn't have solutions. But doing this in this case we simply mean, find all occurrences of  $j$  and replace by  $j + 1$ . Now press EXPAND. The result should be

$$-\frac{6j^3 + 29j^2 + 7j}{48}$$

We make a couple of values also for this

analytic form. Press ENTER to make a copy and then enter 'n', 0, 5, 1. Press SEQ. In a few seconds the HP49G returns



# Sequences, series and limits with the HP49G - Part 2

0  $\frac{-7}{8}$   $\frac{-89}{24}$   $\frac{-37}{4}$   $\frac{-73}{4}$   $\frac{-755}{24}$ , the same list of numbers like for the recurrence. That means that starting at the number  $P(0) = 0$ , counting  $j$  from 1 to infinity and adding always  $-\frac{3}{8}j^2 - \frac{5}{6}j + \frac{1}{3}$  to the previous number, is exactly the same like counting  $j$  from 0 to infinity and plugging the value of  $j$  in  $-\frac{6j^3 + 29j^2 + 7j}{48}$ . DROP the list of numbers and using the already available copy of  $-\frac{6j^3 + 29j^2 + 7j}{48}$  DEFine PANL(j) =  $-\frac{6j^3 + 29j^2 + 7j}{48}$ .

Now, enter 1000 and calculate  $P(1000)$  using the new function PANL. The result  $-\frac{251208625}{2}$  comes instantly and turns the teacher to a maniac running with hands up, shouting at the people: "Teachers and directors against MATH!!!!". ;-) If you remember I already said that the recurrence has some kind of elegance not present in loops. But as you can see there is even more elegance in the analytic closed form. If we can find such a form (though not always possible) we should work with it. Not to speak about other considerations, like for example bounds, condensation points and limits. For example consider the sequence  $S(n) = S(n-1) + \frac{1}{n}$ , with  $n = 1, 2, 3, \dots$  and  $S(1) = 1$ . Can you say if it has a limit for  $n \rightarrow +\infty$ ? You would perhaps say that it seems to have one, because we add something that always gets smaller and smaller and so we expect the overall accumulation to be finite. But let's look at the analytic side of things.

1. Enter  $\frac{1}{n}$  and then  $n$ .
2. Press SIGMA. The result is  $\text{Psi}(n)$ , a special function built-in in the CAS of the small wonder.
3. Enter 'C', press [=] and [ENTER].

4. Enter a 1, press [=].
5. Enter  $n = 2$  (start index shifted by 1) and press SUBST.
6. SOLVE for C. Result is:  $C = -(\text{Psi}(2) - 1)$
7. SUBSTitute.
8. Enter  $n = n + 1$  and SUBSTitute again. The result is the analytic closed form of the recurrence:  $\text{Psi}(n+1) - (\text{Psi}(2) - 1)$ .

Now let's see if it has a limit for  $n \rightarrow +\infty$ . Enter  $n = +\infty$  and press lim. You get  $+\infty$  which clearly shows that adding  $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots$  will grow above all finite quantities.

What about  $S(n) = S(n-1) + \frac{1}{n^2}$ , with  $n = 1, 2, 3, \dots$  and  $S(1) = 1$ ?

Does this has a limit for  $n \rightarrow +\infty$  or not? We could tend to say "No, it doesn't" because of the previous example. But if you follow the instructions of the previous example, you find the analytic closed form:  $-\text{PS}(n+2, 1) + \text{PS}(2, 1) + 1$ , PSI being another special built-in function. If you now take the limit for  $n \rightarrow +\infty$  you find  $\frac{2}{6}$ .

SIGMA will also handle other functions, like for example exponentials. If you enter  $e^{\frac{n-6}{6}}$ , then  $n$  and press SIGMA, you will get  $\frac{e^{\frac{n-6}{6}}}{e^{\frac{-6}{6}} - 1}$ . And what about trigonometrics? Well, if you enter

$\text{SIN} \frac{n}{6} - 1$  and  $n$  and press SIGMA, then you get

$\text{SIGMA} \text{ SIN} \frac{n}{6} - 1, n$ . But we don't give up! Remember what

we said at the old times of the trigonometry and the complex marathon? The HP49G seems to like complex exponentials much more than

# Sequences, series and limits with the HP49G - Part 2

trigonometric functions. We jumped to complex hyper space, got the solution, and jumped back to real normal space. Let's try it also here.

If you still have the result  $\text{SIGMA SIN } \frac{n}{6} - 1, n$  on stack, then

press OBJ->. This is one of the most versatile commands, since it can take just about anything and explode it to its components. In this case

it returns  $\text{SIN } \frac{n}{6} - 1$  and  $n$  to levels 4 and 3. These were the

arguments for SIGMA before it was blown to pieces. On stack level 2 you have a 2, the number of arguments of SIGMA. And on level one you have the function itself, SIGMA. Press DROP2 to get rid of the 2 and of SIGMA. Now, because we are going to work in complex mode, we should make the assumption first, that  $n$  is real. So we avoid complicated results containing  $\text{RE}(n)$  and/or  $\text{IM}(n)$  later. Press UNASSUME to erase all assumptions about  $n$ , so that we make a clean start. UNASSUME leaves its argument on the stack, so press DUP and ADDTOREAL. Now the HP49G assumes that  $n$  is real. (We already have seen in the complex numbers marathon under which circumstances such assumptions are used.) Press SWAP to get the trigonometric function on stack level 1. (Instead of SWAP you can also press the key [arrow right].) Now press EXPLN. This is our jump to complex hyper space. It converts the trigonometric functions

$$e^{i \frac{n-6}{6}} - \frac{1}{e^{i \frac{n-6}{6}}}$$

to complex exponentials. The result is:  $\frac{e^{i \frac{n-6}{6}} - \frac{1}{e^{i \frac{n-6}{6}}}}{2i}$ . Press

SWAP to get the flying  $n$  to the ground and press SIGMA. After some seconds you get the huge result:

$$\frac{\frac{-(1+2i-i\sqrt{3})e^{\frac{i(n-6)}{6}}}{2} - \frac{-(1-2i+i\sqrt{3})}{2e^{\frac{i(n-6)}{6}}}}{2i}$$

Now we can calculate the real and imaginary part of this monster. to:

(This is our jump back to real normal space.) DUPLICATE that and press RE. RE returns the real part of some complex quantity, taking all assumptions into consideration. IM returns the imaginary part and as you have guessed also uses the assumptions that you have previously made. The HP49G says now:

$$\frac{\frac{\cos \frac{n-6}{6} (-2-\sqrt{3}) + -1 \sin \frac{n-6}{6}}{3} - \frac{(2+\sqrt{3}) \cos \frac{n-6}{6} - -1^2 \sin \frac{n-6}{6}}{\text{SQ} \cos \frac{n-6}{6}^2 + \text{SQ} 2^* \sin \frac{n-6}{6}}}{2}$$

We get an even bigger monster. But soon we gonna put it on a diet. Press TCOLLECT and then EXPAND to get the final answer for the

$$\text{real part: } \frac{\sin \frac{n-6}{6} + (2+\sqrt{3}) \cos \frac{n-6}{6}}{2}$$

Press SWAP to get the previously made copy of the expression

$$\frac{\frac{(-(1+2i)-i\sqrt{3})e^{\frac{i(n-6)}{6}}}{2} - \frac{-(1-2i+i\sqrt{3})}{2e^{\frac{i(n-6)}{6}}}}{2i} \text{ on stack level 1.}$$

Now press IM, and then TCOLLECT to get a fat round 0. So the jump to hyper space was successful. (Watch out for low flying assumptions when you are there ;-)) That means that the general analytic closed

form of  $S(n) = S(n-1) + \text{SIN } \frac{n}{6} - 1$ , with  $n = 1, 2, 3, \dots$  is equal

# Sequences, series and limits with the HP49G - Part 2

$$\frac{\sin \frac{n-6}{6} + (2 + \sqrt{3}) \cos \frac{n-6}{6}}{2}$$

This was a simple recurrence but it is the analytic form that allows us to say how the sequence behaves. We can first create the analytic closed form for some initial conditions, say  $S(0) = 0$ . Then we can find our special cases, eventually bounds or condensation points. We couldn't do that with the recurrence.

Now, imagine what happens with even more complicated recurrences. They are too hard to handle in that form but when an analytic closed form comes, many many things get clear. Use SIGMA<sup>3</sup> for such things, and don't listen to the voices that having null understanding of the underlying MATHeMatics still continue to flood the group with stupid opinions, like "They changed things that worked just to put a few more functions on the calc that nobody needs." Such statements you can PURGE immediately.

The opposite of finding the analytic expression for a recurrence, would be to find the recurrence out of an analytic expression But that's easy. You just enter your analytic closed form, for example  $n^2$ , DUPLICATE it, enter  $n = n + 1$  and SUBST. Then you SWAP and press [-] and EXPAND. The result  $2n + 1$  means  $P(n+1) = P(n) + 2n + 1$ .

Now, we said that SIGMA only works when the difference  $F(n+1) - F(n)$  doesn't depend on  $F$  itself. For example it doesn't work with the recurrence  $FIBO(n) = FIBO(n-1) + FIBO(n-2)$ , where  $n$  goes from 1 to infinity and where  $FIBO(1) = 0$  and  $FIBO(2) = 1$ . In this case, if we calculate the difference  $FIBO(n) - FIBO(n-1)$ , then we see that it is equal to  $FIBO(n-2)$ ,

<sup>3</sup> There is also SIGMAX on the third page of the menu CALC/DERIV. It is analogous to SOLVEVX. It only needs the expression for the difference  $F(X+1) - F(X)$  on stack level 1 and does the same like SIGMA for the current VX (which often is X).

that is, it is equal to something that depends on FIBO itself. But that doesn't mean that there are not ways to find an analytic closed form even for such sequences. In fact, we can program the HP49G to do that. This is the program that we do a few pages later. But first a bit more theory. Hold your hats on, we go with great velocity! Teachers, the performance of the MATH-virus has just began!

Because we already are familiar to the Fibonacci sequence, let's stay with it. We use it as an example for a general method, which will help us in many cases to find such analytic closed forms. We have used recurrences, which in general say that we must do something with past members of the sequences, in order to find the future members. The legacy of the past dictates the future, recurrences must have studied history ;-). Now, what we do to past members is often add them together multiplied perhaps with some factor. When we add many things together, hmm, reminds us of infinite sums in some way, doesn't it? So let's consider such a sum, an easy one:

$B(X) = b_0 + b_1 X + b_2 X^2 + \dots = \sum_{n=0}^{\infty} b_n X^n$ . Mathematicians tend to play very often. (That's why we don't have many of them in Greece, now that games are banned ;-)) What games can be arranged with such a thing like the above sum? We can look if it converges to something when we keep on adding and adding. We can check if it converges no matter what  $X$  is, or if it converges only for some particular range of values for  $X$ . Or we can forget the convergence for a moment and look at the coefficients  $b_n$ . What happens if they change? Are there any sets of such coefficients, any special recipes of how construct them? And what happens when we construct them according to some particular recipe.

Consider for example our beloved sequence of RAM capacities, that is, the sequence 1,2,4,8,16,32,64, ... . What if we plug these numbers in the positions of the slots  $b_n$  in the sum? We have then

$$1 + 2X + 4X^2 + 8X^3 + 16X^4 + \dots = \sum_{n=0}^{\infty} 2^n X^n$$

Now, it turns

# Sequences, series and limits with the HP49G - Part 2

out that for  $X \prec 1$  the sum converges to  $\frac{1}{1-2X}$ . Again a closed analytic expression for something infinite! Now you remember of course that the same sequence 1,2,4,8,16,32,64,... is also defined as the recurrence  $D(k) = 2 D(k-1)$ , with  $D(0) = 1$  and  $k = 0,1,2,3,\dots$ .

So this recurrence seems to have to do something with  $\frac{1}{1-2X}$ . (We forget here totally if it converges or not. We focus solely on the coefficients  $b_n$ .) Imagine that you knew right for the start such an

expression, like  $\frac{1}{1-2X}$ , but for some other sequence. "Great!", you say, "so what?". Well, look what happens when you expand  $\frac{1}{1-2X}$

to a series around  $X = 0$ . Enter first  $\frac{1}{1-2X}$ , then  $X = 0$  and then the highest power that you want to have in the series, say 8. Now press SERIES. The result is

Limit:1 Equiv:1

Expans:16 h<sup>4</sup> +8 h<sup>3</sup> +4 h<sup>2</sup> +2 h+1 Remain:h<sup>5</sup>

on stack level 2 and  $h = X$  on stack level 1. Press SUBST to substitute  $h = X$  in all algebraic objects of the list. Now press enter 3 and press GET to get the series from the list. The series is tagged with the string "Expans". You can remove such tags with the command DTAG. Now you should have the series alone on stack level 1: 16 X<sup>4</sup> +8 X<sup>3</sup> +4 X<sup>2</sup> +2 X +1. Do you see? Of course you do! The coefficients of X are the sequence  $D(k) = 2 D(k-1)$ , with  $D(0) = 1$  and  $k = 0,1,2,3,\dots$ . So what a strange thing is this? You

have a function, namely  $\frac{1}{1-2X}$ , which doesn't look so extraordinary at all, and when you expand it to a power series around 0, it *generates* the numbers of the recursion  $D(k) = 2 D(k-1)$ , with

$D(0) = 1$  and  $k = 0,1,2,3,\dots$ , as coefficients of the powers of the

series. This is why  $\frac{1}{1-2X}$  is called the *generating function* of

$D(k) = 2 D(k-1)$ , with  $D(0) = 1$  and  $k = 0,1,2,3,\dots$ . Wow! If we had a way to guess the right generating function, I tell you man, we have the whole infinite recurrence in one single analytic closed form.

Let's trick around a bit with Fibonacci. We want to have  $B(X)$  the generating function of the Fibonacci sequence. That means, we want to have:

$$B(X) = F_0 + F_1 X + F_2 X^2 + F_3 X^3 + \dots \text{blahblah} = \sum_{n=0} F_n X^n.$$

The  $F_n$  are our known Fibonacci numbers. (Those with which the teacher can still punish you, by telling you to calculate  $F_{1000000}$ . ;-)) The recurrence definition of the Fibonacci numbers is  $F_n = F_{n-1} + F_{n-2}$ . So let's substitute this in the sum above. We get:

$B(X) = \sum_{n=0} F_n X^n = \left( \sum_{n=0} F_{n-1} + \sum_{n=0} F_{n-2} \right) X^n$ . Now, because we know that  $F_0 = 0$  and  $F_1 = 1$ , we can take the first two numbers out of the sum and start summing at  $n = 2$ . So we have a new sum:

$$B(X) = F_0 + F_1 X + \sum_{n=2} (F_{n-1} + F_{n-2}) X^n = X + \sum_{n=2} (F_{n-1} + F_{n-2}) X^n.$$

We distribute the multiplication  $(F_{n-1} + F_{n-2}) X^n$  and so we get

$$B(X) = X + \sum_{n=2} F_{n-1} X^n + \sum_{n=2} F_{n-2} X^n. \text{ Now, we know that}$$

$$\sum_{n=2} a + \sum_{n=2} b = \sum_{n=2} a + \sum_{n=2} b. \text{ We apply this and so we get:}$$

$$B(X) = X + \sum_{n=2} F_{n-1} X^n + \sum_{n=2} F_{n-2} X^n. \text{ Now let's look at the first sum}$$

# Sequences, series and limits with the HP49G - Part 2

a bit closer. If the power of  $X$  were not  $n$  but  $n-1$ , then the sum would be:  $\sum_{n=2} F_{n-1} X^{n-1}$ . Do you notice something? The sum would

then be itself  $B(X)$ , the generating function. If we would write out this sum we would get  $F_1 X + F_2 X^2 + F_3 X^3 + \dots$  blahblah, which is indeed the generating function since  $F_0 = 0$ . Now, because all terms in

the sum  $\sum_{n=2} F_{n-1} X^n = F_1 X^2 + F_2 X^3 + F_3 X^4 + \dots$  blahblah share a common the common factor  $X$ , we can factor  $X$  out and get:

$$\sum_{n=2} F_{n-1} X^n = F_1 X^2 + F_2 X^3 + F_3 X^4 + \dots \text{blahblah} =$$

$$X (F_1 X^1 + F_2 X^2 + F_3 X^3 + \dots \text{blahblah}) = X \sum_{n=2} F_{n-1} X^{n-1}$$

In the expression  $X \sum_{n=2} F_{n-1} X^{n-1}$  we can change indices and say  $n-1 = j$ , where  $j$  is a new index going from 1 to  $\infty$ . So we get

$\sum_{n=2} F_{n-1} X^n = X \sum_{n=2} F_{n-1} X^{n-1} = X \sum_{j=1} F_j X^j = X B(X)$ . If we substitute this in our last state of the generating function, then we get:

$$B(X) = X + X B(X) + \sum_{n=2} F_{n-2} X^n. \text{ It starts looking like happiness.}$$

Using the same arguments like for the first sum, we can also convert the second sum  $\sum_{n=2} F_{n-2} X^n$  to  $X^2 \sum_{k=0} F_k X^k$ , which is equal to  $X^2 B(X)$ . Our equation now, looks like

$$B(X) = X + X B(X) + X^2 B(X). \text{ Enter this equation and then } B(X).$$

Press SOLVE and you get  $B(X) = -\frac{X}{X^2 + X - 1}$ . This my dear math freaks is the generating function of the Fibonacci numbers. (And the start of all teacher's nightmares ;-)) Press EQ-> to separate the two

sides of the equation. Enter  $X = 0$  and then 10. Press SERIES. When the calc is done, press SUBST and then 3 GET and DTAG. You have the magnificent series:

$$89 X^{11} + 55 X^{10} + 34 X^9 + 21 X^8 + \dots + 3 X^4 + 2 X^3 + X^2 + X$$

Ho! The coefficients *are* the Fibonacci numbers! The generating function  $B(X) = -\frac{X}{X^2 + X - 1}$  has just generated them!

Another example without explanations. Let's do that for the RAM chips recurrence,  $D(k) = 2 D(k-1)$ , with  $D(0) = 1$  and  $k = 0, 1, 2, 3, \dots$ .

$$B(X) = \sum_{k=0} D(k) X^k = 1 X^0 + \sum_{k=1} 2 D(k-1) X^k =$$

$$1 + 2 \sum_{k=1} D(k-1) X^k = 1 + 2 X \sum_{k=1} D(k) X^k = 1 + 2 X B(X)$$

We have found that  $B(X) = 1 + 2 X B(X)$ . Solving this for  $B(X)$  we find  $B(X) = \frac{1}{1-2X}$ , the generating function of  $D(k) = 2 D(k-1)$ , with  $D(0) = 1$  and  $k = 0, 1, 2, 3, \dots$ . The series expansion of

$$B(X) = \frac{1}{1-2X} \text{ at the point } X = 0 \text{ gives us the series } 1 + 2X + 4X^2 + 8X^3 + 16X^4 + \dots, \text{ whose coefficients are the numbers of the sequence } D(k) = 2 D(k-1), \text{ with } D(0) = 1 \text{ and } k = 0, 1, 2, 3, \dots$$

We have a mechanical (=implementable as an algorithm) method to turn such sequences to something easier to handle with. Now, if we also had a method to find how these numbers, the coefficients of the series expansion of the generating function, depend on  $n$

# Sequences, series and limits with the HP49G - Part 2

(symbolically, not numerically) we would be able to calculate really big Fibonacci or whatever numbers in seconds. The method shown here for finding the generating function will be programmed some pages later. (Yes, it is possible to program this method on the HP49G, but be patient, the student's revenge will come later ;-)) Unfortunately the HP49G can't give you the coefficients of a series expansion, if you do that pure symbolically. If you enter  $F(X)$ ,  $X=0$  and 4, and press SERIES, then it complains "Operator not implemented (SERIES)". (This is another error that you can't catch in error traps, it just escapes from the trap and pops up in front of your eyes.) So we can't use generic SERIES expansion for finding an analytic closed form for the coefficients. For this reason we will use here the *approach of the characteristic polynomial* which is also possible to program on the HP49G. Nonetheless we will make a program for calculating the generating function, even if we don't use it now for getting the analytic closed form of a series. We do that for two reasons: First to demonstrate how the rich command set of the HP49G can be used for very "unusual" things. Second because it is possible that someone of the guys out there programs a generic series expansion, which then can be applied on the generating function and give us the symbolic dependance of the coefficients on the sequence index  $n$ . For now, let's see how the method of the characteristic polynomial works. Again we take the example of the Fibonacci numbers. We give here mainly the "mechanical" part. If someone wants more mathematical proof, then any good mathematics book about sequences and series will give more information.

The method starts letting the  $n^{\text{th}}$  Fibonacci number  $FIBO(n)$  be a characteristic polynomial  $r^n$ . Then, we plug these characteristic polynomials in the definition of the Fibonacci numbers:

$$FIBO(n) = FIBO(n-1) + FIBO(n-2) \quad r^n = r^{n-1} + r^{n-2} \quad \boxed{r^2 - r - 1 = 0}$$

We solve the last equation  $r^2 - r - 1 = 0$  for  $r$ . The HP49G can solve this symbolically. (Of course it can, it is a quadratic equation in  $r$ .)

SOLVE returns the solutions list:  $r = -\frac{-1+\sqrt{5}}{2} \quad r = \frac{1+\sqrt{5}}{2}$ .

Now, the general solution of  $FIBO(n) = r^n$  is any linear combination of the roots. That means:

$$FIBO(n) = C1 \cdot \left(-\frac{-1+\sqrt{5}}{2}\right)^n + C2 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n$$

It starts looking very bad for teachers. If we now find what  $C1$  and  $C2$  are, then we catch them by surprise. We can find what these  $C1$  and  $C2$  are if we use the initial conditions  $FIBO(0) = 0$  and  $FIBO(1) = 1$ . For  $n = 0$  we have:

$$FIBO(0) = 0 \quad C1 \cdot \left(-\frac{-1+\sqrt{5}}{2}\right)^0 + C2 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^0 = 0$$

$$C1 + C2 = 0$$

For  $n = 1$  we have:

$$FIBO(1) = 1 \quad C1 \cdot \left(-\frac{-1+\sqrt{5}}{2}\right)^1 + C2 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^1 = 1$$

$$C1 \cdot \left(-\frac{-1+\sqrt{5}}{2}\right) + C2 \cdot \left(\frac{1+\sqrt{5}}{2}\right) = 1 = 1$$

The system of equations:

$$C1 + C2 = 0$$

$$C1 \cdot \left(-\frac{-1+\sqrt{5}}{2}\right) + C2 \cdot \left(\frac{1+\sqrt{5}}{2}\right) = 1$$

is a linear equations system in  $C1$ ,  $C2$ . The command LINSOLVE is exactly what we need for solving such systems. We enter the vector of

# Sequences, series and limits with the HP49G - Part 2

equations,  $C1 + C2 = 0$   $C1 - \frac{-1+\sqrt{5}}{2} + C2 \frac{1+\sqrt{5}}{2} = 1 = 1$ ,

then the vector of the variables to solve for,  $[C1 \ C2]$ , and then use LINSOLVE. The command returns among other information that we

don't need here, the vector of solutions:  $C1 = \frac{-\sqrt{5}}{5}$   $C2 = \frac{\sqrt{5}}{5}$ .

Substituting these solutions in the analytic closed form of the Fibonacci numbers, we get

$$\text{FIBO}(n) = \frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^n + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^n$$

You don't believe that the Fibonacci numbers can be calculated with this formula, using square roots and other things that are not likely to return integer results? Don't worry, I can't believe that too! We enter

$$\text{FIBO}(n) = \frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^n + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^n \text{ and press DEF.}$$

Now the function FIBO is ready for use. Come on FIBO, give us some Fibonacci. Enter a 0 and press the menu key of FIBO in the menu VAR. After 2 very suspended seconds we have the result

$$\frac{-\sqrt{5}}{5} + \frac{\sqrt{5}}{5} \text{ which we EXPAND to get 0, the first Fibonacci number.}$$

Next one please. Enter 1, press FIBO. You get the result,

$$\frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2} + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}. \text{ EXPAND it and you have 1, the}$$

second Fibonacci number. The final EXPAND is a bit too much to press it after every run of FIBO. We recall FIBO on the stack and add EXPAND at the end of the program:

```
<<
->
'- 5/5*((-((-1+ 5)/2))^n+ 5/5*((1+ 5)/2)^n'
EXPAND
>>
```

We store that in FIBO and go ahead. Enter 2, press FIBO. Result is 1, OK. Enter 3, press FIBO to get 2. 4 FIBO is 3, 5 FIBO is 5 and so on. Now for the big ones. Enter 20, press FIBO. You get 6765. If you want to accelerate the calculation switch to numeric mode. You lose accuracy and you can the right results only up to 999999999999. 30 FIBO gives 832040. 40 FIBO gives 102334155. 50 FIBO gives 12586269025. In numeric mode it works very very fast, in exact mode it is not so fast. And for big big FIBOS which can't be calculated in numeric mode it is very slow. So it seems that after all we have to use

```
<< -> n
<<
CASE
{ 6. 7. 9.} @If the argument is a local or
n TYPE POS @global name or algebraic
THEN @
'APPLY(FIBO,n) @Just apply FIBO
EVAL @on n and return FIBO(n)
END
n 1 @If n is equal to 1
THEN
0 @return 0
END
0 1 2 n @If nothing of the above is
START @true, calculate FIBO
DUP ROT +
NEXT
NIP
END
>>
```

It is faster than the analytic version because it takes so long to

$$\text{EXPAND for example } \frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^{1000} + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^{1000}.$$

It looks for us like one single command, namely EXPAND the monster. But the HP49G must fire up an algorithm to expand this

# Sequences, series and limits with the HP49G - Part 2

thing. And the algorithm itself must do many many things when expanding such beasts. The program with the START-NEXT loop needs 10 seconds to return the right result (huge integer with 209 digits). The analytic version..., well I didn't time it because I lost my patience. So did the teachers win, after all? No! We won a wonderful insight in the secret life of sequences. And we will see that we also win ways to get information about sequences that seem impossible to handle. Before we go on with the programs, one last remark about the Fibonacci. They seem to be so easy to understand, because their construction recipe is so easy. But would you ever imagine that they have to do with geometry? Did you notice what the analytic closed form of them contains? No? Look again! Oh, what a marvel! The

golden section, the divine proportion!  $= \frac{1+\sqrt{5}}{2}$  And also its

counter part  $= \frac{1-\sqrt{5}}{2}$  ! The Fibonacci numbers, constructed out of the divine proportion! Their analytic closed form is equivalent to: 
$$\text{FIBO}(n) = \frac{\sqrt{5}}{5} \left( \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right)$$
 . This insight alone was worth the long

excursion, wasn't it? Now, just sit and wonder if Fibonacci knew what he was initiating when he discovered his numbers. And a philosophical question: Was this relation  $\text{FIBO}(n) = \frac{\sqrt{5}}{5} \left( \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right)$

existent at the moment of the discovery of the Fibonacci numbers? Was it then existent but not discovered? If the definition of the Fibonacci exists, then this formula also exists but we don't know that it exists until we find it? Or does it come into existence at the very moment of its *own* discovery? Think of it, when you have time. But if you tend to say that the formula existed when the Fibonacci were discovered, then ask yourself: Did the Fibonacci exist from the moment on, when the integers were discovered? Fascinating!

Enough philosophy for the moment. Let's get on the dirtier part of the job, coding. First of all we must think about our notation of sequences. We used lists that have the form

$\{\text{memberGeneralForm}\{\text{indexstart end}\}\}$  to represent sequences.

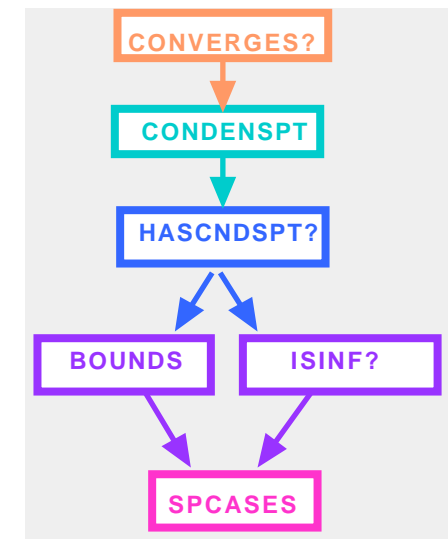
If possible we should generalise this notation to include recurrences. We add the following variation of this representation:  $\{\text{recurrenceEquation}\{\text{initialCondition}\}\}$ . Now, our sequences can be of two forms, the one that we had until now and in addition the new form for recurrences. For example, for the Fibonacci sequence we would write:

$\{\text{FIBO}(n) = \text{FIBO}(n-1) + \text{FIBO}(n-2) \{\text{FIBO}(0) = 0 \text{ FIBO}(1) = 1\}\}$

The sequence of the RAM chip numbers would be denoted as:  $\{\text{D}(n) = 2 \text{ D}(n-1) \{\text{D}(0) = 1\}\}$ . Our programs should then check to

see what kind of sequence they have to do with and act accordingly. If you remember the whole group of programs that we made until now was structured like the picture at the right shows. All of the programs base on SPCASES. If we use this structure further, we only need to change SPCASES, or any program below SPCASES, so that the kind of sequence is checked. In case of recurrence, we try to find an analytic closed form, then construct an equivalent analytic sequence of the first kind, and replace the recurrence with this equivalent form. The programs above SPCASES will only see

what SPCASES shows them, namely the possible special cases of a normal analytic sequence, with which they can do something. The check for the kind of the sequence can be made very thorough by checking for example if the first element of the sequence list contains a "=" and if all elements of the sub list also contain also a "=". But we





# Sequences, series and limits with the HP49G - Part 2

restrict ourselves to check if the first element of the sequence list contains a "=", in which case we assume a recurrence. The interested reader could make additional checks and let the program error out "civilised" without leaving the stack totally messed up.

We start giving the code of some smaller auxiliary programs that we can use also for other purposes. We will need them later for the bigger programs. The HP49G can't distribute the symbolic sum over two or more terms in the summand. We can't transform for example

$$\sum_{n=0}^{\infty} a_n + \sum_{n=0}^{\infty} b_n$$
 to 
$$\sum_{n=0}^{\infty} a_n + \sum_{n=0}^{\infty} b_n$$
. And we need this capability for a program that would calculate generating functions. We could also find such a capability useful for other algebraic manipulations in future marathons. So, let's program it. The strategy is to convert the whole summand to a sum of terms and then match all patterns of the form

$$\sum_{n=0}^{\infty} A + \sum_{n=0}^{\infty} B$$
 to 
$$\sum_{n=0}^{\infty} A + \sum_{n=0}^{\infty} B$$
 and of the form 
$$\sum_{n=0}^{\infty} -A$$
 to

$$\sum_{n=0}^{\infty} -A$$
 until nothing changes anymore. ( We will see later why we need the second transformation.) The next program implements this strategy.

```
<< ->LST 1          @Convert to a list for sub-
<<                  @sequent DOSUBS
  IF                  @If current element is
    {                } OVER POS
  THEN                @then apply FDISTRIB to the
    SWAP FDISTRIB SWAP@summand and convert it to a
  END                  @sum of terms
>>
DOSUBS                @Do to each list element
->ALG                  @Re-convert to algebraic
WHILE                  @MATCH until nothing changes
  { ' (&V=&L,&H,&A+&B) '
    ' (&V=&L,&H,&A)+ (&V=&L,&H,&B) ' }
  MATCH
```

```
REPEAT
END
WHILE                  @MATCH until nothing changes
  { ' (&V=&L,&H,-&A) '
    ' - (&V=&L,&H,&A) ' }
  MATCH
REPEAT
END
>>
```

We store this in FDISTR. Perhaps you have already asked yourself, why we don't need to match also the patterns that eventually have the form ' (&V=&L,&H,&A-&B) ' to corresponding patterns that have the form ' (&V=&L,&H,&A)- (&V=&L,&H,&B) ' }. This is because of two reasons that in this case combine wonderful to let us do what we want. First: When ->LST is applied on an algebraic object that contains a sum, then the whole summand is returned as a single object. For example, if you apply ->LST to 'X+ (n=0,10,n^2-1)' then the returned list is { X n 0 10 'n^2-1' + } rather than { X n 0 10 n 2 ^ 1 - + }. Second: If the summand is some factored expression, then FDISTRIB will convert it to a sum, that contains terms combined *all* with +. That means than for example 'a\*(x-c)' will be converted to 'a\*x+-(a\*c)' rather than 'a\*x-a\*c'. So we can be sure that the only match needed is matching ' (&V=&L,&H,&A+&B) ' to ' (&V=&L,&H,&A)+ (&V=&L,&H,&B) '. The Professor knew exactly what he did when he made the CAS. (Sure he did, that's why he is the Professor! ;-)) The second match is needed for having only positive terms in the sums, because the programs later will love this property.

As always, we test FDISTR now. Enter the sum

$$\sum_{index=1}^{U-3} (\sin(X) - \cos(index \ X)) e^{(index+1)}$$
. Press FDISTR to get

$$\sum_{index=1}^{U-3} \sin(X) e^{(index+1)} - \sum_{index=1}^{U-3} \cos(index \ X) e^{(index+1)}$$
. It works.

If someone of you out there finds an example in which FDISTRIB

# Sequences, series and limits with the HP49G - Part 2

behaves differently, please please post it. The whole program and the next programs are based on this property.

The next small auxiliary program that we need, is a program that returns all terms of an algebraic expression.

```
<< EQ-> - FDISTRIB 1      @Convert a=b to a-b
->LIST { } -> sums terms  @and store in locals
<<                          @While the list sums is
    WHILE                  @not empty, repeat
        sums SIZE 0 >
    REPEAT                  @Show some message to
        "Sums: " sums SIZE + " @Trabakoulas
" + "Terms: " + terms SIZE +
    1 DISP sums            @get head of sums
    HEAD sums TAIL 'sums'  @Store the rest in sums
    STO -> obj              @Store head in local obj
    << IF                   @If obj is real, integer
        { 0. 6. 28. } obj @or name
        TYPE POS
    THEN                    @then add it in terms
        terms obj +
        'terms' STO
    ELSE                    @else
        obj OBJ->          @explode it
        IF                 @If the obj was two arguments
            { + } SWAP @hold together with a "+"
            POS
        THEN                @then add both arguments in
            DROP sums      @sums
            ROT +
            SWAP +
            'sums' STO
        ELSE                @else add obj in terms
            DROPN terms
            obj +
            'terms' STO
    END
```

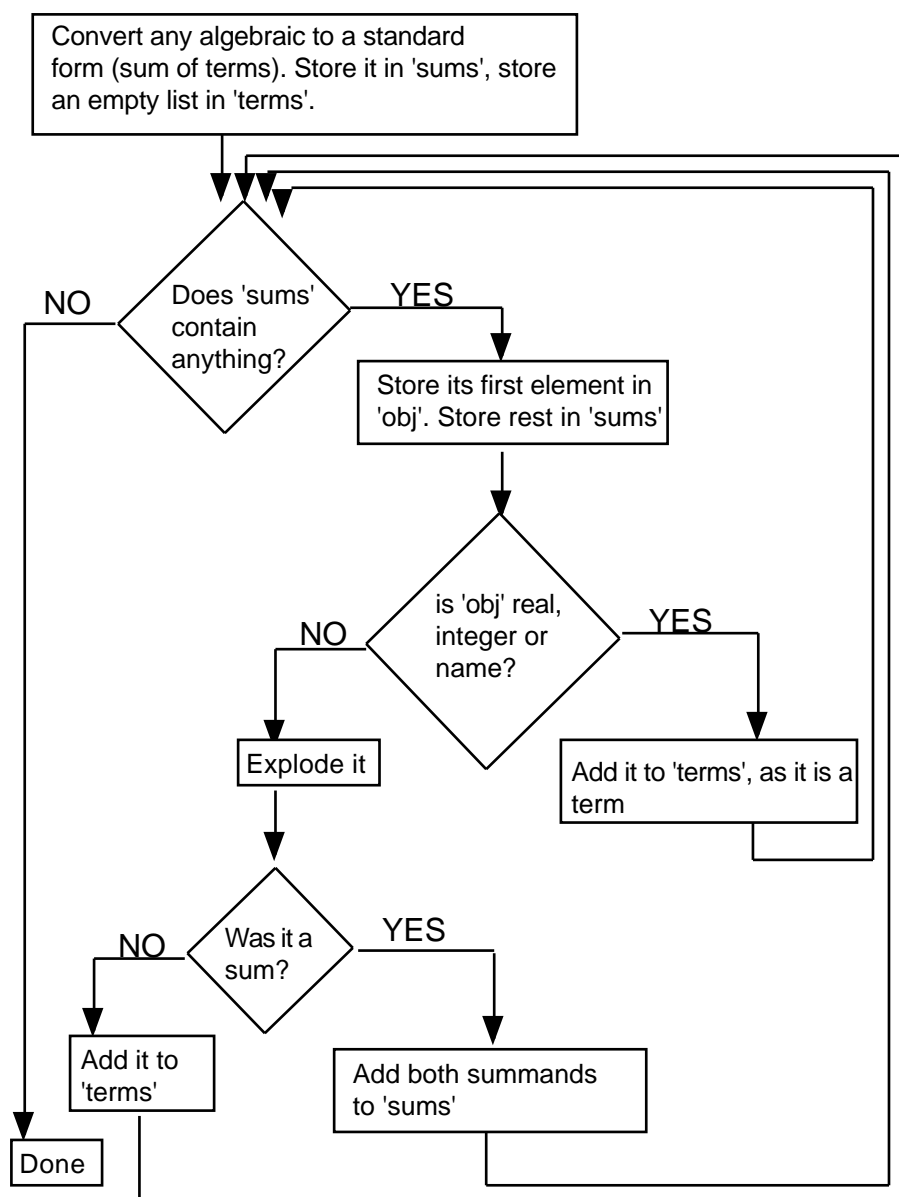
```
END
>>
END terms      @return terms
>>
>>
```

We store this in ->TERMS. It is good to see how this program works, so let's make a flow diagram (next page). The program first turns equations to differences using EQ->. If EQ-> has an argument that wasn't an equation, then it returns the argument and a 0, which if subtracted from the argument doesn't change anything at all. So we make sure that it will work for equations and expressions. The whole expression is put in a list 'sums' and an empty list is put in 'terms'. The program keeps on exploding the first element of sums, and if it sees that the object just being exploded was itself a sum, it just adds the arguments of + to the end of the list 'sums', because they could be themselves sums. If the exploded expression wasn't a sum, then it adds it to 'terms'. In every pass the expression that is exploded is also removed from the first position of 'sums'. The additional check if the expression is real, integer or name is done, because applying OBJ-> to such an object would error with "Bad Argument Type". Since we know, that if we have a real number or integer or name this was a term, we add them also to 'terms'. When all expressions of 'sums' have been processed, then we are ready and returns the list terms. The FDISTRIB at the start of the program again makes sure that we have only arguments hold together with a +. (Like in FDISTR .)

And now for the test. Enter some equation, like for example  $e^x (\sin(x) - x (\cos(x) + 3)) = P(x) (x + 3 x^2)$ . Press now the soft key ->TERMS. You get as result a list containing all terms of the equation  $\{\sin(x) e^x -(x P(x)) \dots -(3 x e^x)\}$ . Keep in mind however that the terms are not in the same order as in the equation.

And now the main programs. We make a program that takes a list representing a sequence and finds the type of the sequence. As said before, we have two main types of sequences. First, the analytic

# Sequences, series and limits with the HP49G - Part 2



sequences  $\{\text{memberGeneralForm}\{\text{indexstart end}\}\}$ . We give such sequences the type 1. If the program finds such a sequence, it returns the sequence unchanged and a 1. Then we have the recurrence sequences  $\{\text{recurrenceEquation}\{\text{initialConditions}\}\}$ . We have two variations of these sequences. The first is of type  $\{P(n+1) = \text{someSumOfPreviousMembers}\{\text{initialConditions}\}\}$ . The expression "someSumOfPreviousMembers" denotes expressions like for example  $P(n-1) + P(n-2)$  or  $3 P(n-1) + 2 P(n-2) - P(n-3)$  but not  $P(n-1) + \frac{P(n-2)}{P(n-3)}$ . Such sequences we will handle using the method of characteristic polynomials. We give such sequences the type 2. When the program finds such a sequence, it returns a list of the form  $\{\text{terms factors initConditions index}\}$  and a 2. "Terms" denotes the terms  $P(n)$ ,  $P(n-1)$ , and so on, that appear in the sequence. "Factors" are the factors with which these terms are multiplied. "InitConditions" is the list of initial conditions and "var" is the index of the sequence. (Why we do that? We will see in a few minutes.) The second variation of recurrences is of the type  $\{P(n+1) = P(n) + \text{someFunctionOfTheIndex}\{\text{initialConditions}\}\}$ . The term "someFunctionOfTheIndex" denotes something that doesn't contain any sequence member. This type of recurrence sequence will we will handle using SIGMA. We give such sequences the type 3. If the program finds such a sequence, it returns a list  $\{\text{terms factors initConditions index matches}\}$  and a 3 on the stack. The additional element "matches" is a list that indicates with a 1 which terms in "terms" are members of the sequence, like  $P(n)$ ,  $P(n-1)$  and so on, and with a 0 which are not. The code is on the next page.

# Sequences, series and limits with the HP49G - Part 2

```

<<
-> seq                @Store in local variable
<<
  "Checking sequence type" @Message in a bottle
  1 DISP
  IF                    @If the first element
    "Analytic closed form" @of the sub list in the
    2 DISP              @sequence is a name (the
    seq 2 GET HEAD      @sequence index)
    TYPE 6 ==
  THEN                  @then just return the
    seq 1                @sequence and a 1
  ELSE                  @else we must make some
    "Extracting terms"  @some checks
    2 DISP
    seq HEAD ->TERMS    @return all terms
    DUP HEAD OBJ-> NIP  @get first term and use
it
    "" + "" SWAP +      @to construct the pattern
    "(&N)'" + OBJ->      @P(&N), P being the name
                        @of sequence members
    { } NOVAL seq        @Store in local variables
    2 GET -> terms var   @all that we need to
    seqpatt mtch factors @proceed.
    bcond
      <<
        CASE            @In case
          "Recurrence 1"
          2 DISP
          terms 1        @Match any occurrence of
          << seqpatt 1    @type P(&N) with a 1. If
            2 ->LIST      @we had a match, then any
              MATCH 'mtch' @term of the form
              SWAP STO+    @factor*P(&N) will be
              EXPAND       @matched to factor*1
            >> DOSUBS
            mtch           @Stream AND over the
            << AND         @matches flags

        >> STREAM
        SWAP DUP          @Store matched factors in
        'factors'         @local 'factors'
        STO
        1                 @Find the positions of
        << LNAME NIP       @all terms which do not
          IF DUP           @contain the seq. index
          TYPE 5
          THEN AXL END var POS NOT
        >> DOSUBS
        1 +               @Add a 1 in the list of
        << AND             @positions and stream
        >> STREAM         @AND
        AND               @If we had only P(&N) and
                        @no other expressions
                        @with the seq. index
                        @then return the approp.
                        @list and a 2.
        THEN
          terms
          factors bcond
          var 4
          ->LIST 2
        END
        @In case
        "Recurrence 2" 2 DISP
        mtch            @Return matches list
        factors         @and factors list
        2               @Check if we have only
        << LNAME NIP     @factors of P(&N) not
          IF DUP        @containing the seq. index
          TYPE 5        @or other factors
          THEN AXL
          END
          IF SWAP
          THEN var POS NOT
          ELSE DROP 1
          END
        >> DOLIST
        1 +             @Add 1 to the result list
        << AND           @stream AND over the result list

```

# Sequences, series and limits with the HP49G - Part 2

```

>> STREAM
mtch LIST      @Check if only two terms
2 == AND       @match pattern P(&N)
factors mtch
2              @Get the factors of
<<            @P(&N)
  IF NOT
  THEN DROP
  END
>> DOLIST      @Check if factor of first
DUP HEAD       @P(&N) is 1 and of second
1 == SWAP 2    @is -1 (Diff. of two seq.
GET -1 == AND  @members
AND
THEN           @If conditions OK then
  terms factors @return approp. list and
  bcond var     @ a 3
  mtch 5 ->LIST 3
END            @Else return sequence and
  seq 0         @a 0 (we don't handle
such           @sequences)
  END
  >>
  END
  >>
  >>
  >>

```

Store in SEQTYPE. Now the obligatory test of SEQTYPE. Enter the analytic type sequence  $(-1)^n \frac{n^2 - 3}{3n^2} \{n \ 0 \}$  and press

SEQTYPE. Almost immediately you get the sequence unchanged and a 1. The program recognised that as an analytic type sequence. Enter  $\{P(n) = P(n-1) + P(n-2) \mid P(0) = 0 \ P(1) = 1\}$  and press

SEQTYPE. The program shows some messages and returns the list  $\{\{P(n) \ -P(n-1) \ -P(n-2)\} \ \{1 \ -1 \ -1\} \ \{P(0)=0 \ P(1)=1\} \ n\}$  with the summands of the recurrence, their factors the initial conditions

and the index variable. It also returns a 2 because this is a recurrence of first type. (Which we handle with the method of characteristic polynomials later on.)

Enter the sequence  $P(n) = P(n-1) + \frac{3}{(n+1)^2} \mid \{P(0)=0\}$  and press

SEQTYPE. The program returns the list

$$P(n) \ -P(n-1) \ -\frac{3}{(n+1)^2} \ \ 1 \ -1 \ -\frac{3}{n^2+2} \ \frac{3}{n+1} \ \{P(0)=0\}$$

$$n \ \{1. \ 1. \ 0.\}$$

with the summands of the recurrence, their factors (or the summand itself in case it is not of the form  $P(\&N)$ ), the initial conditions, the index variable and a list that indicates which of the summands are of the form and  $P(\&N)$ . It also gives a 3 because it recognised the list as recurrence of type 2, which we will handle later on with SIGMA. If you enter some other sequence, like for example  $\{P(n) = P(n-1) + P(n-2) \mid P(0) = 0 \ P(1) = 1\}$  which we don't handle (yet), then the program returns the sequence unchanged and a 0.

The results of SEQTYPE are needed by the next program that we make, RCR->ANL. It takes a sequence and if it is a recurrence (of the types that we handle) it converts it to an analytic closed form. I think you see now what we aim at. We want to convert recurrences to analytic closed forms, in order to let them examined by the other programs that we have already from the first part. This has also the advantage that we don't need to change the programs from the first part, except for minor modifications.

Note: The program that comes along with this document is named RCR2ANL. After you transferred it to your HP49G, you must rename it to RCR->ANL.

# Sequences, series and limits with the HP49G - Part 2

Let's take a look at the code of RCR->ANL now.

```
<<
PUSH      @Save user's settings
{ -114 -128 -123 } @ Display powers
CF        @in ascend. order,
          @allow compl. vars
          @allow mode switch
{ -103 -109 -120 } @real mode,
SF        @num. factorize
          @silent
-> seq    @store in local
  <<
  seq SEQTYPE @find seq. type
  CASE      @in case of seq.
    DUP 2 == @type 2
  THEN
    DROP OBJ-> @explode list
    DROP{ } { } @store in locals
    -> terms
    factors bcond
    var uc ninit
    <<
    "Creating @Disp. message
    characteristic
    polynomial"
    1 DISP
    terms factors
    / @Divide terms by their factors
    EXPAND @and expand
    1
    <<
    @Extract index out of
    { '&F(&N)' &N } @of the form '&F(&N)'
    MATCH DROP
  >> DOSUBS
  var bcond 1
  <<
  @Extract start values of
  EQ-> DROP @index from initial
```

```
{ '&F(&N)' &N } @(boundary) conditions
MATCH DROP
>> DOSUBS
DUP 'ninit' STO @Store these indices
+ @Add to the list of
<< MIN @initial indices (why?)
>> STREAM = @and find the smallest
SUBST EXPAND @Substitute the smallest
DUP + @Subtract smallest index
<< MIN
>> STREAM -
XQ RCLVX @Construct X^index for
SWAP ^ @every index
factors SWAP * @Multiply each power of X
0 + @with its factor. Add 0
LIST @to the list (why?). Find
@sum of list.
"Solving" 1 @Message in a bottle.
DISP
FROOTS AXL 1 @Use FROOTS to find roots
<< @If we have a root
IF @multiplicity
  NSUB 2
  MOD NOT
  THEN
  IF @which is greater than 1
    DUP 1 >
  THEN @then
    1 SWAP @repeat it the appr.
    1 - @amount of times
    START
    DUP
    NEXT
  ELSE @else drop root multiplicity
    DROP
  END
END
>> DOSUBS
```

# Sequences, series and limits with the HP49G - Part 2

```

var ^      @Raise root to the power n
"Multiplying roots with
unknown coefficients
"
1 DISP      @Message again
1
<<          @Construct coeffs. C1, C2,...
  uc "C"
  NSUB R->I
  + OBJ-> +
  'uc' STO
>> DOSUBS uc
SWAP * 0 +   @Multiply then with roots^n
LIST DUP     @Find sum of Cn*X^n
"Substituting initial   @Message
conditions"
1 DISP
var ninit =   @Substitute init. conds.
SUBST
bcond 1       @Create equations P(n)=initCond
<< EQ-> NIP
>> DOSUBS =
"Solving linear system   @Message
for "
uc ->STR + 1 DISP
AXL           @Solve linear system for
uc AXL LINSOLVE @C1, C2 ...
"Substituting solutions"
1 DISP      @Message
UNROT DROP2  @Substitute C1, C2 back
AXL 1
<< SUBST
>> DOSUBS
-105 CF      @Set again exact mode
var ninit     @Find smallest init. index
+
<< MIN
>> STREAM

3 ->LIST      @Create analytic seq. as
2 ->LIST      @the other programs want them
>>
END
DUP 3 ==      @Sequence of type 3
THEN
DROP OBJ->
DROP { }
-> terms      @Store in locals
factors bcond var
mtch ninit
<<
"Determining analytic
closed form"
1 DISP      @Message
var bcond
HEAD         @get first init. cond.
EQ-> DROP    @make equation n=smallestIndex
{ '&F(&N)' &N }
MATCH DROP =
'ninit' STO
terms
mtch 2
<<          @If we have a term of the form
  IF        @P(n) then drop it.
  THEN DROP
  END
>> DOLIST
0
+ LIST NEG    @Add all terms not of the form
              @P(N)
IF            @If sum trig. function appears
  DUP ->LST
  DUP { SIN }
  HEAD POS
  OVER { COS }
  HEAD POS

```

# Sequences, series and limits with the HP49G - Part 2

```

OR SWAP { TAN }
HEAD POS OR
THEN          @then,
  var          @make assumptions
  UNASSUME      @and convert to complex
  DROP ninit    @exponential. Also set flag 1
  EQ-> 1 + >=    @as an indicator for later.
  ASSUME DROP
  EXPLN 1 SF
END
var SIGMA      @Use SIGMA to find P(n+1)-P(n)
"Substituting initial
conditions"
1 DISP          @Message
'C' + DUP        @Substitute init. conds.
bcond HEAD
EQ-> NIP =
ninit '0=1' +    @Add equations n=start and
                  @0=1 to get n=start+1
SUBST
'C' SOLVE        @Solve for C and
SUBST            @substitute back
var DUP 1 + =    @Substitute n=n+1
SUBST EXPAND
LIN              @Linearize
IF              @If trigs were involved
  1 FS?
THEN
  DUP RE          @Find real part
  -103 CF         @Expand in real mode to avoid
  EXPAND          @complicated forms
  TCOLLECT        @Collect trigonometrics
  EXPAND
  SWAP            @Do the same again for the
  -103 SF IM      @imaginary part.
  -103 CF
  EXPAND
  TCOLLECT

```

```

EXPAND i * + @Construct real+i*imaginary
END          @Make analytic sequence as the
ninit        @other programs want it.
EQ-> -105 CF
3 ->LIST
2 ->LIST
>>
END
DUP          @Sequence of type 0?
0 ==
THEN
  DROP POP    @Stop here! We don't handle that (yet)
  "Can't deal with this
  kind of sequences"
  DOERR
END
DROP          @Else drop type
END
>> POP        @Restore user's settings
>>

```

Before we test the program, a couple of words about LINSOLVE, about FROOTS and about adding equations. LINSOLVE will take a vector of linear equations from stack level 2, and a vector of unknowns from stack level 1, and will return the vector of solutions for the unknowns along with other information. For example, entering  $[2 \ X - Y = 1 \ X + Y = 3]$  and  $[X \ Y]$  and then pressing

LINSOLVE will return the vector of solutions  $X = \frac{4}{3} \ Y = \frac{5}{3}$  on stack level 1, and other information on levels 2 and 3 ( which we will describe more detailed in some future marathon).

FROOTS solves polynomials of the current VX but it doesn't return a list of solutions of the form  $\{X = \text{sol1} \dots\}$ . Instead of this it returns a vector of roots and multiplicities. When it is not only important to know the roots but also their multiplicities (like in our case here), it is better to use this command. For example, using SOLVEX to solve



# Sequences, series and limits with the HP49G - Part 2

$X^3 - 5X^2 + 8X - 4$  will return  $\{X = 1 \ X = 2\}$ , which doesn't tell us which root is double. The polynomial could be  $(X - 1)(X - 1)(X - 2)$  or  $(X - 1)(X - 2)(X - 2)$ . But FROOTS will return  $[1 \ 1. \ 2 \ 2.]$ , which tells us that the first root 1 is a single root (1.) while the other root 2 is a double root (2.). That means that the polynomial is  $(X - 1)(X - 2)(X - 2)$ .

Perhaps you noticed that we add a strange equation  $0 = 1$  to the equation  $n = \text{startIndex}$ . This is only an abbreviation for adding nothing (0) to the left hand side and 1 to the right hand side. The HP49G can also add equations with the normal command  $+$ . If two equations are added, then the sides of the equations are added separately to each other, and we (ab)use this to make for example  $n = 2$  out of  $n = 1$ . If an equation and an expression are added together, then the expression is added to both sides of the equation. For example  $X = 3, 5, +$  will return  $X + 5 = 8$ .

And now for the tests. Enter  $(-1)^n \frac{n^2 - 3}{3n^2} \{n \ 0 \}$  and press

RCR->ANL. Since this is already analytic, the program returns it unchanged almost immediately.

Enter our old friend, the Fibonacci sequence

$$\{P(n) = P(n-1) + P(n-2) \quad \{P(0) = 0 \quad P(1) = 1\}\}$$

and press RCR->ANL again. After some 23.5 seconds you get the analytic closed form of the Fibonacci sequence:

$$\frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^n + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^n \{n \ 0 \}$$

Hurrah! Notice however: You must enter such sequences in exactly this form. The member with the biggest index alone in the left hand side and the rest in the right hand side of the recurrence, and the initial conditions in ascending order of the index variable.

Now another recurrence of the second type. Enter

$$P(n) = P(n-1) + \frac{3}{(n+1)^2} \quad \{P(0) = 0\} \quad \text{and press RCR->ANL.}$$

About 22 seconds later you get the awaited answer

$$-\frac{6 \text{ PSI}(n+1+1,1) - (2-6)}{2} \{n \ 0 \}$$

And one with trigonometric functions. Enter the sequence

$$P(n) = P(n-1) + \cos \frac{n}{3} \quad \{P(0) = 0\} \quad \text{and press RCR->ANL.}$$

After looong 106 seconds the HP49G says:

$$\frac{2 \cos \frac{(n-1)}{3} - 1}{2} \{n \ 0 \}$$

We now move on to the next program, the program that finds generating functions for the same types of recurrences that REC->ANL also handles. We don't use generating functions for anything now, but because they are so useful we should make a program that can find them. (And who knows what we'll do in future ;-))

First of all, here is the commented code of GENFUNC.

```
<<
DUP SEQTYPE NIP          @Find sequence type
IF
  2 ==                    @If it is of type 2
THEN
  OBJ-> DROP OVER
  EQ-> DROP OBJ->
  DROP2 ->
  seq bcond var           @Store sequence, init conds,
  <<                      @and index variable in locals
```

# Sequences, series and limits with the HP49G - Part 2

```

' (itVar=0, ,COEFF(itVar)*X^itVar)'
"Inserting sequence in
" OVER + 1 DISP @Message
'COEFF(itVar)'
seq EQ-> DROP
2 ->LIST MATCH @Match P(n) in series
DROP
"Replacing itVar with
"var + "
" + 1 DISP @Message
'itVar' var 2 ->LIST @Match index var. in
MATCH DROP @series
1 bcond
SIZE
FOR I
"Extracting boundary @Message
condition " bcond I GET + "
" + 1 DISP
bcond I GET EQ-> @Extract known initial
NIP 'X' I 1 - R->I ^ @members of the series
* SWAP ->LST DUPDUP @out of the sum
{ } HEAD POS 3 -
GET 1
+ OVER { } HEAD POS
3 - SWAP PUT ->ALG +
NEXT
"Inserting recurrence
in " OVER + "
" + 1 DISP @And another message
seq EQ-> 2 ->LIST @Match P(n) with its recurrence
MATCH DROP @definition in the series
"Distributing over +
" 1 DISP @Yet another message
FDISTR @Use FDISTRIB to distribute
"Extracting series

```

```

" 1 DISP
DUP ->LST 1
<<
IF
{ } OVER
POS
THEN
5 ->LIST
->ALG
END
>> DOSUBS
1
<< @Hold the sums, throw away rest
IF
DUP TYPE 9
== NOT
THEN
DROP
END
>> DOSUBS
"Extracting common
factors" 1 DISP
DUP 1
<< @Create list of common factors
{ } 0 -> @of the summands in the sum.
comFact comPower @These factors can be
<< @extracted out of the sum
DUP OBJ->
DROP2 4 ROLL2 3 DROPN
FACTORS
"Raising factors to
powers" 1 DISP 1
<< @Raise to appropriate powers
IF
NSUB 2
MOD NOT
THEN
R->I

```

# Sequences, series and limits with the HP49G - Part 2

```

      ^
      END
      >> DOSUBS
      "Checking if factors
independent of " var + 1 DISP 1
      <<
      @Add to list of common factors
      IF
      @only if independent of index
      LNAME
      @variable
      IF
      DUP
      TYPE 5
      THEN
      AXL
      END var
      POS NOT
      THEN
      comFact SWAP +
      'comFact' STO
      END
      >> DOSUBS
      "Getting index of @Get common powers of X
series" 1 DISP
      seq EQ-> DROP
      var &N = SUBST &N
      2 ->LIST
      -> rl
      << 1
      <<
      IF
      DUP
      rl MATCH
      THEN
      'comPower' STO
      ELSE
      DROP
      END
      >> DOSUBS
      >>

```

```

power "
      "Extracting common
      @Extract common
      @powers of X
      @out of series
      comPower + " of X" +
      1 DISP 1
      <<
      IF
      { 'X^&N' &N } MATCH
      THEN
      comPower - EXPAND X
      SWAP ^ comFact SWAP +
      'comFact' STO X
      comPower ^
      END
      >> DOSUBS
      "Creating new summand
      " 1 DISP
      CASE
      DUP { } SAME
      @If no common powers of X
      @the return a 1
      THEN
      1
      END
      DUP SIZE 1 ==
      @If only one common power
      @then get that power
      THEN
      HEAD
      END
      @else multiply common
      LIST
      @powers
      END
      "Inserting new summand
      " 1 DISP
      SWAP OBJ-> ROT
      @Insert new summands in
      DROP 6 ROLL UNROT
      @series
      SWAP 1 + ->LIST ->ALG
      "Multiplying with
      @Multiply sums with
      common factors" 1 DISP
      @their extracted common
      comFact
      @factors
      CASE
      DUP { } SAME

```

# Sequences, series and limits with the HP49G - Part 2

```

      THEN
        1
      END
      DUP SIZE 1 ==
      THEN
        HEAD
      END
      LIST
    END *
  >>
  >> DOSUBS
  "Substituting new
series
" 1 DISP
  2
  << 2 ->LIST @Substitute old with new
  MATCH DROP @series
  >> DOLIST
  "Substituting
generating function
" 1 DISP @Put GF where the sums are
  { ' (&N=&L,&U,&F(&n)*X^&n)' GF }
  MATCH DROP
  "Solving for @Solve for GF
generating function
" 1 DISP
  GF = GF SOLVE
  >>
  ELSE @We don't deal with other sequences
  "Can't deal with this
kind of sequences"
  DOERR
  END
>>

```

The program will only deal with recurrences of the Fibonacci type. Of course interested readers will add their code for other types of recurrences.) We test the program giving it the Fibonacci sequence.

Enter  $\{P(n) = P(n-1) + P(n-2) \quad \{P(0) = 0 \quad P(1) = 1\}\}$  and press GENFUNC. The HP49G needs about 30 seconds to return

$$GF = -\frac{X}{X^2 + X - 1},$$

the generating function of the Fibonacci sequence.

Enter now  $F(n) = \frac{F(n-1)}{8} + F(n-2)$   $\{F(0) = 0 \quad F(1) = 2\}$  and

press GENFUNC again. In 36 seconds you see that the generating function of this series is  $GF = -\frac{16 X}{8 X^2 + X - 16}$ .

By the way, one of the example problems in the manuals of the TI92 (oh yes, I read them - how else could I say that the HP49G is better?) is of the type: We start with, say 100 trees. Each spring we cut 1/3 of the available trees and we plant another 50. How much trees we have after 10 years? The recurrence for this sequence is

$$T(n) = T(n-1) - \frac{T(n-1)}{3} + 50 \quad \{T(0) = 100\} \quad \text{which is the same}$$

with  $T(n) = \frac{2 T(n-1)}{3} + 50 \quad \{T(0) = 100\}$ . Our programs don't

handle such recurrences. But perhaps the following thoughts on the next page will wake your appetite for additional code ;-)

# Sequences, series and limits with the HP49G - Part 2

$$GF(X) = \sum_{n=0} T(n) X^n = 100 + \sum_{n=1} T(n) X^n =$$

$$100 + \sum_{n=1} \frac{2}{3} \frac{T(n-1)}{3} + 50 \sum_{n=1} X^n = 100 + \sum_{n=1} \frac{2}{3} \frac{T(n-1)}{3} X^n + 50 \sum_{n=1} X^n =$$

$$100 + \frac{2}{3} \sum_{n=1} T(n-1) X^n + 50 \sum_{n=1} X^n =$$

$$100 + \frac{2}{3} X \sum_{n=1} T(n-1) X^{n-1} + 50 \sum_{n=1} X^n =$$

$$100 + \frac{2}{3} X \sum_{n=0} T(n) X^n + 50 \sum_{n=1} X^n =$$

$$100 + \frac{2}{3} X GF(X) + 50 \sum_{n=0} X^n - 1 =$$

$$50 + \frac{2}{3} X GF(X) + 50 \sum_{n=0} X^n$$

Now, if we only new what to fo with the sum  $\sum_{n=0} X^n$  which isn't turned in some expression containing the generating function. But clever guys as we are, we notice that  $\sum_{n=0} X^n$  is the series expansion of

$\frac{1}{1-X}$  at  $X = 0$ . Can we put that in the above equations instead of

$\sum_{n=0} X^n$  ? Let's try that. If we replace  $\sum_{n=0} X^n$  with  $\frac{1}{1-X}$ , then we get  
 $GF(X) = 50 + \frac{2}{3} X GF(X) + 50 \frac{1}{1-X}$ . Solving for  $GF(X)$  gives

us  $GF(X) = -\frac{150 X - 300}{2 X^2 - 5 X + 3}$ . Expanding  $-\frac{150 X - 300}{2 X^2 - 5 X + 3}$  to a

series at  $X = 0$  returns the following series:

$$100 + \frac{350}{3} X + \frac{1150}{9} X^2 + \frac{3650}{27} X^3 + \frac{11350}{81} X^4 + \dots$$

If replacing  $\sum_{n=0} X^n$  with  $\frac{1}{1-X}$  was OK, then the coefficients must be members of the sequence.

We test our assumption. We enter the following small program:

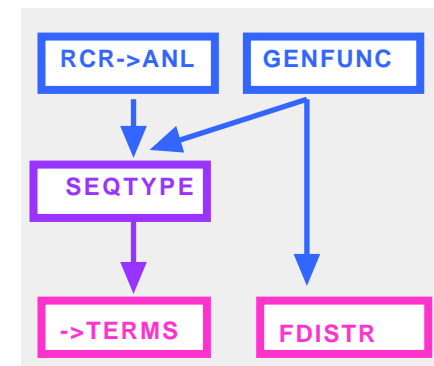
```
<< DUP
-> n
<< 2 n * 3 / 50 + EXPAND >>
>>
```

and store it in TISEQ. (Or something else if you think that the HP49G could be angry about that name and stop servicing ;-)) Now, we enter 100 (the starting number of trees). Pressing TISEQ (or whatever you called the program) leaves 100 and 350/3 on the stack. Another press and you have 100, 350/3 and 1150/9. It seems to be OK that we

replaced  $\sum_{n=0} X^n$  with its counterpart  $\frac{1}{1-X}$ . Can you add code to

GENFUNC, so that it returns the generating functions of recurrences of the type  $\{T(n) = a T(n-1) + b \mid T(0) = c\}$ , where a, b and c are constants (independent of the index variable n)? And would it be possible to let RCR->ANL also take care of this type of sequences?

Now that we are at last ready with the new programs let's take a look, how they depend on each other. We are going to put the programs of part 1 on this fundament. Those programs of part 1 haven't changed



# Sequences, series and limits with the HP49G - Part 2

significantly. There are a few minor changes and corrections but the whole remains the same. So we don't give their listings again here. The whole group of programs looks now like the picture at the right.

Let's test them with some sequences. We start with a sequence that is already in its analytic closed form:

Enter:

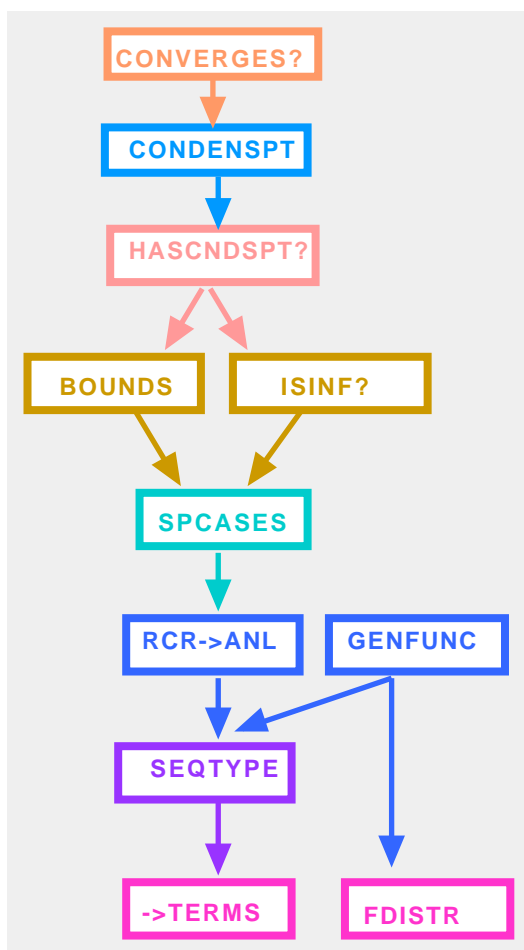
$$\frac{2}{n+1} \{n \ 0 \ \}$$

Store the sequence in some variable because we are going to use it more than once.

Now, recall it back to the stack and press SPCASES. In about 2.5 seconds the same sequence is returned, wrapped in a list, because it doesn't branch to other sequences.

Recall the sequence again and press ISINF?. In 4.5 seconds the HP49G returns a 1, and so it says, "Yes, the sequence is infinite".

Recall it again, press BOUNDS. In 5.5 seconds the HP49G returns the list  $\{0 \ 2\}$ , the bounds of the sequence, and a 1, to indicate that



bounds exist.

Recall it again. Press HASCNDSPT? It takes 7.5 seconds to see a 1 on the stack, which means that condensation points exist.

You know what I'm going to say, recall it again and press CONDENSPT. In 20 seconds you get the list  $\{0\}$  with the only condensation point of the sequence.

Last test with this sequence, recall it and press CONVERGES?. You wait again 20 seconds and you get a 1 for a converging sequence.

Now we test one more complicated sequence, but still in an analytic closed form. Enter  $(-1)^n \tan \frac{n}{3} \frac{n-3}{n^2} \{n \ 0 \}$  and test again with all programs of part 1. You find for example that the

bounds are  $\frac{-\sqrt{3}}{12} \ 2 \ \sqrt{3}$  and that the sequence has one

condensation point  $\{0\}$ . (That means, it converges.) Note that the programs need often about 5 minutes or more to complete, so be patient until they finish (or crash ;-))

The real fun is when we use recurrences. Enter the recurrence  $\{P(n) = P(n-1) + 2 \ P(n-2) \ \{P(0) = 0 \ P(1) = 1\}\}$ , store it in some variable and use the program RCR->ANL to turn it to the analytic closed form  $\frac{1}{3} \ 2^n + \frac{-1}{3} \ (-1)^n \ \{n \ 0 \ + \}$ , for which

is already much easier to say if it converges or if it has condensation points with only taking a look at it. Recall the recurrence and use GENFUNC to find that its generating function is  $-\frac{X}{2 \ X^2 + X - 1}$ .

Recall it again and use ISINF? to see that this is an infinite sequence. Unfortunately BOUNDS returns  $\{? \ ?\}$  and ? to denote that it can't find if the sequence is bounded. We can "see" easily that it isn't

## Sequences, series and limits with the HP49G - Part 2

bounded but the program can't "see" that. This has to do with the fact that TABVAR doesn't find that  $\frac{1}{3} 2^n + \frac{-1}{3} (-1)^n$  grows to infinity when  $n \rightarrow \infty$ . Here we have room for improvement (which will be implemented on the next part of this marathon ;-)).

And a last example with the other recurrence type. Store

$$P(n) = P(n-1) + \cos \frac{n}{3} \quad \{P(0) = 0\} \text{ to some variable. Then}$$

recall it and run RCR->ANL. The result is Nick.

$$\frac{2 \cos\left(\frac{(n-1)}{3}\right) - 1}{2} \quad \{n \geq 0\} . \text{ BOUNDS finds the}$$

bounds to be  $\frac{-3}{2} \frac{1}{2}$ . Check what the other programs say.

Of course the programs introduced here are not jewels of the programming art. They have their quirks, they are slow and they will crash sometimes. But it is nowhere intended here to make a second CAS. The marathons are only made to show how much possibilities hide inside this small machine. Especially when the numerous built-in commands are used in combination, one can achieve almost anything. (We used LINSOLVE and TABVAR for completely different reasons than those which these commands are often used for.) And another reason for the marathons to show that a CAS should better be a system of tools that enables us to program our tools, with which we program other tools, and so on. It shouldn't be a fixed system, that provides a lot of functions but makes it difficult to make new ones according to our needs.

It is late at night and my eyes approach their condensation point. (Sleep.) So let's end this part with the description of what comes in the next part. We are going to give the answer to the question of the example sequence in the manuals of the TI, we are going to examine the monotony behaviour of sequences and also take a closer look to some sequences that appear quite often. We are also going to take a

closer look to some particularly nice sequences and check what the HP49G provides to help us work with them. Then we are going to examine some of the important rules about sequences, like for example limits of sums, differences, products and quotients of sequences and the like.

Converging to sleeping state I send you the  $n^{\text{th}}$  member of the greetings-sequence, which I expect you to convert to its analytic closed form ;-)

# Sequences, series and limits with the HP49G - Part 3

Good morning everybody!

After a looong sleep of about one and a half weeks faaar faar away from home, I return to Mathlands fully recovered and with a fat cold that I caught who knows where. I tell you, my nose was one of those super strange manifolds that don't fit in any other space except their own! It is a very big advantage of Mathlands that you never catch a cold no matter how long you walk on the paths of the marathon adventures. (Eventually occurring psychic diseases are completely different problems ;-))

We had a special appearance of the TI in the last episode of this marathon, where we looked closer at the sequence

$$T(n) = T(n-1) - \frac{T(n-1)}{3} + 50 \quad \{T(0) = 100\} \quad \text{which the TI can}$$

plot out of the box, while the HP49G can't do much with it, even equipped with our programs. Of course this is unacceptable and so we are going to teach the HP49G to wipe this hubris out of the world. ;-)  
If we could expand RCR->ANL so that it returns an analytic closed form of the above sequence, then we would have done a first step in the right direction. So let's look at this sequence a bit closer again. We

can rewrite it as  $T(n) = \frac{2}{3} T(n-1) + 50 \quad \{T(0) = 100\}$ . Now we

see that it can be expressed more generally as  $\{T(n) = b T(n-1) + c \quad \{T(0) = a\}\}$ ,  $a$ ,  $b$  and  $c$  being arbitrary constants. We try to get the generating function of the beast:

$$GF(X) = \sum_{n=0} T(n) X^n = a + \sum_{n=1} T(n) X^n =$$

$$a + \sum_{n=1} (b T(n-1) + c) X^n = a + \sum_{n=1} b T(n-1) X^n + c \sum_{n=1} X^n =$$

$$a + b \sum_{n=1} T(n-1) X^n + c \sum_{n=1} X^n =$$

$$a + b \sum_{n=1} T(n-1) X^{n-1} + c \sum_{n=1} X^n =$$

$$a + b \sum_{n=0} T(n) X^n + c \sum_{n=1} X^n =$$

$$a + b \sum_{n=0} GF(X) + c \sum_{n=0} X^n - 1 =$$

$$a + b \sum_{n=0} GF(X) + c \sum_{n=0} X^n - c =$$

$$a - c + b \sum_{n=0} GF(X) + c \frac{1}{1-X}$$

Which means that  $GF(X) = a - c + b \sum_{n=0} GF(X) + c \frac{1}{1-X}$ .

SOLVEing this equation for  $GF(X)$  on the HP49G we obtain:

$$GF(X) = -\frac{(a-c) X - a}{b X^2 - (b+1) X + 1}$$

Now, let's expand the right hand side to a series about  $X = 0$  and try to "guess" the general dependence of the coefficients on  $a$ ,  $b$  and  $c$ .

Press EQ-> and NIP so that the expression  $-\frac{(a-c) X - a}{b X^2 - (b+1) X + 1}$

remains on stack level 1. Enter  $X=0$  and 5 and then press SERIES. After a few seconds the HP49G returns a list on stack level 2 and  $h = X$  on stack level 1. Press SUBST and then 3 GET to extract the series expansion from the list. Press DTAG to remove the label "Expans" from the series. Now you have on stack level 1:

$$(b^5 a + c b^4 + c b^3 + c b^2 + c b + c) X + \dots$$

$$+(b a + c) X + a$$



# Sequences, series and limits with the HP49G - Part 3

Now comes the difficult part of "guessing" how the coefficients of the powers  $X^n$  depend on  $a$ ,  $b$  and  $c$ . We observe that all coefficients are the sum of two expressions. The first is  $b^n a$ . The second is  $c \sum_{j=0}^{n-1} b^j$ . So each summand of the the series expansion can be written

as:  $b^n a + c \sum_{j=0}^{n-1} b^j X^n$ . Fortunately the HP49G can calculate the symbolic sum  $\sum_{j=0}^{n-1} b^j$ . If you enter  $\sum_{j=0}^{n-1} b^j$  and press EXPAND, the

HP49G answers:  $\frac{|b|^n - 1}{b - 1}$ . Now we now that the coefficients of the

powers  $X^n$  are  $b^n a + c \frac{|b|^n - 1}{b - 1}$ . And since these coefficients are themselves the members of the sequence, we have the analytic closed form  $T(n) = b^n a + c \frac{|b|^n - 1}{b - 1}$ . So RCR->ANL can be extended to

also handle with recurrences of the type  $\{T(n) = b T(n-1) + c \mid T(0) = a\}$ . I leave this as an exercise for the interested reader, saying only that the SEQTYPE has to recognise this type of the sequence and RCR->ANL has to convert it to its analytic closed form.

We move on now to another property of sequences, the monotony. A sequence is monotonically increasing iff any of its members is greater than its predecessor, which means that  $a_n > a_{n-1}$ . It is monotonic non decreasing if  $a_n \geq a_{n-1}$ . A sequence is monotonic decreasing iff any of its members is less than its predecessor, which means that  $a_n < a_{n-1}$ . It is only monotonic non increasing if  $a_n \leq a_{n-1}$ .

How could we make a program which tests the monotony of a given sequence? Well, generally speaking, it looks like such a program

should have to do with EXPANDING the inequalities like the above  $a_n > a_{n-1}$ ,  $a_n \geq a_{n-1}$  and so on. If the result is 1, then the inequality is true. If it is 0 then it is false. But we must do a bit of additional work.

First of all we must do the above for each special case that SPCASES returns. If we deal for example with the sequence  $\frac{1}{n} \{n \ 1\}$ ,

then SPCASES returns  $\frac{1}{n} \{n \ 1\}$ , which contains only one case, so the method of the above paragraph can be applied directly. But if we have to do with  $(-1)^n \frac{n+3}{2n} \{n \ 1\}$ , then SPCASES

will return  $-1 \frac{n+3}{2n} \{n \ 1\} \ 1 \frac{n+3}{2n} \{n \ 2\}$ ,

that is two cases of which one rises and the other falls. We could be inclined to do the following: Apply the method of the previous paragraph to each case and then we follow the schema:

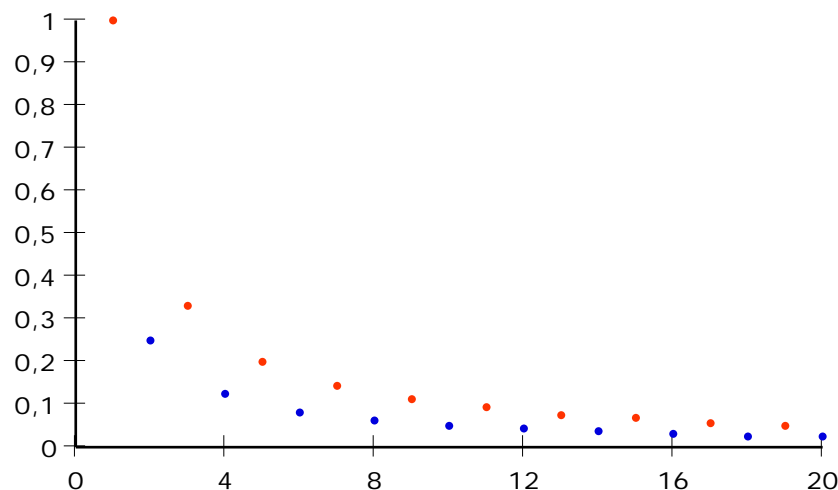
Method that doesn't always work	
If for each special case:	Then the sequence
$a_n > a_{n-1}$	Monotonic increasing
$a_n \geq a_{n-1}$	Monotonic non decreasing
$a_n \leq a_{n-1}$	Monotonic non increasing
$a_n < a_{n-1}$	Monotonic decreasing

But this will not always work. We can understand this better if we follow a simple hypothetical example. Let's suppose that we have a sequence that has the following recipe:  $a_n = \frac{1}{n}$  if  $n$  is odd, and

$a_n = \frac{1}{2n}$  if  $n$  is even, and with  $n = 1, 2, 3, \dots$ . The fact that we can't easily find an analytic closed form for this sequence is not important

# Sequences, series and limits with the HP49G - Part 3

here, as we only need the reason why the above algorithm wouldn't always work. Each special case,  $a_n = \frac{1}{n}$  for odd  $n$ , and  $a_n = \frac{1}{2n}$  for even  $n$ , is monotonic decreasing if taken alone for itself. But both special cases taken together form a new sequence which neither increases nor decreases. So we must think of a different method.



Notice how the differences of the special case behave and you're on the right way. We have  $a_2 - a_1 < 0$ ,  $a_3 - a_2 > 0$ ,  $a_4 - a_3 < 0$ , and so on. That means, instead of checking each special case for itself, we must check the difference of each pair of "adjacent" special cases. These differences are the quantities which must be all positive, if the sequence is non decreasing or negative if the sequence is non increasing.

Last but not least, we must make the appropriate assumptions because many (most?) of the inequalities that appear, can only be solved if the HP49G knows some assumption about the index variable. Consider

for example the simple sequence  $\frac{1}{n} \{n \ 1 \}$ . To let a program

decide if it increases or decreases, we must first check if  $\frac{1}{n} > \frac{1}{n+1}$  holds. Let's try that. Enter 'n' and use UNASSUME to remove any

assumptions about  $n$ . In the equation writer type  $\frac{1}{n} - \frac{1}{n+1} > 0$  and

press ENTER. Press now EXPAND. The result is the inequality itself, as the HP49G can't decide if it holds or not. And it does the right thing, because the inequality holds only for  $n < -1$  or  $n > 0$ . We didn't specify what  $n$  is, so the HP49G tells us that under such conditions no answer can be given. But now let's tell the confused machine that  $n$  starts at 1 and goes to infinity. Enter  $n \ 1$  and press ASSUME. DROP and EXPAND. Oops, nothing happens! The HP49G can still not answer. The HP49G seems to be only able to solve inequalities containing  $<$  or  $>$ , but not  $<=$  or  $>=$ . (And that means even more additional work, grrrrrr!!) But now, enter  $\frac{1}{n} - \frac{1}{n+1} = 0$  and press EXPAND. Aha! The HP49G says 1, which

means that the inequality holds. If you now enter 'n', press UNASSUME, DROP, then re-enter  $\frac{1}{n} - \frac{1}{n+1} = 0$  and then

EXPAND, then you get again the inequality as result. That means: It seems to be better to use the range of the index variable to make the appropriate assumptions, and then expand the inequalities  $a_{n+1} - a_n = 0$  and/or  $a_{n+1} - a_n < 0$ , but not  $a_{n+1} - a_n > 0$  or  $a_{n+1} - a_n < 0$ . But then we can only find if the sequence non increasing or non decreasing. How could we find if it increases or decreases monotonically? Well, one way to do that is to just solve  $a_{n+1} - a_n = 0$  for  $n$ . For example we have already seen that for the

sequence  $\frac{1}{n} \{n \ 1 \}$  the inequality holds:  $\frac{1}{n} - \frac{1}{n+1} = 0$ . If we

try to solve  $\frac{1}{n} - \frac{1}{n+1} = 0$  for  $n$ , then the HP49G returns an empty

# Sequences, series and limits with the HP49G - Part 3

list, as it wants to tell us that there is no solution. That means that the "equals" in  $\frac{1}{n} - \frac{1}{n+1} = 0$  is never true, and so we know that  $\frac{1}{n} - \frac{1}{n+1} > 0$  holds, which means that the sequence not only falls, but it also falls monotonically.

The following program implements these ideas. It takes a sequence as input and returns its monotony behaviour, coded as in the following table.

Sequence	Result
Monotonic increasing	2
Non decreasing for sure and perhaps monotonic increasing	:?:1
Non decreasing	1
Constant	0
Non increasing	-1
Non increasing for sure and perhaps monotonic decreasing	:?:-1
Monotonic decreasing	-2
Nothing of the above	3
Can't determine monotony behaviour	?

Note that the two last rows in the above table are different results. The last row tells us when the HP49G can't find what happens, for example because it can't solve some inequality. The row before the last tells us that the sequence is not ascending and not descending but

shows some other behaviour.

```
<<
  PUSH SPCASES                                @Find special cases
  "Determine differences                       @Add difference of last
of special cases" 1 DISP                      @special case minus first
  DUP HEAD OBJ-> DROP OBJ->                   @special case with the
  DROP UNROT 5 PICK SIZE R->I                 @appropriate indices.
  + ROT 3 ->LIST 2 ->LIST 1                   @(Why do we have to do
->LIST +                                       @ that?)
  1                                           @Do to each special case
<< OBJ-> DROP SWAP OVER                       @Substitute n=n in first
HEAD DUP NSUB 1 - R->I +                     @special case, n=n+1 in
= SUBST SWAP 2 ->LIST >>                   @second, and so on
DOSUBS
DUP HEAD 2 GET ->                           @Get index range of first
initcond                                     @special case, store local
<< 1
  << HEAD >>                                @Get each special case
  DOSUBS LIST                               @Difference of spc. cases
  EXPAND
  1                                           @Do to each difference
  << initcond 2 ->LIST                       @Convert it to spc. case
  >>                                         @with initial condition
  DOSUBS                                     @of the first spc. case
  >>
  1                                           @Do to each spc. case
  << OBJ-> DROP OBJ-> DROP
  -> seq var lo hi
  << var UNASSUME lo                         @Make assumption
  ASSUME DROP -103 CF
  "Checking An>=An+1                         @Expand inequalities
" 1 DISP seq 0 EXPAND
  "Checking An<=An+1
" 1 DISP seq 0 EXPAND
  "Checking An=An+1
" 1 DISP seq 0 = var
  IFERR
  ZEROS
  @Try to solve An=An+1
```

# Sequences, series and limits with the HP49G - Part 3

```

THEN
  DROP2 { NOVAL }
ELSE
  { } + @Make sure we have a list
END @of solutions
CASE
  "Checking unevaluated
inequalities" 1 DISP PICK3 @In case the HP49G didn't
  TYPE 9. == PICK3 @expand the inequalities
  TYPE 9. == AND
THEN
  3 DROPN 0 0 / @Return ?
END
  "Checking constant
" 1 DISP PICK3 PICK3 AND OVER @In case the solution
  0 0 / POS OR @of An=An+1 returned ?
THEN
  3 DROPN 0 @We have a "constant" seq.
END
  "Checking monotony
" 1 DISP DUP { } SAME @In case no solution of An=An+1
THEN
  IF @If An>An+1
    PICK3
    THEN
      3 DROPN -2 @Then it decreases
monotonically
    ELSE
      3 DROPN 2 @else it increases
monotonically
    END
  END
  DUP NOVAL POS @In case the HP49G couldn't
THEN @find if An=An+1 has solutions
  IF @If An>An+1 return 3
    PICK3
  THEN
    3 DROPN -3
  ELSE
    3 DROPN 3 @else return -3
  END
END
  DUP NOVAL POS NOT @Case the HP49G found
THEN @solutions of An=An+1
  IF @If An>An+1
    PICK3
    THEN
      3 DROPN -1 @Then non increasing
    ELSE
      3 DROPN 1 @Else non decreasing
    END
  END
END
var UNASSUME DROP @Remove assumptions
>>
>>
DOSUBS -> mlst @Store results locally
<<
CASE
  "Checking unsolved
inequalities" 1 DISP mlst @Any ? appeared?
  0 0 / POS
  THEN
    0 0 / @Return ?
  END
  "Checking monotonic
increasing" 1 DISP mlst 1 @All 2?
  << 2 == >> DOSUBS 1 +
  << AND >> STREAM @Increasing monotonically
  THEN
    2
  END
  "Checking eventually @All 3?
monotonic increasing"
  1 DISP mlst 1
  << 3 == >> DOSUBS 1 +

```

# Sequences, series and limits with the HP49G - Part 3

```

    << AND >> STREAM
  THEN
    @Incr. eventually monot.
    :: 1
  END
  "Checking non
  decreasing" 1 DISP mlst 1 @All 1?
    << 1 == >> DOSUBS 1 +
    << AND >> STREAM
  THEN
    1 @Non decreasing
  END
  "Checking constant
  " 1 DISP mlst 1
    << 0 == >> DOSUBS 1 + @All 0?
    << AND >> STREAM
  THEN
    0 @Then "constant"
  END
  "Checking non
  increasing" 1 DISP mlst 1
    << -1 == >> DOSUBS 1 + @All -1?
    << AND >> STREAM
  THEN
    -1 Then non increasing
  END
  "Checking eventually
  monotonic decreasing" 1 DISP mlst 1
    << -3 == >> DOSUBS 1 + @All -3?
    << AND >> STREAM
  THEN
    :: -1 @Decreasing eventually monot.
  END
  "Checking monotonic
  decreasing" 1 DISP mlst 1
    << -2 == >> DOSUBS 1 + @All -2?
    << AND >> STREAM
  THEN
    -2 @Then monot. decreasing

```

```

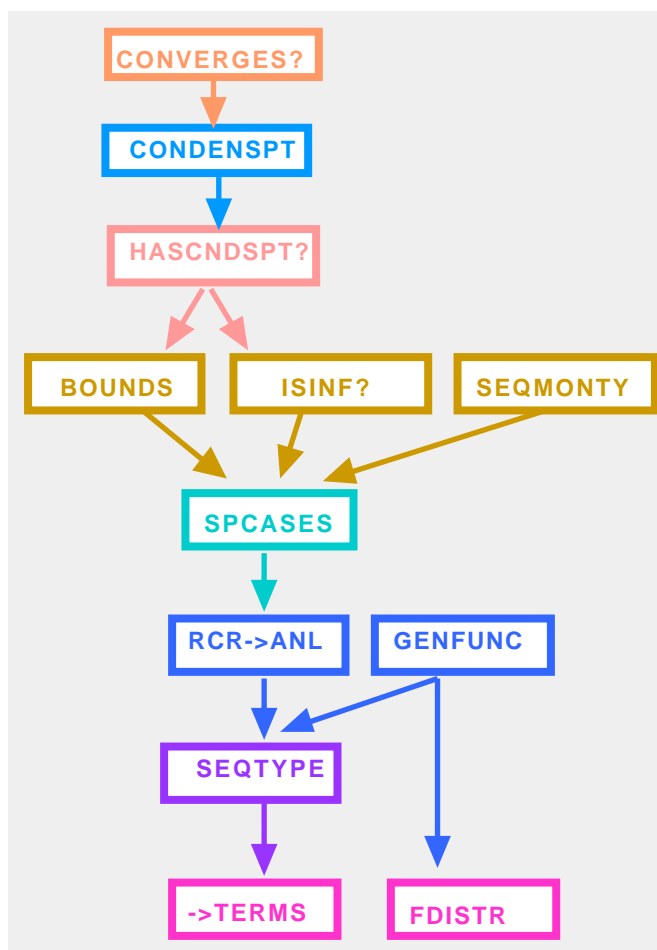
  END 3
  END
  >>
  POP
  >>

```

Store this in SEQMONTY (or use the program that comes along with this document). I think that there is not much to explain here, except that we use the fact that the HP49G will return a ? for an equations or inequality that has an infinite number of solutions, like for example  $n = n$ .

Before we test the program, let's update the building of programs for sequences. It starts looking like modern art, doesn't it? ;-)

Now let's test it. We start with something easy. Enter the sequence  $\frac{1}{n}$  {n 1 } and press SEQMONTY. In about 17 seconds you



# Sequences, series and limits with the HP49G - Part 3

get a -2 for a monotonic decreasing sequence.

Enter the sequence  $\frac{(-1)^n}{n}$  {n 1 } and press SEQMONTY. The

HP49G needs 61.5 seconds to return a 3, which means that this sequence is neither monotonic increasing or decreasing nor non increasing or non decreasing.

Enter  $\frac{3n+1}{2n+2}$  {n 0 } and press SEQMONTY again. In

about 20 seconds the program returns a 2 for a monotonic increasing sequence.

We move on to Fibonacci. Enter the recurrence sequence  $\{F(n) = F(n-1) + F(n-2) \quad \{F(0) = 0 \quad F(1) = 2\}\}$  and press

SEQMONTY. The HP49G goes all the way from the 5<sup>th</sup> floor of the building down to earth and up again while it flashes its messages to you in about 3 minutes (!) and it returns ? :1 to show you that this sequence is for sure non decreasing and perhaps also monotonic increasing. In fact it is only non decreasing, but the HP49G converted the recurrence to its analytic closed form:

$$F(n) = \frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^n + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^n$$

Then it tried to solve:

$$\frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^n + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^n =$$

$$\frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^{n+1} + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^{n+1}$$

It couldn't find some solution of the above equation for n and so it said: Master, I can't solve that but perhaps some solution exists, so I tell you that if such a solution exists, the sequence is non decreasing. But if none exists, then it is monotonic increasing. Hence the question mark in front of the returned 1.

Try other examples and don't forget to blame it on me when the program crashes ;-)

One of the sequences that occur quite often is the arithmetic sequence.

Its analytic closed form is (in our notation):  $\{a0 + n \cdot d \quad \{n \quad 0 \quad \}$

and its recurrence  $\{a(n) = a(n-1) + d \quad \{a(0) = a0\}\}$ . It's interesting to see if RCR->ANL can handle this. Enter the recurrence and press RCR->ANL to see that it does and it returns the sequence  $\{a0 + n \cdot d \quad \{n \quad 0 \quad \}$ . The constant d is the difference between each pair of adjacent members. This sequence is called arithmetic sequence, because any member of the sequence is the arithmetic mean

of its previous and its next member:  $a_n = \frac{a_{n-1} + a_{n+1}}{2}$ . This also

defines a recurrence. Solving for  $a_{n+1}$  we get  $a_{n+1} = 2 \cdot a_n - a_{n-1}$ . Can we feed RCR->ANL with this recurrence? Let's try. We must change index because RCR->ANL expects the member with the single index n without anything else on the left hand side of the recurrence equation and all members with smaller indices on the right hand side. So we enter  $\{a(n) = 2 \cdot a(n-1) - a(n-2) \quad \{a(0) = a0 \quad a(1) = a0 + d\}\}$ .

Pressing RCR->ANL returns  $\{-C2 \cdot 0^n + C2 \cdot 0^n \quad \{n \quad 0 \quad + \}\}$

which is... wrong! Panic spreads and Nick runs to save his life. But while running he is still thinking about the problem, such an psychomath he is ;-). Suddenly he changes direction and runs back shouting "I know why it failed!". And he proceeds with his explanations, which nonetheless can't excuse him.

The recurrence  $a(n) = 2 \cdot a(n-1) - a(n-2)$  is first transformed by the program to the characteristic polynomial  $X^2 - 2 \cdot X + 1$  which has the

# Sequences, series and limits with the HP49G - Part 3

solutions  $X = 1, X = 1$ . Then the general linear combination  $C1 \cdot 1^n + C2 \cdot 1^n$  is formed and the program tries to solve the linear system:

$$\begin{aligned} C1 \cdot 1^0 + C2 \cdot 1^0 &= a0 \\ C1 \cdot 1^1 + C2 \cdot 1^1 &= a0 + d \end{aligned}$$

for  $C1$  and  $C2$ , which of course fails! LINSOLVE returns in this case  $[C1 = -C2 \cdot 1 = 0]$ . The "solution"  $1 = 0$  shows that we run into troubles, which are caused by the two roots  $X = 1, X = 1$  of the polynomial.

The method of the characteristic polynomial doesn't work in this case. And since it is always better to return no results instead of the wrong results, we can just use the "impossible solution"  $1 = 0$  to detect that something went wrong. In general we have to just check if all solutions contained in the returned vector are "possible" or not. But before we do that we must examine what "equals" means for the HP49G.

First of all the HP49G has three "equals". The first is the normal  $=$  which we use in equations, and plays a different role according to how we manipulate the equation. If we solve some equation for a variable, then the sign  $=$  plays the role of a proposition. The commands of the HP49G that solve equations consider the sign  $=$  as a proposition. That means, they search to find for what values of the variable to solve for the equation holds. The opposite happens for equations used for SUBSTITUTIONS. SUBST consider the equation on stack level 1 to be an identity. That means, an equation which holds always.

Then we have the command  $==$ . This is an "equals" that resembles a boolean operation. It simply tries to evaluate left and right hand sides and check if both sides are equals. This command can return a 1 for "yes", a 0 for "no", or the symbolic expression  $leftHandSide == rightHandSide$  if no complete evaluation is possible. The command  $==$  can also return 1 if no evaluation is

necessary. For example using  $==$  with the two arguments  $X + Y$  and  $Y + X$ , will return equals even if no values are stored in variables  $X$  and  $Y$ .

The third "equals" is the command SAME. This is a "stronger equals" than  $==$ , because it checks if two given objects are completely identical, not in their mathematic sense, but in their "computer" sense. For example using SAME for  $X + Y$  and  $Y + X$  will return 0, because the two expressions are not identical for the HP49G. (The HP49G represents them using different bit patterns.)

The program RCR->ANL can be corrected now, to return  $\{? \{n \ 0 \ + \ \}\}$  instead of wrong results, when the vector returned by LINSOLVE contains "impossible" things. We simply replace each  $=$  in the vector of solutions to  $==$ , we expand, and if the result is 0, then we use it further. We replace it with a 1, if it is 1 or if it is of type 9 (symbolic expression), because then there will be (hopefully) some value for  $C1, C2$  etc. that satisfies the equation. The corrected code is:

```
.....
uc AXL LINSOLVE
"Checking solutions"
1 DISP DUP AXL 1
<< { '&A=&B' '&A==&B' } MATCH
DROP EXPAND
IF
  DUP TYPE 9 ==
  OVER 1 == OR
  THEN DROP 1
END
>>
DOSUBS
1 +
<< AND >> STREAM
IF
THEN
  "Substituting solutions"
```

# Sequences, series and limits with the HP49G - Part 3

```

1 DISP UNROT DROP2
AXL 1
<< SUBST >> DOSUBS
ELSE
4 DROPN 0 0 /
END
.....

```

The program that comes along with this document contains the corrected code, so you don't need to type anything. Perhaps you didn't even notice this problem, if you already have installed the programs at the start of this part.

Enter  $\{a(n) = 2 a(n-1) - a(n-2) \mid \{a(0) = a0 \mid a(1) = a0 + d\}\}$  and press RCR->ANL again. The result is  $\{? \mid \{n \ 0 \ + \ \}\}$ , which the HP49G uses to tell you that it can't solve this problem. If you enter  $\{F(n) = F(n-1) + F(n-2) \mid \{F(0) = 0 \mid F(1) = 2\}\}$  and use the corrected RCR->ANL again, then you get the result:

$$\frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^n + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^n \{n \ 0 \ + \}$$

which shows that the program still remembers what to do when the method of the characteristic polynomial works :-)

Well, if RCR->ANL can't help then perhaps the program GENFUNC can help us. We could find the generating function of the arithmetic sequence  $\{a(n) = 2 a(n-1) - a(n-2) \mid \{a(0) = a0 \mid a(1) = a0 + d\}\}$ , then expand it to a series around  $X = 0$  and then try to "guess" the analytic closed form of the coefficients. We enter the sequence  $\{a(n) = 2 a(n-1) - a(n-2) \mid \{a(0) = a0 \mid a(1) = a0 + d\}\}$  and press GENFUNC to get the result  $GF = \frac{(X+1) a0 + d X}{X^2 - 2 X + 1}$ . If you

expand the right hand side  $\frac{(X+1) a0 + d X}{X^2 - 2 X + 1}$  to a series around  $X = 0$  you get:

$$a0 + (3 a0 + d) X + (5 a0 + 2 d) X^2 + (7 a0 + 3 d) X^3 + \dots$$

which is... again wrong!!! Instead of this it should be  $a0 + (a0 + d) X + (a0 + 2 d) X^2 + (a0 + 3 d) X^3 + \dots$ . Again, Nick must run for his life ;-)

Let's follow what the program GENFUNC did but this time the mathematic way.

First it constructs the sum  $\sum_{n=0} a(n) X^n$ . Since we know that the initial conditions are  $a(0) = a0$  and  $a(1) = a0 + d$ , we can extract these summands from the sum and get  $a0 + (a0 + d) X + \sum_{n=2} a(n) X^n$ .

This is also carried out correctly by the program. Then the program plugs the recurrence definition  $a(n) = 2 a(n-1) - a(n-2)$  in the sum and yields  $a0 + (a0 + d) X + \sum_{n=2} (2 a(n-1) - a(n-2)) X^n$ . OK.

Then the program uses FDISTR to convert the sum and to yield  $a0 + (a0 + d) X + \sum_{n=2} 2 a(n-1) X^n - \sum_{n=2} a(n-2) X^n$  which is also carried out correctly. After this the program gets all sub expressions that are summations and returns the list

$$\sum_{n=2} 2 a(n-1) X^n \quad \sum_{n=2} a(n-2) X^n \quad \text{which it will use for}$$

extracting common factors and powers of  $X$ . By doing this the list is



# Sequences, series and limits with the HP49G - Part 3

converted to  $\sum_{n=2}^2 X^{a(n-1)} X^{n-1} X^2 \sum_{n=2}^2 a(n-2) X^{n-2}$

which is also OK. Then the new summations are substituted in the original expression giving us the result:

$$a_0 + (a_0 + d) X + \sum_{n=2}^2 X^{a(n-1)} X^{n-1} - X^2 \sum_{n=2}^2 a(n-2) X^{n-2}$$

which is also correct. But then the program is too naive. It simply and erroneously MATCHed any sum with the generating function GF, giving the result  $a_0 + (a_0 + d) X + \sum_{n=2}^2 X^{a(n-1)} X^{n-1} - X^2 GF - X^2 GF$ . From this

time on everything went wrong. The sum  $\sum_{n=2}^2 a(n-1) X^{n-1}$  can't be just replaced by GF because:

$$\sum_{n=2}^2 a(n-1) X^{n-1} = \sum_{n=1}^2 a(n) X^n = \sum_{n=0}^2 a(n) X^n - a_0 = GF - a_0$$

which means that it has to be replaced by  $GF - a_0$ . And that means that Nick has to type again. Grrrrrr!!

The corrected program first determines how the sum, like for example

$\sum_{n=2}^2 a(n-1) X^{n-1}$  can be converted to a sum in which the

coefficient and the power of X are only n and not n-1 or any other expression. It just "re-numbers" the coefficients and powers by putting a different starting value for the index n in the sum. So it constructs

$\sum_{n=1}^2 a(n) X^n$ . Then it checks to see what initial members must be subtracted from the sum if the sum has to be written as

$\sum_{n=0}^2 a(n) X^n$  and it finds that

$\sum_{n=1}^2 a(n) X^n = \sum_{n=0}^2 a(n) X^n - a_0$ . This sum is put back to the original expression and then the replacement by GF can be carried out because now we have the right sum. The second sum is treated the same way.

The old code was:

```
"Substituting new
series
" 1 DISP
  2                                @Substitute old with new
  << 2 ->LIST                      @series
      MATCH DROP
  >> DOLIST
  "Substituting
generating function
" 1 DISP                                @Put GF where the sums are
  { ' (&N=&L,&U,&F(&n)*X^&n)' GF }
      MATCH DROP
  "Solving for                      @Solve for GF
generating function
" 1 DISP
  GF = GF SOLVE
```

This code is replaced by:

```
DUP 4 ROLLD 2                      @Make a copy of new sums
<< 2 ->LIST MATCH DROP >>
DOLIST SWAP
"Series index change,
init. cond. insertion
" 1 DISP
  1
  <<
    { ' (&N=&L,&U,&S)*&F' @Get sum without factors
```

# Sequences, series and limits with the HP49G - Part 3

```

' (&N=&L,&U,&S)' }
MATCH DROP
DUPDUP
{ ' (&N=&L,&U,&F*&X^&O)' &O }
MATCH DROP SWAP OBJ->
DROP2
-> pow ind lol upl sum @Store power of X, index
<< ind 0 upl @ upper, lower sum.
sum pow ind 2 @sum indices and summand
->LIST
MATCH DROP
{ } + + + + \->ALG @Make sum with low=0
bcond 1
<< EQ-> SWAP
OBJ-> DROP2
IF @If init. cond. has index
pow @lower than start of sum
ind lol =
SUBST
EXPAND <
THEN - @then subtract
ELSE @else drop
DROP
END
>>
DOSUBS @DO to each init cond
2 ->LIST
>>
>> DOSUBS @Do to each sum
"Substituting new
series
" 1 DISP 1
<< MATCH DROP >> DOSUBS
"Substituting
generating function
" 1 DISP { ' (&N=0,&U, &F(&n)*X^&n)' GF }
MATCH DROP

```

"Solving for  
generating function  
" 1 DISP GF = GF SOLVE

Again the program that comes with this document is already corrected, so you might not see the above erroneous result.

We test the program again. Enter the arithmetic sequence  $\{a(n) = 2 \ a(n-1) - a(n-2) \ \{a(0) = a_0 \ a(1) = a_0 + d\}$  and

press GENFUNC to get  $GF = -\frac{(X-1) \ a_0 - d \ X}{X^2 - 2 \ X + 1}$ , which expanded

to a series around  $X = 0$  returns  $a_0 + (a_0 + d) \ X + (a_0 + 2 \ d) \ X^2 + \dots$ . The program still works with the recurrences it already handled correctly. For example you still get the generating function

$GF = -\frac{X}{X^2 + X - 1}$  for the Fibonacci recurrence.

So the simple arithmetic sequence with its two recurrence forms helped us find errors in the programs (and put Nick's life in danger ;-)). But with the help of the HP49G Nick survived and can continue the Marathon. I tell you, mathematics can be really dangerous ;-)

Now that the dangers seem to be far away, let's continue on the arithmetic sequence. If for a given arithmetic sequence we have three of the four variables of its analytic closed form  $a_n = a_0 + n \ d$ , then we can solve for the remaining variable. for example, if we know that  $a_n = 10$ ,  $a_0 = 0$  and  $d = 2$ , then we can solve for  $n$  and find

$n = \frac{a_n - a_0}{d} = \frac{10 - 0}{2} = 5$ . Similarly if we know that  $a_n = 13$ ,  $a_0 = 1$

and  $n = 2$  then we can solve for  $d$  and find  $d = \frac{a_n - a_0}{n} = \frac{13 - 1}{2} = 6$ .

We can make a program that takes  $a_n$ ,  $a_0$ ,  $n$ , and  $d$  from the stack and solves for the first unknown parameter:

# Sequences, series and limits with the HP49G - Part 3

```
<<
-> an a0 n d      @Store in local variable
'an=a0+n*d'      @Defining procedure
DUP LNAME        @Return vector of names
IF
  DUP TYPE 5 == @If no names present
THEN
  DROP2 COLLECT @Check if sequence is possible
  NOT EVAL
ELSE
  @Else solve for first variable
  1 GET SOLVE   @in names vector
  2 ->LIST
END
>>
```

Store this in SOLARSEQ (or just get the program that comes with this document). The program works the following way:

First, all arguments are stored in local variables. The defining procedure of these local variables is the part 'an=a0+n\*d'. These variables exist only within this algebraic object but not afterwards. You remember of course that one type of a local variables procedure definition is:

```
<<
-> arg1 arg2 ...    @Store in local variable
'algebraicObject'  @Defining procedure
>>
```

where the 'algebraicObject' contains arg1 and/or arg2 etc. This object is evaluated using the input values of the local variables. Now, there can be additional commands/functions *after* the algebraic object, but these can't use the input values of the local variables anymore, as the defining procedure has already been finished. We use this kind of local variables structure only to automatically evaluate the equation 'an=a0+n\*d' and work with the evaluated equation afterwards.

After this the program checks to see if there are names in the evaluated

equation. For example the user may have entered 10, 0, n and 2. In this case the program solves the equation '10=0+n\*2' for the variable n and returns the list {10 = n 2 n = 5} which contains the evaluated equation and its solution for n. If the program finds no names, then it uses the evaluated equation to check if the equation holds or not. For example. if the user enters 10, 0, 3, 2 then the program will return 0 because there the equation 'an=a0+n\*d' doesn't hold for the given values. But if the user enters 10, 0, 2, 5 then the program returns 1, because these values satisfy the equation 'an=a0+n\*d'. This is one way to solve equations with many variables automatically for that variable for which no numeric value has been provided, without using the built-in numeric solver.

We continue with the arithmetic sequence. One recurrence definition of this sequence was  $\{a(n) = a(n-1) + d \quad \{a(0) = a0\}\}$ . This already shows that the difference between adjacent points is constant and equal to d. We can also demonstrate this on the HP49G. We enter the analytic closed form  $a0 + n \cdot d$ , and then enter n, 0, 9, 1. Then we use the command SEQ to create the sequence  $\{a0 \ a0 + d \ a0 + 2 \cdot d \ \dots \ a0 + 9 \cdot d\}$ . Now we can use the command LIST to find a list containing the differences  $a_1 - a_0$ ,  $a_2 - a_1$ , ...,  $a_9 - a_8$ , that is the first differences of the elements in the list. If we EXPAND then we get  $\{d \ d \ d \ \dots \ d\}$ , which demonstrates (but doesn't prove at all) that all differences are the same.

We can generalise and also consider the differences of the differences, then the differences of the differences of the differences, and so on. We name such differences, first differences, second differences and so on. If we encounter some sequence whose  $n^{\text{th}}$  differences are all equal to each other, then we say that the sequence is an  $n^{\text{th}}$  order arithmetic sequence. Consider for example the sequence  $\{n^3 \ \{n \ 0 \ \dots\}\}$ . Its first few members and differences  $^1$ ,  $^2$  and  $^3$  are summarised in the table on the next page.

# Sequences, series and limits with the HP49G - Part 3

n	0	1	2	3	4	5	6
n <sup>3</sup>	0	1	8	27	64	125	216
1		1	7	19	37	61	91
2			6	12	18	24	30
3				6	6	6	6

We can see that the third differences are all the same, so the sequence  $\{n^3 \mid n \in \mathbb{Z}\}$  is a third order arithmetic sequence.

You can find the 3<sup>rd</sup> differences on the HP49G. Enter  $n^3$ ,  $n$ , 0, 10, 1. Then press SEQ. Now, if you press three times LIST, then you get a list containing only sixes, which demonstrates that  $\{n^3 \mid n \in \mathbb{Z}\}$  is an arithmetic sequence of third order.

At this point we make a small excursion to the future calculus marathon and we notice the resemblance of the behaviour of the differences and differences of differences etc. of  $n^3$  with the derivatives of  $n^3$  with respect to  $n$ . The third derivative of  $n^3$  is also a constant:  $\frac{d^3(n^3)}{dn^3} = 3 \frac{d^2(n^2)}{dn^2} = 3 \cdot 2 \frac{d(n)}{dn} = 6$ . The differences are the discrete analoga of the derivatives. End of excursion, back to the main path.

As you might have guessed, the next thing that we do is a program that finds the  $n^{\text{th}}$  differences of a given sequence at a certain point. For example we consider again the sequence  $\{n^3 \mid n \in \mathbb{Z}\}$ . Let's say that we want the 2<sup>nd</sup> difference. This is not enough information because as we can see on the following table, the second difference depends on the members that are subtracted from each other, or in other words the index  $n$ . We see that it is not so easy to say which difference belongs to a specific point. For example, which is the second difference of the point  $(n+1)^3$ ? To avoid confusion we can make the convention shown with arrows on the table, according to

which the point  $(n+1)^3$  corresponds to the 2<sup>nd</sup> difference  $6n+12$ .

n	n	n+1	n+2	n+3
n <sup>3</sup>	n <sup>3</sup>	(n+1) <sup>3</sup>	(n+2) <sup>3</sup>	(n+3) <sup>3</sup>
1		3n <sup>2</sup> +3n+1	3n <sup>2</sup> +9n+7	3n <sup>2</sup> +15n+19
2		6n+6	6n+12	

Now we can also specify the point of which we want to have the 2<sup>nd</sup> or any other difference.

The program in its first implementation will only try to find differences if the sequence is in its analytic closed form, or if it can be converted to such a form. If such a conversion is not possible, it will simply return ?? to denote that the differences can't be found because no analytic closed form can be found. The program also will check if the difference that we want to have doesn't depend on the index  $n$ , or in other words, if it remains the same no matter which point we specified. If this is the case, then it will return the result as =: , where is the found difference. The result will then be tagged with = to denote that the difference is the same everywhere. If the difference does depend on  $n$ , then the result will be returned as n: , where  $n$  is the specified point. And if Nick's is too stupid to explain what he means by all this, then don't worry, you'll understand as soon as the first examples will be calculated. ;-)

How are we going to prove, if some difference doesn't depend on the specified point  $n$ ? Good question as we can't sit and calculate if the difference remains the same for all possible values of  $n$ . But, there is something in mathematics that carries the name perfect induction. And this is what we will use here. It sounds like: Prove that something is valid for some  $n$ . If you then can prove that it also holds for  $n+1$  then you win because then it holds for any  $n$ . The question of course is then, how can we now for what  $n$  this "something" (namely, if the wanted difference doesn't depend on  $n$ ) holds? We need that  $n$ , or else we can't find  $n+1$ . The answer to the question is, do we really need a special value for  $n$ ? Perhaps it is better to give an example here.

# Sequences, series and limits with the HP49G - Part 3

We have already seen per demonstration that the third differences of  $\{n^3 \mid n \in \mathbb{N}\}$  are all 6. (It was no proof, but now we will prove it with the HP49G.) If we want to prove that with perfect induction, then we must prove first that for some arbitrary  $n$  the statement is true. We need at least four points because we want the 3<sup>rd</sup> differences. If we start at some arbitrary  $n$ , then the next 3 indices are  $n+1$ ,  $n+2$  and  $n+3$ . The corresponding members of  $\{n^3 \mid n \in \mathbb{N}\}$  are then  $n^3$ ,  $(n+1)^3$ ,  $(n+2)^3$  and  $(n+3)^3$ . Lets make a list with these points. Enter  $(n+m)^3$ ,  $m$ , 0, 3 and 1. Now press SEQ to get the list of points:

$\{n^3 \quad n^3+3 \quad n^2+3 \quad n+1 \quad n^3+6 \quad n^2+12 \quad n+8 \quad n^3+9 \quad n^2+27 \quad n+27\}$

Press LIST and then EXPAND to get the list of the first differences  $\{3 \quad n^2+3 \quad n+1 \quad 3 \quad n^2+9 \quad n+7 \quad 3 \quad n^2+15 \quad n+19\}$ . Again

LIST and EXPAND to get the list of the second differences  $\{6 \quad n+6 \quad 6 \quad n+12\}$ . Finally another LIST and EXPAND will give you a 6, which as you can see doesn't depend on  $n$ . Now we want to do the same for  $n+1$ . Following the same procedure we enter  $(n+1+m)^3$ ,  $m$ , 0, 3, 1 and press SEQ. Then doing LIST and EXPAND three times, we also get another 6, which shows that the result is the same for  $n+1$ . That means, the third difference of  $\{n^3 \mid n \in \mathbb{N}\}$  is constant for any other  $n$  and so the sequence is a third order arithmetic sequence.

Now, let's look at the listing of the program N SEQ which takes a sequence, the order of the difference and an index and does all the above.

```
<<
PUSH 1 CF ROT DUP
IFERR                      @If RCR->ANL errors out
RCR->ANL
```

```
THEN
  DROP 1 SF                @drop sequence, set flag 1
ELSE                        @else
  IF                        @if it returns a list with ?
    DUP ? POS
  THEN
    DROP 1 SF              @drop result, set flag 1
  ELSE
    NIP                    @else drop sequence
  END
END
-> ord pt seq              @Store in locals
<<
IF                          @If flag 1 is clear
  1 FC?
THEN                        @then we determine the nth
  "Determining " ord +
  ".
difference for n=m" +      @differences for n=m and n=m+1
  1 DISP seq OBJ->         @Substitute n=n+m and make a
  DROP OBJ-> DROP2         @sequence for m=start to
  UNROT DUP 'm' +          @m=order+1
  = SUBST 'm' ROT
  DUP ord + 1 SEQ
  1 OVER SIZE 1 -
  START                    @Calculate differences, expand
    LIST EXPAND            @use TEXPAND to expand trigs
    TEXPAND EXPAND
  NEXT
  HEAD DUP                 Get nth difference, make a copy
  "Determining "           Same again for n=m+1
  ord + ".
difference for n=m+1" +
  1 DISP seq OBJ->
  DROP OBJ-> DROP2 UNROT
  DUP 'm' + 1 + = SUBST
  'm' ROT DUP ord + 1
  SEQ 1 OVER SIZE 1 -
```

# Sequences, series and limits with the HP49G - Part 3

```

START
  LIST EXPAND TEXPAND
  EXPAND
NEXT
HEAD ==                @Both differences the same?
IF
  DUP TYPE 9 ==        @If a==b results in symbolic
THEN
  DROP 0                @drop it and return 0
END
IF                      @If test returned 1
THEN
  "=" ->TAG             @Tag with "="
ELSE
  @else
  DROP                  @we calculate nth diff.
  "Determining " @at specified point
  ord + ".
difference for n=" +
  pt + 1 DISP           @same method as above for
  seq OBJ-> DROP @n=pt to n=pt+ord
  OBJ-> 3 DROPN pt
  DUP ord + 1 SEQ 1 OVER SIZE 1 -
  START
    LIST EXPAND
    TEXPAND EXPAND
    NEXT
    HEAD pt ->STR ->TAG
  END
ELSE
  ? "?"                @If flag 1 was set return ??
  ->TAG
END
>> POP
>>

```

What's next after a program? Right, testing. Let's use the the sequence  $\{n^3 \{n \ 0 \} \}$ . Enter it and then enter 3 and 0 to find the third

difference at 0. Press N SEQ and wait about 23 seconds to get  $\approx 6$ . The program have found that this sequence is a third order arithmetic sequence. Its third differences are all the same and equal to 6. Now, enter  $\{n^3 \{n \ 0 \} \}$ , then a 3 and a 0 to find the second difference at 0. The program returns 0:6 after 18 seconds. That means that the second difference it 6 at the point 0, but at other points you will get other differences.

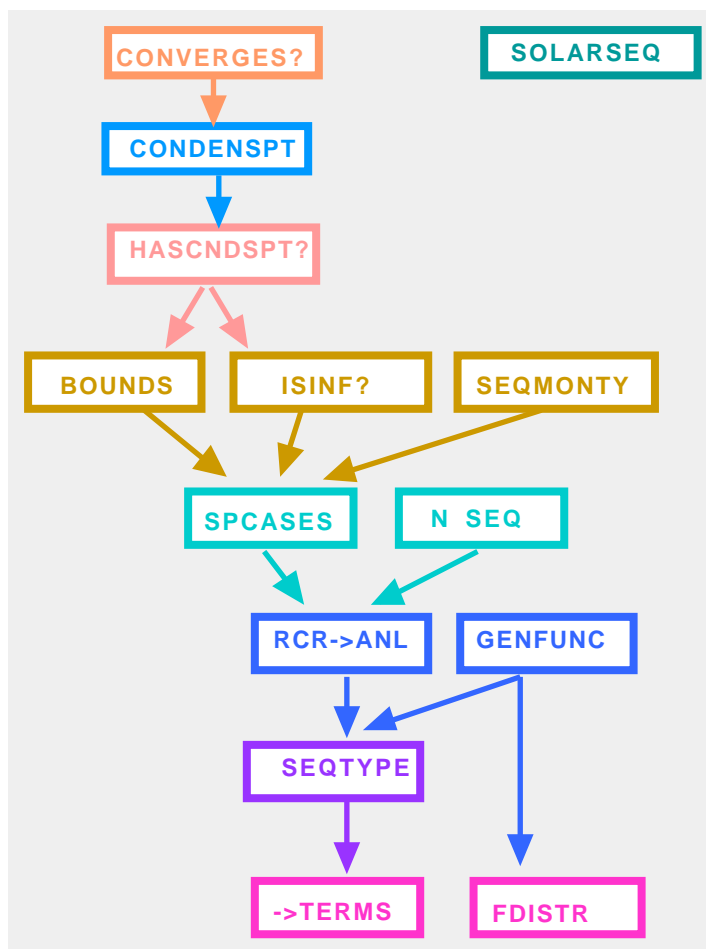
Try with Fibonacci. Enter  $\{F(n) = F(n-1) + F(n-2) \{F(0) = 0 \ F(1) = 2\} \}$ , a 1 and a 0 for the first difference at 0. Press N SEQ. The program uses RCR->ANL to turn this sequence to its analytic closed form and then proceeds to calculating. After about three minutes (!) you get 0:1.

Last example. Enter  $n^2 \text{ SIN } (4 \ n + 1) \frac{1}{2} \{n \ 0 \}$ , 2, 1 and press N SEQ. In 57 seconds the HP49G returns  $\approx (2 \text{ COS}(2 \ n))$ . Now, as Veli-Pekka says, if the HP49G offered some way to make assumptions about variables being integers, it would be possible to get the result  $\approx 0$ . But without that, you just stay at  $\text{COS}(2 \ n)$ . C'est la vie :-)

Our collection of programs has grown again. Take a look at it on the next page. N SEQ uses the program RCR->ANL. We also see that SOLARSEQ hovers alone over the whole building. (That's why it is a solar program. It is lonesome at the top ;)

Now that we have N SEQ, we think about having a program that finds if some given sequence is an arithmetic sequence of some order. That is, we give it a sequence and it tries to find out if there is some  $n^{\text{th}}$  difference that doesn't depend on the index of the sequence. This is a more difficult task. The program can of course build a limited number of differences, but it can't know how far to proceed. If we just build differences one after the other, then we may reach some point where all differences are the same. But we also may never reach such a point. So we can't proceed that way. We reconsider the sequence and its

# Sequences, series and limits with the HP49G - Part 3



differences in general. Suppose we have some sequence  $a_n$  and we derive its members  $a_0, a_1, a_2, a_3$ , and so on. Then we calculate the differences  $a_1 - a_0, a_2 - a_1, a_3 - a_2$ , and so on. Then we find the differences of the differences, that is the expressions

$$a_2 - a_1 - (a_1 - a_0) = a_2 - 2a_1 + a_0,$$

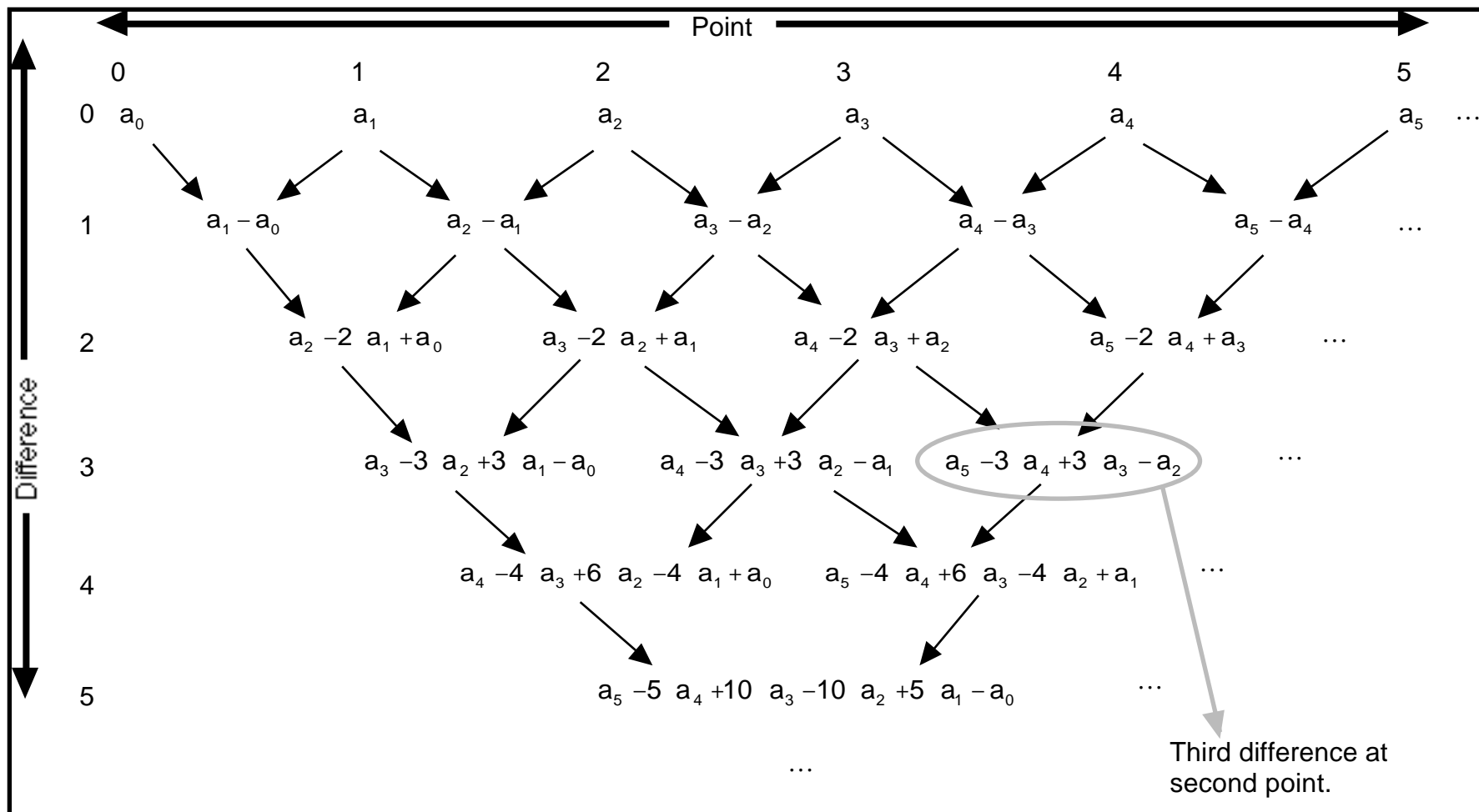
$$a_3 - a_2 - (a_2 - a_1) = a_3 - 2a_2 + a_1$$

and so on. We collect some of them on the table (next page), to get a better overview of them. Now, you of course can at once see how the coefficients of the members behave in the  $r^{\text{th}}$  difference at the  $n^{\text{th}}$  point. If you don't then it doesn't matter, as you presumably haven't seen a Pascal triangle yet. What is this Pascal triangle? That's very easy. It is shown on the next-next page. Each number on this triangle is the sum of the two numbers to the left and two the right above it. The numbers that are at the left or the right edge, are simply "the sum" of only one number above them. For example, if we start counting rows and members at 0, then the 2<sup>nd</sup> number at the 4<sup>th</sup> row is  $1+3=4$ . Of course it would be a tedious task to calculate the, say, 30<sup>th</sup> number in the 60<sup>th</sup> row but fortunately there is an analytic closed form for this. If we denote the row number with  $r$  and the position of the number in the row with  $n$  (starting at 0), then the corresponding Pascal's number is equal to  $\frac{r!}{n!(r-n)!}$ . Now you can see that these numbers appear

also in the differences of the previous table. For example, the third difference at the second point was  $a_5 - 3a_4 + 3a_3 - a_2$ . The coefficients of the members of the sequence in this difference are 1, -3, 3 and 1. They are the same like the Pascal's numbers 1, 3, 3, 1 at the third row of the triangle, except for the sign. That means that we can use the formula of Pascal's numbers for our general  $r^{\text{th}}$  difference. We can take care of the signs by just multiplying each Pascal's number with  $(-1)^n$ , where  $n$  is the position of the number in a row. The coefficient of the  $n^{\text{th}}$  sequence member in the  $r^{\text{th}}$  difference at the  $N^{\text{th}}$  point of the sequence is then  $(-1)^n \frac{r!}{n!(r-n)!}$ . The whole difference

is the sum of products of such coefficients with the corresponding sequence members. Now we must find which are these corresponding members. We try first an example for, say the third difference at the second point. The sum will have the general form

# Sequences, series and limits with the HP49G - Part 3

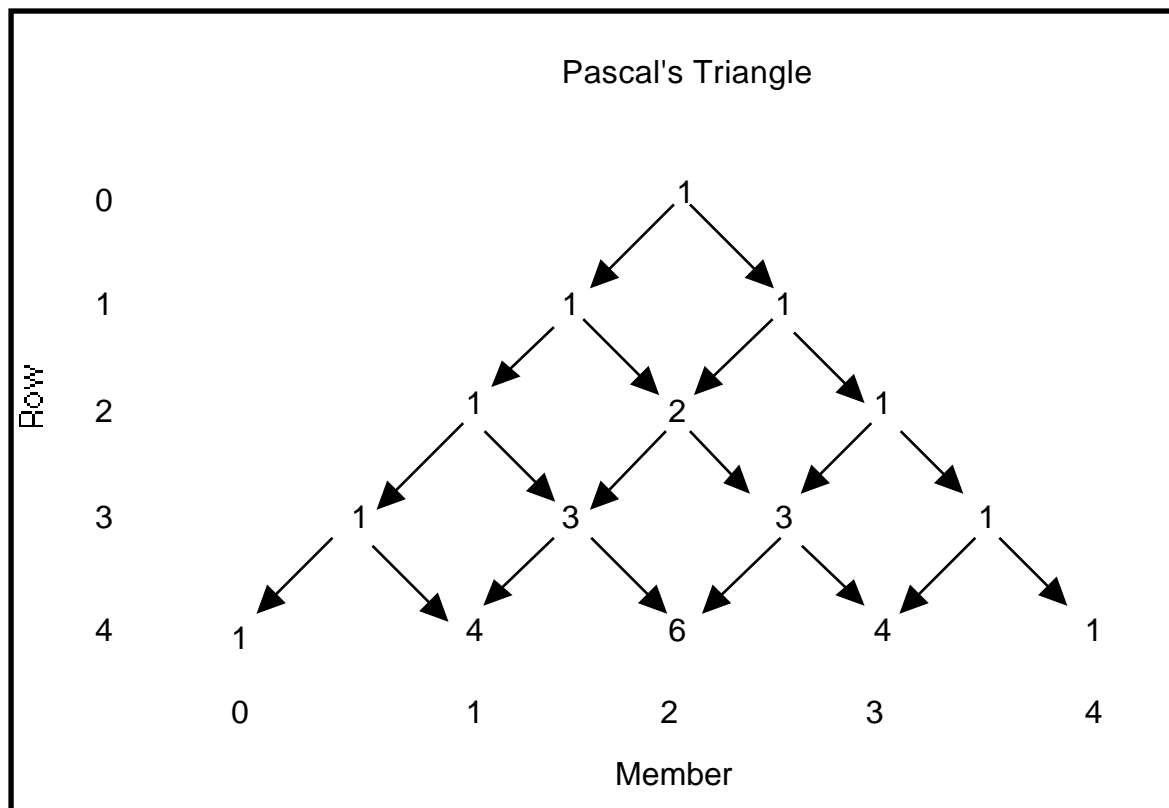


$\sum_{n=0}^r (-1)^n \frac{r!}{n! (r-n)!} a_n$ , in which we must plug the right quantities where the question marks appear. From the table of differences we see that this difference is  $a_5 - 3a_4 + 3a_3 - a_2$ . So, the index of  $a_n$  goes from 2 to 5. That is, the index it has to be  $r + N - n$ , where

$N = 2$ ,  $r = 3$  and  $n = 0, 1, 2, 3$ . The  $r^{\text{th}}$  difference at the  $N^{\text{th}}$  point is  $\sum_{n=0}^r (-1)^n \frac{r!}{n! (r-n)!} a_{r+N-n}$ . This is something that can help us to calculate any difference of any sequence at any point. Let's first define a function based on the above formula. Enter



# Sequences, series and limits with the HP49G - Part 3



$rN(r, N) = \sum_{n=0}^r (-1)^n \frac{r!}{n! (r-n)!} a(r+N-n)$  and press DEF. Now

let's find the 4<sup>th</sup> difference at the 1<sup>st</sup> point of the some sequence  $a(n)$ . Enter 4 and 1 and press  $rN$ . The HP49G needs over 11 seconds (!) to calculate the result  $a(5) - 4 a(4) + 6 a(3) - 4 a(2) + a(1)$ . First thing to complain about is the long time it needs. Second thing is that we supply integers and get results containing reals as indices. And third thing is that it switches from real to complex mode. Let's try to make the time needed a bit shorter. We enter 4. and 1. because we know that calculations with real numbers are faster in sums. If we now

use  $rN$  the HP49G still needs 10.5 seconds. Not much better. Enter '  $rN$ ' and press VISIT. Change the program to:

```
<<
-> r N
' (n=0.,r,(-1.)^n*r!/(n!*(r-
n)!)*a(r+N-n))'
>>
```

Notice the decimal point in  $n=0.$  and in  $(-1.)^n$ . Press ENTER. The new version is now stored in  $rN$ . Enter 4 and 1 and press  $rN$ . Now you get the same result in 3.8 seconds which is much better. Now we make the ugly result a bit better looking. Edit the program again and add XQ as the last command.

```
<<
-> r N
' (n=0.,r,(-1.)^n*r!/(n!*(r-
n)!)*a(r+N-n))' XQ
>>
```

Now the program needs about 10 seconds to give you the 4<sup>th</sup> difference at the 1<sup>st</sup> point of the sequence  $a(n)$ . Grrrr! What we gained through usage of reals we lose again through making the result looking better. As a last resource, enter 4. and 1. and try again. Fortunately now the HP49G needed only 4.2 seconds and returned the result  $a(5) - 4 a(4) + 6 a(3) - 4 a(2) + a(1)$ . Not a speed record, but better than 10 seconds.

Now that we have  $rN$  we can calculate the  $r^{\text{th}}$  difference of any sequence at the  $N^{\text{th}}$  point. Of course the program  $rN$  itself only

# Sequences, series and limits with the HP49G - Part 3

calculates such a difference for a generic sequence whose members are denoted by  $a(\text{index})$  where index is calculated by the function and can also contain the symbolic argument  $N$ . If we store some procedure for calculating the members in variable  $a$  then, when  $rN$  is evaluated the procedure in  $a$  will also be evaluated, letting  $rN$  calculate not the difference of the generic sequence, but the difference of the particular sequence  $a$ . For example, consider the sequence  $a_n = n^3$  with  $n = 0, 1, 2, \dots$ . A simple program to calculate some member of this sequence would be:

```
<<
-> 'n^3'
>>
```

If we store this in  $a$  then entering 2., 1. and pressing  $rN$  returns 12, the second difference at the first point. Entering 2., 2. and pressing  $rN$  returns 12, the second difference at the second point. But if we enter for example 2. and  $N$ , press  $rN$  and EXPAND, then we get  $6N + 6$ , the second difference at point  $N$ . The result clearly shows how the second difference behaves as a function of the point  $N$ . If we enter 3. and  $N$ , press  $rN$  and EXPAND, then the result is 6. It does no more depends on  $N$ , the point at which we take the differences. And so it is the same, constant, for all points. This shows us that the sequence  $a_n = n^3$  with  $n = 0, 1, 2, \dots$  is an arithmetic sequence of third order.

The next idea is to try to find a particular integer value for  $r$ , so that the sum  $\sum_{n=0}^r (-1)^n \frac{r!}{n! (r-n)!} a(r+N-n)$  is independent of  $N$  for some particular sequence stored in  $a$ . If we solve this, then this particular value of  $r$  is the order of the sequence and the sequence is an arithmetic sequence of order  $r$ .

To solve this we must make (again) an excursion at the land of the Summoids and their customs. This is a sometimes fascinating, sometimes confusing, but always interesting land. We are going to explore if/how the HP49G solves such problems. Suppose for

example that you want to solve  $\sum_{n=0}^N n = 10$  for  $N$ . That means you want to find how much  $n$  you must add, to receive the result 10. Can the

HP49G solve this? Let's experiment. Enter  $\sum_{n=0}^N n = 10$ , then  $N$  and the press SOLVE. Rattle, rattle and result:  $\{N = 4 \quad N = -5\}$ . The solution  $N = 4$  is plausible but  $N = -5$ ? How was this result obtained? Well, we must take a look at the workings of the CAS. The CAS has detected a symbolic sum with integer arguments. When such a thing is detected, the CAS first tries to solve the sum, to turn it to its analytic

closed form. This time it was possible to do so. If you enter  $\sum_{n=0}^N n$  and press EXPAND, then you get  $\frac{N^2 + N}{2}$  which is the same like  $\sum_{n=0}^N n$  for

any  $N$ . Then this expression is put in the equation and the CAS tries now to solve  $\frac{N^2 + N}{2} = 10$ , a quadratic equation in  $N$ , which has the above solutions. That's where  $N = -5$  comes from. But because we know that summing starts at  $n = 0$ , we know that  $N \geq 0$ . If we make

this assumption before we solve  $\sum_{n=0}^N n = 10$ , then we get only the

solution  $N = 4$ . This could be used to also solve our problem. But there are difficulties. First of all, let's define precisely what we would like to solve. We already said that we want to find such an  $r$  so that  $\sum_{n=0}^r (-1)^n \frac{r!}{n! (r-n)!} a(r+N-n)$  is independent of  $N$  for our particular sequence. We jump now from our excursion to the future marathon of calculus. When some expression is independent of

# Sequences, series and limits with the HP49G - Part 3

some variable, then mathematically that means that the derivative of expression with respect to that variable equals 0, or written in

math language:  $\frac{\text{expression}}{\text{variable}} = 0$ . Jump back to our excursion path.

For us, this means that we want to solve

$$\sum_{n=0}^r \frac{(-1)^n}{n!} \frac{r!}{(r-n)!} a(r+N-n) = 0 \text{ for } r. \text{ And here is where the}$$

problems start. First of all the sum itself is too difficult to be always solvable by the CAS of the HP49G. Remember that  $a(r+N-n)$  can be anything. In the above example with the sequence  $a_n = n^3$ ,  $n = 0, 1, 2, \dots$ , we would have to solve the equation

$$\sum_{n=0}^r \frac{(-1)^n}{n!} \frac{r!}{(r-n)!} (r+N-n)^3 = 0. \text{ Hopeless. Even Mathematica}$$

has problems to solve this analytically. We must find another way.

Well, if analytic solving doesn't work, what about numeric? Can we use numeric solvers for such problems at all? Again the experiment

will tell us more. We enter  $n = 10$ . (notice the decimal points for

faster numeric evaluation), then  $N$ , the variable to solve for, and then a guess value for the solution, say 0. Now we use the command ROOT, which is the programmable analogon of the numeric solver. Quite fast

the HP49G answers... 4.27272727302! Huh? Re-enter  $n = 10$ .

$N$ , but now enter for the guess value 1. and again ROOT. Solution:... 4.53474750039. Huh? What is wrong here? Why not just 4? And why different solutions according to what guess value we use? Well, the reasons are buried in the way the function works and in the way the command ROOT works.

The function will start evaluation the summand for  $n = \text{start}$  and

building the sum. For each next value of  $n$  it checks first if  $n > \text{End}$  and if not, it evaluates the summand for the next value of  $n$ . When  $n$  is greater than end, the sum is returned. For example, the above sum

with end value 4.27272727302 is  $n = 10$ . because the

function iterates  $n$  starting at 0 up to 4. The number 4.27272727302 is already greater than 4, so the function stops at

$n = 4$ . Same for  $n = 10$ . or any other ending value between 4 and 4.99999999999.

Now on to ROOT. This command takes the equation that we want to solve, variates (systematically) the value of the variable to solve for, and evaluates the equation until leftHandSide=rightHandSide or

until expression = 0. As it (systematically) variates  $N$  in our example, it doesn't know that it has to be integer, so it uses also real values. Eventually it reaches  $N = 4$ . something which evaluates the equation to  $10 = 10$  and so it returns the found solution. The exact value of the solution depends on where ROOT started searching, that means, our guess value. The value of the solution is also stored in  $N$ , or whatever the name of the variable is that we wanted to solve for. So, when we solve such equations we must take the integer part of the solution using the function IP. (Enter 'N' and PURGE to get rid of the stored value now, if you like. Also 'N' UNASSUME DROP to get rid of the assumptions for  $N$  that we already made. BTW, ROOT doesn't care about assumptions.)

It seems that ROOT could be used to solve our problem, but... there's always a "but", isn't there? ROOT can only find solutions for problems with only one unknown. If there are additional variables without values in an equation, ROOT will error out with "Undefined

# Sequences, series and limits with the HP49G - Part 3

Name". Our problem,  $\sum_{n=0}^r \frac{(-1)^n}{n!} \frac{r!}{(r-n)!} a(r+N-n) = 0$ , must

be solved for  $r$ , but it also contains  $N$  which must be symbolic if we want to solve it for *arbitrary*  $N$ . Only the solution for  $r$  will turn the left hand side to something that doesn't contain  $N$  but we must find that value first. Here comes a very very useful property of ROOT (and the numeric solver). ROOT can also solve... programs. Perhaps this sounds a bit strange now, but if we think a bit further then it seems to be a very natural thing. We consider a simple equation like  $3X - 2 = 0$ . What does it say? One way to interpret it is to think of it as a sequence of commands. We could understand it as: "Multiply  $X$  by 3, subtract 2 and check if the result equals 0." Now it sounds like a program, doesn't it? So instead of entering  $3X - 2 = 0$  when we want to solve this equation, we could just as well enter the program:

```
<<
  3 X * 2 -
>>
```

If you enter this, then you can enter  $X$ , a guess value 0 and use ROOT to find a solution. Don't forget that the solution is returned on the stack *and* stored in  $X$ , so you might want to PURGE variable  $X$  to avoid eventual complications with the CAS later.

For us this means that instead of solving the equation

$$\sum_{n=0}^r \frac{(-1)^n}{n!} \frac{r!}{(r-n)!} a(r+N-n) = 0, \text{ we can solve some program}$$

that does exactly the same. But the advantage is that in a program we can check for example if some name appears that has no value, so that in this case we return some value different from 0 which lets ROOT continue searching without erroring. But first things first. Before we make such a program, we need another program that takes a sequence from the stack and creates the user defined function  $a(r+N-n)$ , in

order to be able to construct the sum later. Because it is rather likely that the name  $a$  will be used by the user for storing something else, we change the name  $a$  to something more unlikely. Say, USDEFSEQ?

OK, first we must change this in  $rN$ . Edit this program to:

```
<<
-> r N
' (n=0..r, (-1.)^n*r!/(n!*(r-n)!)*USDEFSEQ(r+N-n)) '
>>
```

Now the program S->UD that makes a user defined program out of a sequence:

```
<<
  PUSH 1 CF DUP
  IFERR
    REC->ANL
  THEN
    DROP 1 SF
  ELSE
    IF
      DUP ? POS
    THEN
      DROP 1 SF
    ELSE
      NIP
    END
  END
END
IF
  1 FC?
THEN
  OBJ-> DROP OBJ->
  3 DROPN 1 ->LIST
  'USDEFSEQ' APPLY
  SWAP = DEFINE
ELSE
  POP "Can't find analytic
closed form" DOERR
END
```

# Sequences, series and limits with the HP49G - Part 3

```
END
POP
>>
```

Store this program in S->UD or use the program that comes with this document. Now enter the sequence  $\{n^3 \{n \ 0 \}\}$  and press the soft key for S->UD. The program generated a new program:

```
<<
-> n 'n^3'
>>
```

and stored it in USDEFSEQ. If you now enter 3 and press USDEFSEQ you get 27. If you enter n and press USDEFSEQ then you get  $n^3$ . You can use it for calculating arbitrary member of the sequence.

Now we can go on, trying to solve our problem using ROOT with a program. One way to solve this problem is to make a program that uses *r*, the order of the difference, the symbolic argument *N*, an arbitrary point of the sequence, and passes these arguments to *rN* to build up the *r*<sup>th</sup> difference at the *N*<sup>th</sup> point of the sequence. The program then has to take the derivative for *N*, expand and check if the resulting expression contains the variable *N*. If it does, then it can return something different than 0. to let ROOT know that the search for a solution for the variable *r* has to be continued. If the expanded derivative doesn't contain *N* anymore, then we have a solution.

```
<<
S->UD          @Create USDEFSEQ
<<            @Program to solve with ROOT
  r 'N'  rN     @Find difference, differentiate for N
  'N'  Ⓜ EXPAND @and expand
  LNAME NIP     @Return vector of variables
  IF           @If it isn't an empty list
    DUP {} ≠
```

```
THEN          @then transform it to a list
  AXL
END
IF            @If the list contains N
  'N' POS
THEN          @then return a 1.
  1.
ELSE          @else return a 0.
  0.
END
>>
'r' 1.        @Variable to solve for and guess value
ROOT
>>
```

Store that in ORDARSEQ. Let's give it a try. We happily enter the sequence  $\{n^3 \{n \ 0 \}\}$  and press the soft key for ORDARSEQ.

After some moments the HP49G returns... an error! The name 'N' is on the stack and the error message says "Undefined Name". What happened here? To understand that we must have a closer look to the way ROOT works. It is true that it will evaluate (run) a program repeatedly for different values of the variable to solve for, in our case variable *r*. It is also true that it looks at the result of the program to decide what to do next, which means that the program has to return a numeric value. The program that we supply (red part of the code) does return a numeric value in any case, provided someone plugs a numeric value for *r*, which ROOT does. Then why the error? There is an additional requirement for ROOT to do its job. *All names that somehow participate in order to calculate the result must contain numeric values!* While ROOT lets the red code run one step after the other, it eventually reaches 'N'. This thing is a name, ROOT says, and tries to replace it with a value. But as no value is associated with this name, ROOT ceases and shows an error message. So we have a problem. (As if there weren't already enough ;-))

So we must find a way to somehow hide 'N' from the sight of ROOT. Perhaps we can "construct" it? Instead of writing explicitly 'N' in the

# Sequences, series and limits with the HP49G - Part 3

program, we could use "N" OBJ->. The program then contains no name 'N'. We change ORDARSEQ and replace all occurrences of 'N' with "N" OBJ->. We enter  $\{n^3 \{n \ 0 \}\}$ , press ORDARSEQ and get the same error again. We can use "N" S~N or anything else that produces the name 'N'. The error remains the same. (Good to know that some things never change ;-)). To understand better what happens here, let's experiment a little bit.

## Experiment 1:

```
<< 'N' DROP R >> 'R' 1. ROOT
```

Enter this and you get the solution 0.

## Experiment 2:

```
<< 'N' 1 + DROP R >> 'R' 1. ROOT
```

The only change to the first experiment is that we add 1 to 'N'. Enter this and you get the error again.

That shows us: ROOT constantly checks anything (?) that has to do with mathematic operations, even simple additions, while it evaluates the program that we gave it to solve. It doesn't only parse the program to see if there are names that have no values. If some name somehow participates in math operations, then it is a suspect and it will be proved for numeric contents. ROOT does this work thoroughly but in this case it is just this thoroughly working that brings us troubles. Sometimes perfection isn't a good thing! Unfortunately!

So the excursion to ROOT was a one way path? Must get back and search some other way? As Trabakoulas says, "the hard thing in treks is not to reach the finish. The hard thing is to realise that one has to turn back to the start". But let's experiment a bit more before we search other ways. Our problem seems to be the involved mathematic operations like additions, exponentiations and so on. Now, we ask: Are there no mathematic operations at all, whose operands ROOT wouldn't check for numeric contents? A possible candidate group of operations that ROOT doesn't care if they return a numeric value or not, could be the group of operations that isn't made for the purpose of returning a numeric result. For example we consider our well known

EXPAND. This command isn't there for giving numeric results, though of course it will do so, if we give it an expression that contains only numbers. So let's try to use ROOT with a test program that contains EXPAND.

## Experiment 3:

```
<< '(N*(N-2)-N^2+2*N+3)*R-2' EXPAND >> 'R' 1. ROOT
```

The expanded algebraic expression is  $3R - 2$ , so it doesn't contain the variable N. If ROOT doesn't check EXPAND for numeric values, then it should find the solution .666666666667, which it does!

Hurrah! With new hope in our souls we still try to find a solution to our problem. What if we avoid anything that is checked by ROOT for returning numeric values? Is there any way to construct the  $r^{\text{th}}$  difference at the  $N^{\text{th}}$  point of the sequence *without any usage of such "checked" operations*? Let's take a look at the  $r^{\text{th}}$  difference again:

$$\sum_{n=0}^r (-1)^n \frac{r!}{n! (r-n)!} a(r+N-n)$$

We could use a loop instead of the function . And we could construct anything that is made by using additions exponentiations and so on, by using.... *strings!!!* Consider for example the result of

$$\sum_{n=0}^r \frac{r^n}{N} \quad \text{for } r=2 \text{ which is } 2^0 + 2^1 + 2^2.$$

The following program returns exactly the same result, but without using any mathematic operation with N as operand.

```
<<
" "
0 2
FOR I
2 I ^ @Math. operation ^ OK because I has a value
```

# Sequences, series and limits with the HP49G - Part 3

```
"*N+" + @Make string "2^I*N+"
+ @Add strings. No math operation!
NEXT
"0'" + @For appropriate closing.
OBJ-> @Transform to alg. object.
'N' @Derivation isn't checked for num result
EXPAND @Neither is checked EXPAND
>>
```

If you run this, you get the same result like with  $\frac{r^n}{N}$ , but without using any command that causes ROOT to say that some name is undefined (without numeric value). We adopt this method for our headache problem and write a program for solving the equation:

$$\sum_{n=0}^r \frac{(-1)^n}{N} \frac{r!}{n! (r-n)!} a(r+N-n) = 0$$

It takes a sequence and a guess value of  $r$  and tries to find a solution. If our sequence is an arithmetic sequence of higher order, then it will return the order  $r$ .

```
<<
PUSH 1 CF
SWAP DUP
IFERR @If seq. can't be made analytic
RCR->ANL
THEN
DROP 1 SF @Drop and set flag 1
ELSE @Else
IF @If result contains ?
DUP ? POS
THEN
DROP 1 SF @drop is, set flag 1
ELSE
NIP @else drop orig. sequence
END
```

```
END
IF @If flag 1 is clear (everything is OK)
1 FC?
THEN @then proceed
OBJ-> DROP OBJ-> 3 DROPN
-> guess seq indx @Locals for guess, seq., index
<<
<< @Here starts the program that ROOT
"" @will solve numerically.
0 r 0 RND @Loop start and end.
FOR I
-1 I ^ @Math. operations OK because
r 0 RND ! @all variables have values and
* I ! @r is given values by ROOT.
r 0 RND I - !
* /
*"(" + @Now we start making the string
seq indx
"" r 0 RND + @String version of r+N-n
"+N-" + @Addition with N would
I + "" + @cause troubles.
OBJ-> @String to algebraic.
2 ->LIST @Match each appearance of n in
↑MATCH @the sequence with r+N-n
DROP EXPAND @Expand the sequence member
->STR @Turn it to a string
2 OVER SIZE @Get rid of the quotes
1 - SUB
")" + + +
"+" +
NEXT
"0'" + @Correct closing of alg.
OBJ-> EXPAND @Transform to alg., expand
->STR "N" "N" @Transform expanded alg. to
SREPL NIP @string and check how many N's
r 1 DISP @Show us current value of r
DUP 2 DISP @and value of expression
>>
```

# Sequences, series and limits with the HP49G - Part 3

```
'r' guess ROOT IP @Take integer part of solution
>>
ELSE @In case of error show a message
POP "Can't find analytic
closed form" DOERR
END
POP
>>
```

The red code in the above program is the program that is passed to ROOT. Notice the usage of 0 RND to round the current value of  $r$  to the next integer. We do that because ROOT puts real values in  $r$ . Also notice that we don't even use the function `DERIV` for finding the derivative with respect to  $N$ . Instead of this we search the string for the character  $N$  and use the number of occurrences as a measure of how far away ROOT is from a solution. *That means that your sequence shouldn't contain any  $N$ , not only as a variable but also as character in any variable name. The same for character  $r$ .* Last thing to notice: The built in numeric solvers are able to show the current values of the variable to solve for if you press a key during the solution process is running. ROOT doesn't have this ability. But still, we can DISPLAY the values of the variable to solve for and anything else, as ROOT repeatedly evaluates the program for different values of the variable to solve for. The code `r 1 DISP DUP 2 DISP` in the program that is passed to ROOT, does exactly this. It DISPLAYs the current value of  $r$  on the first line of the display. Then it makes a copy of the result of each evaluation and DISPLAYs it on the second line, so that we can follow the process and eventually interrupt it if it starts going to the false direction.

Store the program in ORDARSEQ. ( Or as always use the program that comes along with this document. ) Let's test this unusual thing. Enter the sequence  $\{n^3 \{n \ 0 \ \}\}$  and a guess value 1. Press ORDARSEQ. In 53 seconds the result 3. is returned. The sequence was an arithmetic sequence of third order. Another example. Enter  $\{n^4 + 3 \ n^2 \ \{n \ 0 \ \}\}$  and 1. Press ORDARSEQ. The HP49G

rattles for seconds and after solving that rather non-standard procedure with string, it proudly returns 4. As already mentioned, ROOT also stores the found solution in the variable to solve for, so you have a new variable in the directory where you worked. If don't want to have it, then do 'r' PURGE, or add 'r' PURGE at the end of the program, just before the very last POP command.

The other two programs that we already make, S->UD and `rN` aren't used by the new ORDARSEQ. Nonetheless you can keep them as they are useful themselves.

The first moral of the story: Unusual problems (might) need unusual solutions. What we consider useless at the first sight, might prove to be exactly what we need to solve twisted problems in twisted ways, when the standard ways don't work. (Anyway, what is "standard"?)

The second moral of the story: Don't be afraid to go twisted ways. Mathematics doesn't seem to work always as we might "expect". Don't let your mind be wrapped in "normality" and don't forget to ask what might seem nonsense. The description of the numeric solution of

$$\sum_{n=0}^r (-1)^n \frac{r!}{n! (r-n)!} a(r+N-n) = 0 \text{ using ROOT and strings is}$$

way not the only possibility. Search more twisted paths! A whole world waits to be discovered. It's math. Let's get crazy! (As if we weren't already ;-))

The obligatory look to the structure of programs that we have so far (next page). RCR->ANL gets more important every minute.

We move on to another sequence that appears quite often in mathematics but also in "real" life, the geometric sequence. The ratio of two adjacent members of the geometric sequence is constant. The analytic closed definition of this sequence is:

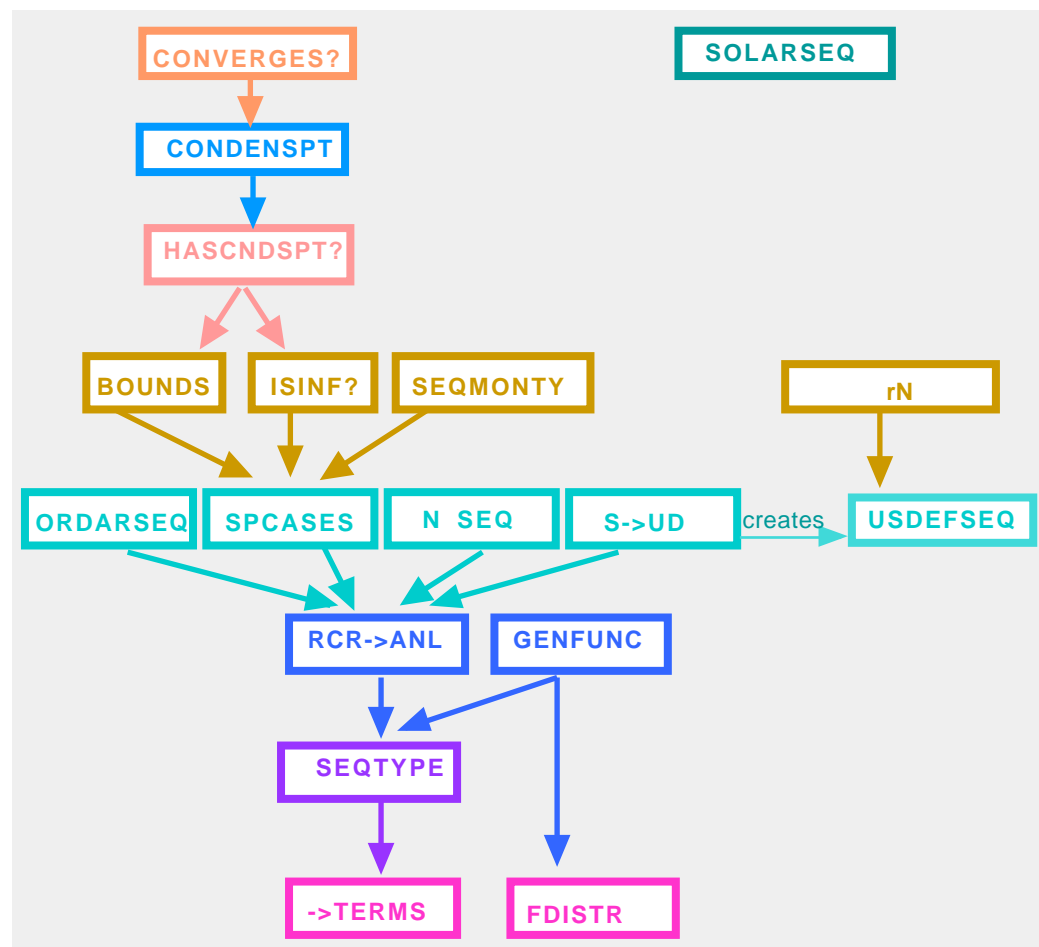
$$a_n = a_0 \cdot q^n, \text{ with } n = 0, 1, 2, \dots \text{ and } q = \text{constant}.$$



# Sequences, series and limits with the HP49G - Part 3

Some properties of this sequence are easy to see right from the start. For example, if  $|q| > 1$  then the sequence is monotonically increasing or decreasing, depending on the sign of  $q$ . Let's see if SEQMONTY can figure this out. We do good if we first make the right assumptions. Since SEQMONTY does this for the index  $n$ , we only need to make the assumptions for  $q$ . The assumption  $|q| > 1$  can't be made directly. If you enter  $|q| > 1$  and then ASSUME, the HP49G says "Bad Argument Value", which shows that the type of argument is OK, but the particular argument can't be used. We have to make this assumption in steps. First enter 'q' and UNASSUME to remove any existing assumptions for  $q$ . Then enter  $q = 1$  and ASSUME. After that enter  $q = -1$  and ASSUME again. (If you take a look at the variable REALASSUME HOME/CASDIR you will see that it contains  $q = -1$  AND  $q = 1$ . Though this is logically impossible<sup>4</sup>, let's try to use it. Enter the geometric sequence  $\{a_0 q^n \mid n \geq 0\}$ . Notice that we write  $a(0)$  as  $a_0$ , a symbolic constant. We don't write  $a(0)$  to be compatible with our conventions of notion of the analytic closed form of sequences. Trabakoulas heard the word "compatible" spoken by a Greek and smiles ;-). Because  $a_0$  is also symbolic, we must assume something about it to. Let's suppose first that  $a_0 = 0$ . Remove any existing assumptions for  $a_0$  and then ASSUME  $a_0 = 0$ . Press SEQMONTY. In 35 seconds the HP49G returns  $-2$  for a monotonically decreasing sequence. Which clearly shows that only the part  $q = 1$  of the existing assumption  $q = 1$  AND  $q = -1$  was used. That means, if we want to check the monotony of this sequence, we must check the particular cases alone. Remove the assumptions for  $q$ , and ASSUME  $q = 1$ .

<sup>4</sup> It should be  $q = -1$  OR  $q = 1$  for logical validity.



Enter  $\{a_0 q^n \mid n \geq 0\}$  and press SEQMONTY again. Now the result is  $2$  for a monotonically increasing sequence. Try other combinations of assumptions, like for example  $a_0 = 0$  ASSUME and  $q = 0$  ASSUME, and so on.

# Sequences, series and limits with the HP49G - Part 3

Of course if you have a geometric sequence with known values of  $a_0$  and  $q$ , then you don't need to make any assumptions. Remove the

assumptions for  $a_0$  and  $q$ , enter  $\frac{1}{2} 2^n \{n \ 0 \}$  and press

SEQMONTY. The result 2 shows that the program correctly found a monotonically increasing sequence.

If  $|q| < 1$ , then the geometric sequence is bounded. What does BOUNDS say in the general case of the sequence? Let's see. Enter  $q \ 1$ , ASSUME,  $q \ -1$ , ASSUME. Now REALASSUME contains the assumption  $q \ 1$  AND  $q \ -1$ , which is logically OK. (There are values of  $q$  that simultaneously satisfy  $q \ 1$  and  $q \ -1$ .) Now, enter the general form of the sequence  $\{a_0 \ q^n \{n \ 0 \}\}$  and press BOUNDS. The HP49G returns  $\{? \ ?\}$  and  $?$ , an answer that means "I don't know". This result comes because we used TABVAR in the program, a command that only works with monovariate functions, while  $a_0 \ q^n$  contains three variables. Remove the

assumptions for  $q$ , enter  $3 \ \frac{1}{2}^n \{n \ 0 \}$  and press

BOUNDS. The HP49G returns again  $\{? \ ?\}$  and  $?$ . The reason for the failure is that TABVAR can't find the variation table of  $3 \ \frac{1}{2}^x$ .

If you enter  $3 \ \frac{1}{2}^x$  and then press TABVAR, then the HP49G errors "Parameters not allowed" Go figure out where it sees parameters here! (The programs above BOUNDS rely on what BOUNDS finds, so they will also return question marks.)

The recurrence form of the geometric sequence is  $a_n = a_{n-1} \ q$ , with  $a_0 = a_0$  and  $n = 0, 1, 2, \dots$ . That means, in our notation we can enter

the geometric sequence as  $\{a(n) = a(n-1) \ q \ \{a(0) = a_0\}\}$ . Will our programs find the analytic closed form out of the recurrence? Let's see. Enter  $\{a(n) = a(n-1) \ q \ \{a(0) = a_0\}\}$  and press RCR->ANL.

The result is  $\{a_0 \ q^n \{n \ 0 \}\}$ . Wow, it worked! Now, let's find the generating function of the geometric sequence. We enter  $\{a(n) = a(n-1) \ q \ \{a(0) = a_0\}\}$  and press GENFUNC. The

HP49G returns  $GF = -\frac{a_0}{q \ X - 1}$ . Press EQ-> and NIP to get rid of

the left hand side. Enter  $X = 0$  and  $5$ . Press SERIES and then SUBST,  $3$  GET, DTAG. The result of these operations is  $q^5 a_0 X^5 + q^4 a_0 X^4 + \dots + q a_0 X + a_0$ , which shows that

$-\frac{a_0}{q \ X - 1}$  was correct.

As you might already imagine, the geometric sequence is called so because each member is the geometric mean of its neighbours:

$a_n = \sqrt{a_{n+1} \ a_{n-1}}$ . This can be also used to make another recurrence definition of the geometric sequence. If we solve for  $a_{n+1}$ , we find:

$a_{n+1} = \frac{a_n^2}{a_{n-1}}$ , or by defining  $m = n + 1$ :  $a_m = \frac{a_{m-1}^2}{a_{m-2}}$ . So the geometric sequence can also be written as the recurrence

$a(n) = \frac{a(n-1)^2}{a(n-2)} \ \{a(0) = a_0 \ a(1) = a_0 \ q\}$ . Notice that we now

have two initial conditions, which is the same like number of the different sequence members on the right hand side of the recurrence

$a(n) = \frac{a(n-1)^2}{a(n-2)}$ . How do our programs RCR->ANL and GENFUNC

react when they take as argument this recurrence? Again, we test. Enter

# Sequences, series and limits with the HP49G - Part 3

$$a(n) = \frac{a(n-1)^2}{a(n-2)} \quad \{a(0) = a0 \quad a(1) = a0 \quad q\} \quad \text{and press}$$

GENFUNC. After some seconds the HP49G strikes. "Not reducible to a rational expression"<sup>5</sup>. Something went wrong here. And it was the program SEQTYPE that determines the type of the sequence. Let's take a look at the happenings. The program finds out that this is a

recurrence. It uses  $\rightarrow$ TERMS to find the terms of  $a(n) = \frac{a(n-1)^2}{a(n-2)}$ .

The program  $\rightarrow$ TERMS correctly returns  $a(n) = \frac{a(n-1)^2}{a(n-2)}$ . Then

the SEQTYPE checks if the recurrence is of the type  $a(n) = c1 \cdot a(k) + c2 \cdot a(l)$ . It does this by matching the terms with  $a(&N)$  and at this point we already have the problem. Both terms do

contain patterns of the form  $a(&N)$ , but  $\frac{a(n-1)^2}{a(n-2)}$  contains members

of the sequence but in non-linear manner, and these case are not covered by GENFUNC. We must make additional checks to prove if the members are contained linearly in the terms. We immediately think

of LININ, but... again a "but". If you enter  $\frac{a(n-1)^2}{a(n-2)}$  and then for

example  $a(n-1)$  and press LININ, then you get the error "Bad argument type". We must go another way. We use the fact that when these checks take place, the HP49G has already found the terms of the sequence and so we know that the expressions that represent the terms are not themselves terms. That means that if more than one members of the sequence are contained in a term, they will be operands of some non-linear operation. So we have only to check if more than one members appear in a term. In the language of string, we must check if

the substring "a(" appears more than once. Of course we can't simply write "a(" because we can't be sure that the sequence definition will always name the sequence member  $a(n)$ . Perhaps the sequence is named Karagiaouoglou(n) or even worse things. (Is there anything worse than that? ;-)) So we must construct the string out of the data of the recurrence definition. We add the (red) code:

.....

```
"Recurrence 1"
2 DISP
terms 1          @Match any occurrence of
<< seqpatt 1      @type P(&N) with a 1. If
    2  $\rightarrow$ LIST    @we had a match, then any
        MATCH 'mtch' @term of the form
        SWAP STO+     @factor*P(&N) will be
        EXPAND        @matched to factor*1
>> DOSUBS
mtch              @Stream AND over the
<< AND            @matches flags
>> STREAM
SWAP DUP          @Store matched factors in
'factors'         @local 'factors'
STO
1                 @Find the positions of
<< LNAME NIP      @all terms which do not
    IF DUP        @contain the seq. index
    TYPE 5
    THEN AXL END var POS NOT
>> DOSUBS
1 +               @Add a 1 in the list of
<< AND            @positions and stream
>> STREAM        @AND
AND
terms 1           @Find num. of occurrences
<<               @of pattern "a(" in terms
     $\rightarrow$ STR seqpatt
    seqpatt 2 3
    SUB DUP SREPL
```

<sup>5</sup> The errors described here are those which are generated by the old version of the programs. If you already have installed the new ones, you won't get these errors. But we still analyse the code of the old programs for demonstrating how the errors are produced.

# Sequences, series and limits with the HP49G - Part 3

```

NIP 1 ⚡
>> DOSUBS 1 +
<< AND >> STREAM
AND          @If we had only P(&N) and

```

.....

Now SEQTYPE successfully finds that this is not a type 1 recurrence, but it erroneously finds that it is a type 2 recurrence. So we must add some checking again.

```

index
<< LNAME NIP      @factors of P(&N) not
IF DUP           @containing the seq.

TYPE 5           @or other factors
THEN AXL
END
IF SWAP
THEN var POS NOT
ELSE DROP 1
END
>> DOLIST
1 +              @Add 1 to the result list
<< AND           @stream AND over the result

list
>> STREAM
mtch LIST        @Check if only two terms
2 == AND         @match pattern P(&N)
factors mtch
2                @Get the factors of
<<              @P(&N)
IF NOT
THEN DROP
END
>> DOLIST        @Check if factor of first
DUP HEAD         @P(&N) is 1 and of second
1 == SWAP 2      @is -1 (Diff. of two seq.
GET -1 == AND    @members

```

```

AND
terms 1          @Find num. of occur.
<<              @of pattern "a("
->STR seqpatt
seqpatt 2 3
SUB DUP SREPL
NIP 1 ⚡
>> DOSUBS 1 +
<< AND >> STREAM
AND

```

We make exactly the same additional check, namely if the pattern "a(" appears more than once in the terms.

The corrected version of SEQTYPE recognises that this is a type of recurrence with which it doesn't deal and so programs that rely on this program behave themselves better. If you now enter the same sequence

$$a(n) = \frac{a(n-1)^2}{a(n-2)} \quad \{a(0) = a0 \quad a(1) = a0 \quad q\} \quad \text{and press}$$

GENFUNC, the program will error with "Can't deal with this kind of sequences". The same does now RCR->ANL.

Next thing we do is a program similar to SOLARSEQ, that takes the four arguments from the stack, *an*, *a0*, *n*, and *q* and solves for the first name that it finds.

```

<<
-> an a0 n q      @Store in local variable
'an=a0*q^n'      @Defining procedure
DUP LNAME        @Return vector of names
IF
DUP TYPE 5 ==    @If no names present
THEN
DROP2 COLLECT @Check if possible
NOT EVAL
ELSE
1 GET SOLVE      @Else solve for first variable
@in names vector

```

# Sequences, series and limits with the HP49G - Part 3

```
2 ->LIST
END
>>
```

We name this SOLGESEQ (in analogy to SOLARSEQ). Try some examples for different combinations of  $a_n$ ,  $a_0$ ,  $n$ , and  $q$ . Note that the program will eventually return solutions that are impossible if  $n$  is supposed to be integer. You must decide for yourself if the solutions are correct for your problem or not.

Let's now take a look again at the program CONVERGES?. The program uses CONDENSPT to find out if a given sequence converges or not by simply counting the condensation points. Actually the HP49G has a built-in command that can be used for this purpose. It is the well known command lim (LIMIT). When for example the index  $n$

of the sequence  $\frac{1}{n}$  goes to  $+$ , we can enter  $\frac{1}{n}$ , then  $n = +$  and

press [lim] to obtain 0, the limit of the sequence. For  $\frac{1}{n}$  it doesn't

matter if  $n$  is real or integer. For big integer or real values of  $n$  the

expression  $\frac{1}{n}$  approaches 0. But with other sequences, like for

example  $\frac{(-1)^n}{n}$  we get troubles. If you enter  $\frac{(-1)^n}{n}$ ,  $n = +$  and press

[lim] the HP49G answers with the error "Mode switch not allowed here". You see what happens. Having no way to specify that  $n$  has only integer values, there is the possibility that  $n$  takes real values as it approaches infinity. The expression  $(-1)^n$  will be in general complex if  $n$  is real and so the HP49G wants to switch to complex mode. And that is not allowed when we use lim. (Why?) The feature INTEGERASSUME is the most important missing thing of the HP49G.

But we can help ourselves using SPCASES. This program will turn

$\frac{(-1)^n}{n}$  to  $\frac{1}{n}$  and  $\frac{-1}{n}$ , the two possible outcomes when  $n$  is real.

Applying lim to each of these outcomes we get in both cases the result

0, which shows that the sequence  $\frac{(-1)^n}{n}$  converges to 0. We change

CONVERGES? so that it works this way. First find all possible outcomes, then apply lim to each of them. If all limits are the same and also not infinities or undefined, then the sequence converges. The new code is:

```
<< PUSH RCLVX -> vx      @Store current vx
<< SPCASES                @Find special cases
"Limit of special case" @Message
1 DISP 1
<< NSUB " of " + @Message # of current special
ENDSUB + 2 DISP@case
OBJ-> DROP OBJ-> DROP NIP =
IFERR lim                @If lim errors then return ?
THEN DROP2 ?
END
>> DOSUBS                @do to each special case
"Checking infinities
& undefined results"
1 DISP
IF DUP ? POS            @If some result is ? then
THEN ?                  @return ?
ELSE
IF DUP POS              @If we have infinities
OVER NEG POS OR
THEN 0                  @then return 0
ELSE
IF DUP SIZE 1 ==        @If only 1 limit
THEN 1                  @then return 1
ELSE                    @Else compare limits
"Comparing results
" 1 DISP DUP 2
```

# Sequences, series and limits with the HP49G - Part 3

```

<< ==      @Test limits equality
>> DOSUBS 1
+
<< AND
>> STREAM
END
END
END vx STOVX      @Restore vx (we do that because
>> POP           @lim sometimes changes vx
>>

```

enter the sequence  $a(n) = a(n-1) + \frac{1}{n}$   $a(1) = 0$  and press

CONVERGES? again, then the results are  $\{+ \}$  and 0, because the sequence doesn't converge.

If you take time to look at the overall interdependence of the programs, you see that SPCASES is a rather useful program for all the others over it. And this shows how badly an INTEGERASSUME feature is needed for the CAS of the HP49G. The only purpose of SPCASES is

Now we test. Though lim errors when we give it  $\frac{(-1)^n}{n}$ , the program CONVERGES? does its work

fine. Enter  $\frac{(-1)^n}{n}$   $\{n \ 1 \}$  and press

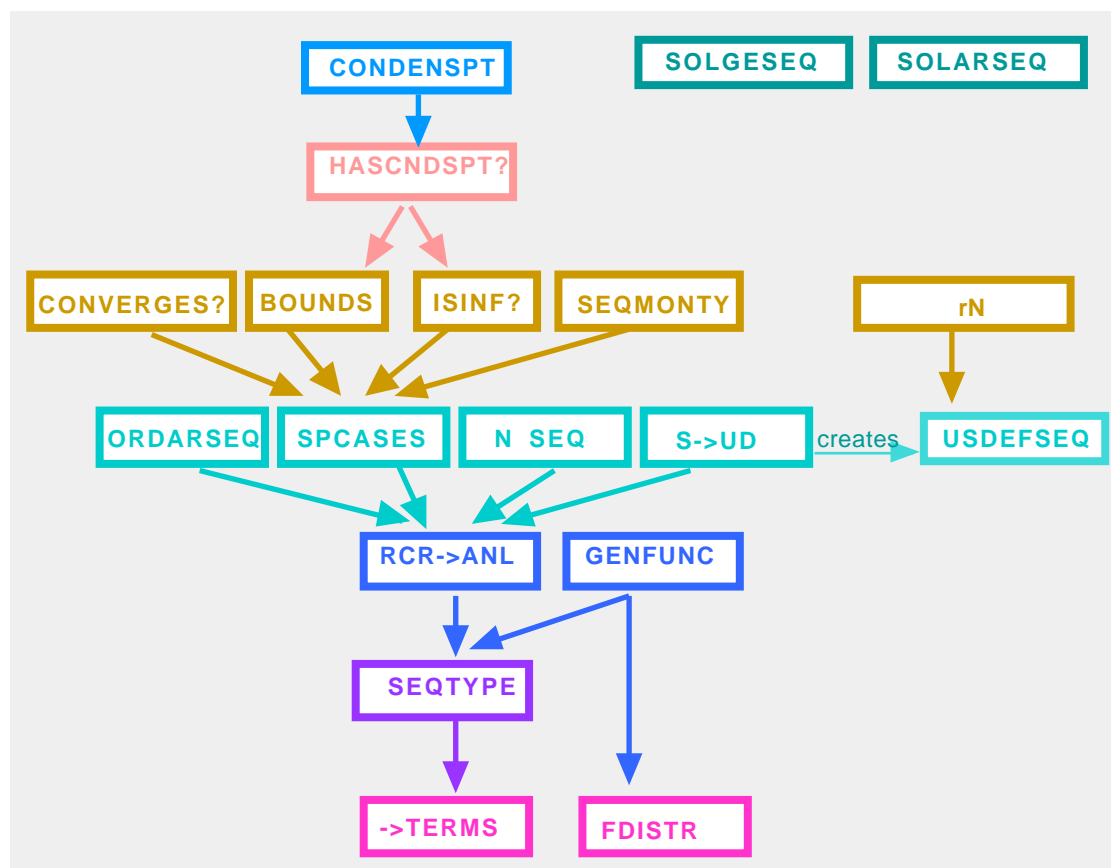
CONVERGES?. In some seconds you get the results, the list  $\{0 \ 0\}$  and a 1. The list contains

the limits of the two possible outcomes of  $\frac{(-1)^n}{n}$

when  $n$  is an integer from 0 to  $+$ . The 1 tells you that the sequence does converge. It converges to 0 the limit that is contained in the list. Another

example: Enter  $a(n) = a(n-1) + \frac{1}{n^2}$   $a(1) = 0$ .

Press CONVERGES?. The program calls SPCASES which in turn calls RCR->ANL to convert the sequence to its analytic closed form. Then SPCASES finds that there is only one possible outcome and CONVERGES? uses lim to find that the sequence converges to  $\frac{2-6}{6}$ . If you



# Sequences, series and limits with the HP49G - Part 3

to find what possible outcomes some expression have when its index is integer.

Though having  $\lim$  is already enough for many convergence investigations, we might sometimes want to use the convergence criterion of Cauchy. Mr Cauchy has found out that if the absolute value of the difference of two subsequent members of some sequence approaches 0 as the index approaches infinity, then the sequence converges. The HP49G can be used for this. (The interested reader should try to make a program that does what we are going to do now manually.)

Let's consider again the sequence  $\frac{(-1)^n}{n} \{n \geq 1\}$  for which  $\lim$

can't find the limit for  $n \rightarrow +\infty$ . Enter  $\frac{(-1)^n}{n}$  and press [ENTER] to

DUPLICATE the expression. Enter  $n = n + 1$  and press SUBST. Now you have  $\frac{(-1)^n}{n}$  on stack level 2 and  $\frac{(-1)^{n+1}}{n+1}$ . We want to find if

$\left| \frac{(-1)^n}{n} - \frac{(-1)^{n+1}}{n+1} \right|$  goes to 0 as  $n$  goes from 1 to  $+\infty$ . Because the

index starts at 1 we should help the HP49G by making the right assumptions. Enter 'n' then UNASSUME DROP. (We start with no assumptions for  $n$ .) Then enter  $n \geq 1$ , ASSUME and DROP again. Now the HP49G "knows" that  $n$  is greater than or equal to 1. Press the key [-] and then [ABS]. The HP49G rattles a little bit and returns

the result  $\left| \frac{(-1)^n}{n} - \frac{(-1)^{n+1}}{n+1} \right|$ . (Why did it rattle? This is not a very

"expanded" result.) Before we give this to  $\lim$ , we can simplify it. Press EXPAND. The HP49G switches to complex mode (another result of the inability to assume integer values for variables), and

returns  $\frac{\sqrt{\sin(n)^2 + \cos(n)^2} |2n+1| |n^2+n|}{\text{SQ}(n^2+n)}$ . Switch back

to real mode and press TRIG. (Remember the trigonometry marathon?)

After a few seconds the HP49G returns  $\frac{2n+1}{n^2+n}$ . This is something

which we can use with  $\lim$ . Enter  $n = +\infty$  to specify where the index goes to. Press [lim] to get 0. That shows us that

$\lim_{n \rightarrow +\infty} \left| \frac{(-1)^n}{n} - \frac{(-1)^{n+1}}{n+1} \right| = 0$  which means that  $\frac{(-1)^n}{n} \{n \geq 1\}$

converges. Note that the Cauchy criterion doesn't give us the limit itself, but only the fact that the sequence converges. It only says that if

$\lim_{n \rightarrow +\infty} |a_n - a_{n+1}| = 0$ , then  $a_n$  converges. Enter now 'n', UNASSUME and DROP to get rid of the assumption for  $n$ .

We have used assumptions many times in this marathon. While the HP49G provides ways to make new assumptions and get rid of assumptions programmatically, there is no built-in command that returns the current assumptions for some variable. So we program this functionality, which may prove useful for other programs.

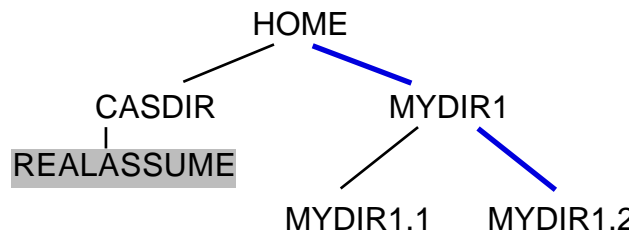
```
<<
PUSH 1 CF -> var          @Store variable in local
<<
{ HOME CASDIR REALASSUME } RCL @Recall REALASSUME
1
<<
IF
  LNAME AXL                @If variable isn't in
  var POS NOT              @assumption
THEN                       @Then drop assumption
  DROP
ELSE                       @Else set flag 1
  1 SF
END
```

# Sequences, series and limits with the HP49G - Part 3

```
>> DOSUBS      @Do to each assumption
IF             @If flag 1 is clear
  1 FC?
THEN          @Then return an empty list
  { }
ELSE          @Else return the first
  HEAD        @(and only) element of the list
END POP
>>
>>
```

with HOME and ending with the directory where the object resides and the name of the object itself. For example, the variable REALASSUME is in CASDIR and CASDIR is in HOME. If you need REALASSUME from, say MYDIR1.2, you put { HOME CASDIR REALASSUME } on the stack use RCL. Note that if the needed object is itself a program that must run, then you must use EVAL after recalling it. RCL will only put the object on the stack, no matter what type of object it is. The subsequent EVAL will run the recalled object if it is a program. The command PATH will return the current path.

Store this in RCLASSM. The program takes a name and returns the current assumptions for this name or an empty list if no assumptions exist for this name. Note that RCL can be used with a name to recall the contents of the variable that is in the current path, or with a list that represents a path to a variable.



Current path here:  
{HOME MYDIR1 MYDIR1.2}

Any variable in HOME, MYDIR1, MYDIR1.2 is accessible from MYDIR1.2

But you can recall REALASSUME from CASDIR by using the list {HOME CASDIR REALASSUME} with RCL. Doing this you remain in MYDIR1.2 .

Note also that while you are for example in a directory called MYDIR1.2, which is a subdirectory of MYDIR1, which itself is a subdirectory of HOME, you can access any variable that is somewhere in the path { HOME MYDIR1 MYDIR1.2 }. That means that any variable that is in HOME or in MYDIR1 will be evaluated exactly the same way as if it were in MYDIR1.2. A program that runs in MYDIR1.2 and needs to call some program in MYDIR1 has only to contain the unquoted name of the program in MYDIR1. Similarly, if you store a value in variable A in directory MYDIR1 and some program in MYDIR1.2 needs the value, you simply include the unquoted variable A in the program in directory MYDIR1.2.

But an object stored in a variable that doesn't reside somewhere in the current path, can't be accessed that way. One possible way to access such objects is to put their path on the stack and use RCL. The path to an object is a list. This list contains all the directories in order starting

Some additional comments on sequences. The recipe used for finding the members of sequences can be just about anything. Consider for example the following algorithm: Start at a certain integer. To find the next member of the sequence, add the product of all nonzero digits of the integer to the integer itself. For example, let's start at 1.

Member	Product of nonzero digits	integer+product
1	1	2
2	2	4
4	4	8
8	8	16



# Sequences, series and limits with the HP49G - Part 3

16	6	22
22	4	26
26	12	38
...	...	...

A program that takes a number of this sequence and calculates the next:

```
<<  DUP {}
      WHILE SWAP MANT DUP 0
      REPEAT DUP IP ROT + SWAP FP SWAP
      END
      DROP 1 + LIST
>>
```

The function MANT returns the mantissa of a number. The function LIST returns the product of all objects in a list. In the program we add 1 to the list of digits because if the number has only 1 digit, the function LIST will error as it wants at least two objects in the list. Since multiplication of anything with 1 doesn't change anything adding 1 to the list is cheap way to avoid errors.

If you store this program in PRODSEQ, you can just enter a number, (the initial condition) and calculate subsequent members of the sequence by pressing the soft key for PRODSEQ a couple of times.

Using the function SEQ you can also find several members at once. Enter the program

```
<<
  << DUP PRODSEQ >>
  'n' 1 9 1 SEQ +
>>
```

Store it in SEQPRODSEQ. It takes a number as the initial member and calculates the first 10 members of this sequence. For example enter 1. and press SEQPRODSEQ. Try some other initial values.

The strange thing about this sequence is, that no matter where you start

sooner or later you end up with the same sequence of numbers. For example the initial number 1. creates the sequence:

{1. 2. 4. 8. 16. 22. 26. 38. 62. 74. ...}

Of course starting at any member of the above sequence, that is 2, 4, 8, and so on, will produce the same numbers. But start at 3. Enter 3. and press SEQPRODSEQ. The result is:

{3. 6. 12. 14. 18. 26. 38. 62. 74. 102. ...}

As you can see, from 26 on we have again the same members. Some initial numbers need more time some less but they all finally do the same. Strange isn't it?

At the end of part 2, I said that in part 3 we are going to see how sequences can be plotted on the HP49G. I think that part 3 is already big enough and that plotting sequences deserves its own part, so let's leave that for the next part.

Ending this part, I want to thank all people out there again and especially Bill Storey and captain Adolph (AKA TimeToPaws) for sending me their corrections for the trigonometry and this marathon. Thanks you very much guys, that keeps me going :-)

$$\left\{ \text{Greetings}(n) = \text{Greetings}(n-1)^{n-1} \quad \left\{ \text{Greetings}(1) = g|_{g>1} \right\} \right\}$$

# Sequences, series and limits with the HP49G - Part 4

Hi again!

Since we mess up with sequences it would be interesting to take a closer look to the sequence of corrections of programs. Yes, you guessed right, there is yet another small correction to our programs. Suspicion comes that this sequence never converges ;-)

Anyway, it is only a couple of additional bytes that sometimes can be very helpful. Consider for example the sequence

$\frac{\ln(n)}{n} \{n \ 1 \}$ . If you give it to BOUNDS, then the result is

$\{? \ ?\}$  and ?. Why? Because SPCASES rebuilds the sequence and

returns  $\frac{\ln(|n|)}{n} \{n \ 1 \}$ . This makes things more difficult for

the subsequent operations, since an absolute value is involved, and no assumptions are made for  $n$ . You can check this by removing all assumptions for  $n$ , then entering  $n$  and pressing [LN]. If flag -119 is clear, that is, if you have rigorous mode on, then you get  $\ln(|n|)$ . But now, enter  $n \ 1$ , ASSUME, and enter  $n$  and press [LN] again. Now you have  $\ln(n)$ . (Remove the assumptions for  $n$  again.)

We add code that makes these assumptions and removes them when the programs are ready. We only need additional code for the programs SPCASES and BOUNDS. The additional code (in red) is the same for both programs and is listed in the following paragraphs.

SPCASES:

```
....
-> genmemb var lo hi explist perlist spclist
<<
  "Making assumptions" 1 DISP
  var UNASSUME DROP
  IF lo  $\infty$   $\neq$   $\infty$  NEG  $\neq$  AND
  THEN var lo  $\geq$  ASSUME 2 DISP
```

```
END
IF hi  $\infty$   $\neq$   $\infty$  NEG  $\neq$  AND
THEN var hi  $\leq$  ASSUME 2 DISP
END
"Checking powers of 1
and -1 containing "
.....
.....
ELSE genmemb var lo hi 3 ->LIST
      2 ->LIST 1 ->LIST
END
var UNASSUME DROP
>> POP
>>
```

BOUNDS:

```
....
-> set var lo hi
<<
  "Making assumptions" 1 DISP
  var UNASSUME DROP
  IF lo  $\infty$   $\neq$   $\infty$  NEG  $\neq$  AND
  THEN var lo  $\geq$  ASSUME 2 DISP
END
IF hi  $\infty$   $\neq$   $\infty$  NEG  $\neq$  AND
THEN var hi  $\leq$  ASSUME 2 DISP
END
"Building var. table of
"
.....
.....
<< MAX EXPAND >> STREAM DUP2 2 ->LIST
UNROT - ABS <
var UNASSUME DROP
>> 'flaglst' SWAP STO+
.....
```

# Sequences, series and limits with the HP49G - Part 4

Now BOUNDS returns  $0 \frac{\ln(3)}{3}$  and 1 if you give it the sequence

$\frac{\ln(n)}{n}$  {n 1 } . Of course the programs are as always updated, so you only need to transfer them to your HP49G.

Now that we have done the work let's go for fun. We are going to plot sequences with one and with two indices with the HP49G. Hold your breath, some fascinating things are coming. Trabakoulas lights up his cigar (no Havana, it's Greekana, the worst smelling cigars since Archimedes ;-), Nick lights his pipe and here we go.

First, let's do a simple analytic closed form sequence. Let's plot the sequence  $\frac{\ln(n)}{n}$  {n 1 } so that we can visually check if

BOUNDS returned the right results. The HP49G has no special plot type for sequences. But since we can tell the HP49G exactly which points of a function to plot, and to plot them without connecting them with lines, we are able to use the plot type FUNCTION for plot of analytic closed form sequences.

Press [BLUE-SHIFT], hold the button pressed and press [F4]. The input screen PLOT SETUP appears, and the input field "Type:" is highlighted. If this input field doesn't contain "Function", then do the following: Press the menu key "CHOOS" ([F2]). A pop-up menu with a list of all available plot types appears. Use the arrow keys to go to the plot type "Function". Press [ENTER]. Now use the arrow keys to go to the input field "EQ:". Here we must input the expression that we want to plot. Press [EQW] and type  $\frac{\ln(n)}{n}$ .

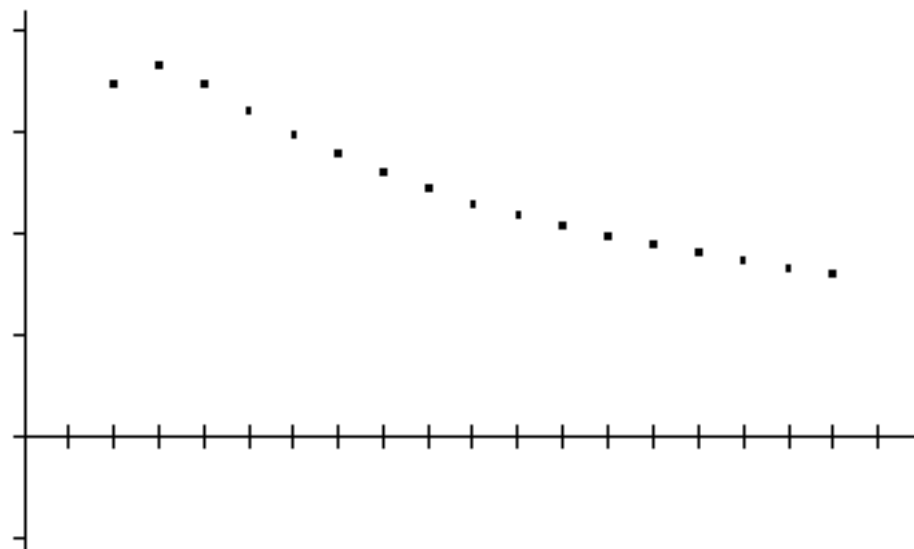
Press [ENTER] to put the expression in the input field "EQ:". Go to the input field "Indep:". Enter 'n'. This is the independent variable. Go to the option field "\_Connect". Use the menu key labelled "CHK" or the key "+/-" to uncheck this option. We don't want the connecting lines between the plot points but just the plot

points themselves. Now go to the fields "H-Tick" and enter 1. Go to the input field "V-Tick" and enter 0.1. Uncheck the option "\_Pixels" as the units of the tick marks are not pixels.

Press [BLUE-SHIFT], hold the button pressed and press [F2]. The input screen PLOT WINDOW - FUNCTION appears. Enter "H-View" from -1. to 20. and "V-View" from -0.1 to 0.4. Now we have specified which range of coordinates will be shown in the plot. We must also specify which points will be plotted. Go to input field "Indep

Low:" and enter 1. The first value that n in  $\frac{\ln(n)}{n}$  will take is 1. Enter 18. in the field "High". The last value of n will be 18. Enter 1. in the input field "Step" and uncheck the option "Pixels". Now we have specified that n will go from 1. to 18 in step of 1 unit and not 1 pixel.

Press the menu key [ERASE] and then the menu key [DRAW]. ([F4] and [F5]). The finished plot looks like:

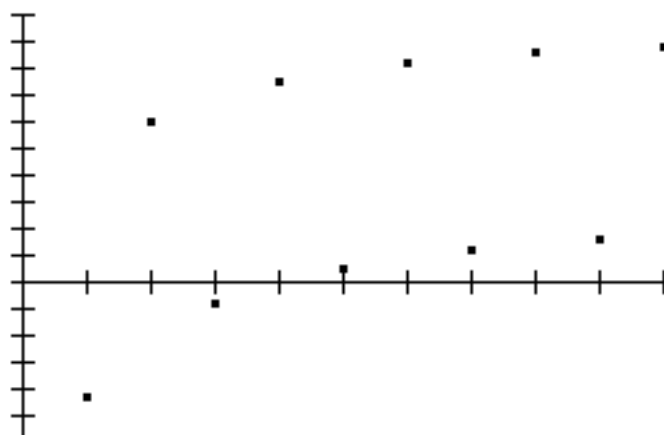


# Sequences, series and limits with the HP49G - Part 4

You can press the menu key [TRACE] ([F3]) to trace the plot. But tracing will not only be done for the plotted points but for all points of the continuous function  $\frac{\ln(n)}{n}$ , as if you had a plot point on every

column of pixels in the plot environment. Nonetheless you can use the keys [ARROW-LEFT] and [ARROW-RIGHT] to move the cursor to any plotted point. You can also press the key [+/-] that toggles normal and inverse cursor. When the cursor is inverse, then its pixels appear white if they are over a plot point. If you press the menu key [(X,Y)] ([F2]) or the key [+] once, then a pair of coordinates appears at the bottom of the screen, to let you know what the exact coordinates of the cursor are. Pressing the menu key [(X,Y)] ([F2]) or the key [+] or any menu key again, takes the coordinates away and redisplay the menu labels. If you press the menu key [TRACE] ([F3]) again, then tracing is deactivated and you can move the cursor free anywhere on the plot.

Press [ON] to leave the plot. Let's do another one. Press [BLUE-SHIFT], hold the button pressed and press [F4] to go to the input screen PLOT SETUP again. Enter  $\frac{(-1)^n (n+2) + 2 (n-1)}{3n+4}$  in the field "EQ:". Press [BLUE-SHIFT], hold the button pressed and press [F2] to go to the input screen PLOT WINDOW - FUNCTION again. Enter a horizontal view from 0 to 10 and a vertical view from -.6 to 1.



Enter 1 in "Indep Low:" and 10 in "High:". Press [ERASE] and then [DRAW] to see the plot.

"OK, but what can we do when we

have recurrences?" I hear you asking. Well, we can do much! In fact we can plot recurrences in a dozen ways, some of which will be described in the following paragraphs, just to wake your appetite.

First of all, here is another benefit of the program RCR->ANL. If it is able to convert a recurrence to its analytic closed form, then we can use this analytic closed form to plot the recurrence. Remember Fibonacci? Ha! Of course you do. Let's plot that. Enter the famous recurrence  $\{F(n) = F(n-1) + F(n-2) \quad \{F(0) = 0 \quad F(1) = 1\}\}$  and use the program RCR->ANL to convert it to its analytic closed form:

$$\frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^n + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^n \quad \{n \quad 0 \quad + \quad \}$$

function HEAD to get the first expression of the list on the stack:

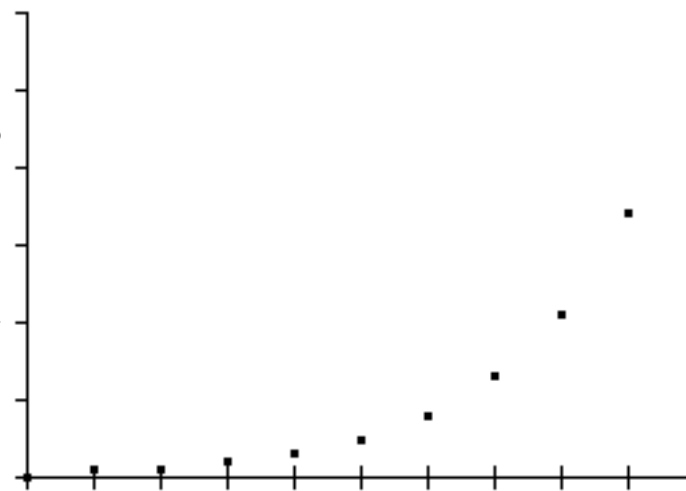
$$\frac{-\sqrt{5}}{5} - \frac{-1+\sqrt{5}}{2}^n + \frac{\sqrt{5}}{5} \frac{1+\sqrt{5}}{2}^n$$

. Enter STEQ to store it in EQ.

Go to the screen PLOT SETUP and enter 10 in "V-Tick:". Go to the input screen PLOT WINDOW - FUNCTION and enter -10 to 100 as vertical view range, and 0 to 10 as low and high values of the independent variable n.

Press [ERASE] and [DRAW] to see the plot.

Another recurrence? OK, let's go! Enter the sequence



# Sequences, series and limits with the HP49G - Part 4

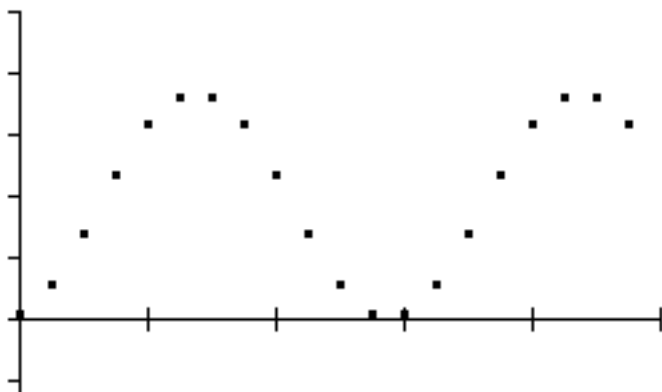
$F(n) = F(n-1) + \sin \frac{n}{6}$   $\{F(0) = 0\}$ . Press RCR->ANR. The

program returns:

$$\frac{(\sqrt{2} + \sqrt{6}) \cos \frac{6 \operatorname{ATAN}(2 + \sqrt{3}) + (n+4)}{6}}{2} + 2 + \sqrt{3}$$

$\{n \ 0 \ + \}$

after a while, so be patient. Enter HEAD. Though this expression can be simplified (using TEXPAND and EXPAND), you can directly STEQ to store it in EQ. Go to the input screen PLOT SETUP and enter 0.5 for "V-Tick:". Go to the input screen PLOT WINDOW - FUNCTION and enter 0 to 20 for "H-View:", -1 to 5 for "V-View:" and "High:". Press [ERASE] and [DRAW] to see the plot.



Now, I hear already the next question: "OK, when RCR-ANL can convert the recurrence to the corresponding analytic closed form, then we can plot. But what if RCR->ANL isn't successful? Can we plot then?".

Listening to this question Trabakoulas has that particular kind of smile that means only one thing: You bet we can! (No, this was not the

influence of Greekana cigars ;-)) Actually it is exactly this case, which lets our minds go rattle rattle again and makes us explore the not so standard graphs.

Let's have an example of a recurrence that can't be plotted as an algebraic object but as a program. We consider the sequence

$$T(n) = \frac{2}{3} T(n-1) + 50 \quad \{T(0) = 100\}$$

. How could we plot that?

We start with the "do all yourself method". We make a program that calculates all pairs of x-y-coordinates and turns all corresponding pixels on. Consider for example:

```
<<
ERASE
{#0 #0} PVIEW
DRAX
100 -> T0      @Store initial value of 100 in T0
<<
1. 15.         @We plot for n=1 to n=15
FOR n
  n            @Put current n (indep. var.) on stack
  IF          @If n=1
    n 1. ==
  THEN        @then put T0 on stack
    T0
  ELSE        @else "cook" T(n)
    2. 3. / T0 * 50 +
  END
  DUP 'T0' STO @Store a copy on T(n) in T0
                @(to use it for the next n)
  R->C PIXON   @Convert to complex and turn
                @the corresponding pixel on.
NEXT
>>
7 FREEZE
>>
```

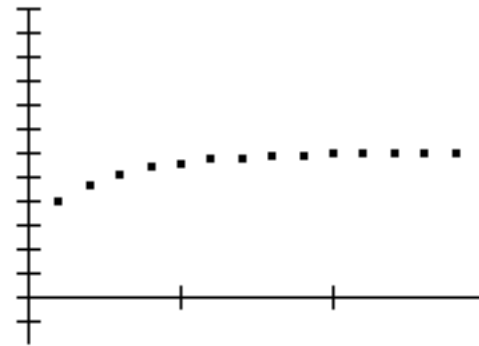
# Sequences, series and limits with the HP49G - Part 4

While you are typing this program, you notice that we calculate:

```
2. 3. / T0 *
```

Since we only need numeric results we could just multiply T0 by .666666666667. But on the other hand it is really cumbersome to enter this number by hand, while entering `2. 3. /` is easier. But the HP49G has the capability of replacing `2. 3. /` with .666666666667 while we are typing/editing the program, that is on the fly! While you edit, use the arrow keys to move the cursor until it is over the entered `2.` Now press [RED-SHIFT] and then [APPS] set the beginning of the marked text. Use again the arrow keys to put the cursor just after the entered `/`. Then press [RED-SHIFT] and then [MODE] to set the end of the marked text. The whole part `2. 3. /` is now highlighted. Press the key [TOOL], then press twice [NXT]. The menu key on the left ([F1]) is now labelled "EXEC". Press it and see how the entered `2. 3. /` gets replaced by the number .666666666667, which is the result of the operation. EXEC is an editing help. It will try to execute whatever instructions in a program are marked and will replace them with their result. Store this program in RECPlot1.

Before you run it, we must setup the plot parameters. Press simultaneously [BLUE-SHIFT] and [F4] to go to the input screen PLOT SETUP. Here we only need to set "H-Tick:" to 1 and "V-Tick:" to 25. We also uncheck the option "\_Pixels". Press simultaneously [BLUE-SHIFT] and [F2] to go to the input screen PLOT WINDOW - FUNCTION. Enter "H-View:" from 0 to 15 and "V-View:" from -50 to 300. Press [ENTER] to accept the settings. Now let RECPlot1 run. The produced plot looks like on the next picture. The program that we used to do this plot doesn't really use the built-in plot types but rather some of the general graphics commands of the HP49G. The command ERASE, just clears the current plot picture. DRAX draws a pair of axes. The command PVIEW shows the current plot picture with the coordinates (specified at stack level 1) at the upper left corner of the screen. These coordinates can be of two types. Either a list with two binary integers or a complex number. The list with two binary integers represents pixel coordinates. If we start counting the pixels of a graphics object from its upper left corner starting at #0 horizontally and



vertically, proceeding downwards and to the right, then we get the pixel coordinates of any pixel of the graphics object. This type of coordinates is absolute. On the other hand, when we setup the view range of some plot, we have another set of coordinates, the user coordinates. These are given as complex numbers. If we

denote the view range with  $X_{\min}$  to  $X_{\max}$  horizontally and  $Y_{\min}$  to  $Y_{\max}$  vertically, then the upper left corner of the graphics object is  $(X_{\min}, Y_{\max})$ , the upper right corner is  $(X_{\max}, Y_{\max})$  and so on. The commands for transforming one type of coordinates to the other type are P->C and C->P.

Our program does nothing more than calculating the coordinates of the pixels to be turned on, and turning them on with PIXON. (There is also the command PIXOFF which turns pixels off.) When it finishes, it uses the command FREEZE to let the plot persist. (Otherwise the HP49G would show the normal stack display again at the end of the program.) FREEZE takes a real number or integer as argument. To understand how FREEZE works, you must know that the display is divided in three areas. Staring at the top, the first is the status area where the HP49G displays information about settings. The second is the stack area, the area where inputs and outputs are shown. The third is the menu area, which shows the labels of the current menu. Each of these areas has a corresponding number. The status area has the number 1, the stack area has the number 2 and the menu area has the number 3. FREEZE uses these numbers to "freeze" what is shown in some particular area after a program has been run. For example, if you want to let the status area persist after a program has finished, you must include `1 FREEZE` in your program. For combinations of areas you can use the sum of these numbers. If you for example want to hold the status and stack area "frozen" after the end of a program, then

# Sequences, series and limits with the HP49G - Part 4

you must include 3 FREEZE in the program, because the sum of the numbers for the status and the stack area is 1+2=3. The retain the whole display you use the sum of all areas, that is 7.

	Status area	1
	Stack area	2
	Menu area	4

When you press any key after the program has finished and the display is "frozen", then the current display is restored. (In our case the normal stack display.) You can press key [ARROW-LEFT] to go to the plot environment again. You can turn on the display of cursor coordinates by pressing [F2] or [+] and you can move the cursor around using the arrow keys. But that's all. No tracing can be done in this case, because we didn't store the equation to be plotted in EQ. We did all ourselves, no built-in plot was used. That is one disadvantage of the "do it all yourself" method. The other is, especially for a recurrence, that we can't change the initial conditions so easily. We must edit the program and change the appropriate line for example to:

200 -> T0

to see what happens when we start with 200. Also, if we want to keep the already plotted sequence and superimpose it with the new, we must remove the command ERASE from the program. And we must do that again and again for any new initial value. As experimenting with different initial values of recurrences can be very interesting, you can imagine that doing all ourselves is a bit cumbersome. The advantage of this method however, is that you can draw just about anything. If you can calculate the coordinates of the pixels to be turned on, then you can always use this method.

Having the first disadvantage in mind, the impossibility of using the tracing functions of the HP49G, we search for another way to plot

$$T(n) = \frac{2}{3} T(n-1) + 50 \quad \{T(0) = 100\}$$

We must somehow put

the "cooking recipe" for the members of this sequence into EQ and let

the HP49G plot it. For this purpose it is of great help to understand how the HP49G does function plots.

You remember what happened as we examined if there is a way to

$$\text{solve } \sum_{n=0}^r \frac{(-1)^n}{n!} \frac{r!}{(r-n)!} a(r+N-n) = 0 \text{ using ROOT? We}$$

solved a program instead of the above expression. Well, the same can be done when plotting. When the HP49G plots a function it starts evaluating the object in the system reserved variable EQ for different values of the independent variable. These different values start at the value which we input in "Indep Low:" and end at the value which we input in "High:". In case we didn't enter anything in these input fields, the values of horizontal and vertical view range are used. The HP49G steps from the lower to the upper value in steps of the value that we enter in "Step:". If we don't enter anything there, then a default of "every second pixel of the screen" is used. The value of the independent variable is "kept in mind" while it is used to evaluate the object in EQ. The two corresponding values, independent variable value and object evaluation value, build together a pair of x-y-coordinates that correspond to a certain pixel on the screen. This pixel is turned on and the HP49G continues with the next value of the independent variable. That means, that anything that returns one real number at stack level 1 when it is evaluated, can be used for a function plot. Programs, not all programs but those which behave this way, can be used for plotting. We consider the sequence

$$T(n) = \frac{2}{3} T(n-1) + 50 \quad \{T(0) = 100\}$$

. How can we make a

program that evaluates to exactly the same numbers like the sequence? Well take a look at:

```
<<
IF                               @If n 0
  n 1.
THEN                             @then put 100 on stack
  100.
```

# Sequences, series and limits with the HP49G - Part 4

```
ELSE                @else "cook" T(n)
  2. * 3. / 50 +
END
DUP                @Make a copy of current T(n) for the
>>                @next evaluation
```

Store this in EQ (STEQ). Press [RED-SHIFT], then [BACKSPACE] to clear the entire stack!! That's important!! Let's setup the plot parameters. Press simultaneously [BLUE-SHIFT] and [F4] to go to the input screen PLOT SETUP. Choose plot type "Function". Set "Indep:" to 'n', then "H-Tick:" to 1 and "V-Tick:" to 25. Uncheck the option "\_Pixels". Press simultaneously [BLUE-SHIFT] and [F2] to go to the input screen PLOT WINDOW - FUNCTION. Enter "H-View:" from 0 to 15 and "V-View:" from -50 to 300. Enter 1 for "Indep Low:", 15 for "High:" and 1 for "Step:". Uncheck the option "\_Pixels". Now press [ERASE] and then [DRAW]. Voila! The plot is done in glory. Now, before pressing [TRACE], move the cursor horizontally leftwards until it is at one pixel to the left of the first plotted point. The vertical position of the cursor doesn't matter. (We will soon see why.) Press [TRACE]. If you press the key [ARROW-RIGHT] now, you can see that the cursor doesn't follow the plotted points! It goes higher much quicker than the sequence and arrives at the final value of about 150 much earlier. Why is that? Well, many questions, so the answer is a bit long but worth it. First of all, the program that we used to plot the sequence, is evaluated by the HP49G. The current value of the independent variable n, is checked if less than or equal to 1. In this case the number 100 is put on the stack. If greater then it is first multiplied by 2... and here we have the problem. When you press [TRACE] the x-current coordinate of the cursor is used as the value of the independent variable. If you start at some coordinate greater than 1, then the following part of the program is evaluated:

```
ELSE                @else "cook" T(n)
  2. * 3. / 50 +
END
DUP
```

You see that the ELSE-part of the IF-THEN-ELSE clause multiplies the number on stack level 1 with 2. But there is no number on stack level 1. (Or if there is one, it remained there from other calculations and the plot will be wrong. That's why you had to clear the whole stack at the beginning.) So if you start tracing while the cursor is at some x-coordinate greater than 1, you will get an error "Too few arguments". If you start at 1 (or a bit smaller), then the number 100 will be put on the stack and the DUP command at the end of the program make a copy of the value for calculation of the next point, so that tracing works. (You can think of tracing like pixel-for-pixel evaluation of the object stored in EQ.) The second mysterious phenomenon, namely the faster ascending of the cursor, has to do with our definition of the object in EQ. Notice that the independent variable is only used for just comparing its value with 1. It is not used for calculating the next y-coordinate value. It is more or less a dummy which allows us to just put an initial value on the stack, a value that is used as a hook for the rest of the calculations and tracing. When you press the key [ARROW-RIGHT] to trace further to the right, the program is evaluated again using as value for the independent variable the current x-coordinate of the cursor. When the x-coordinate becomes a tiny little bit greater than 1, then the procedure is used: "multiply by 2, divide by 3, add 50" for calculating the y-coordinate. Each time you go one pixel to the right, this procedure is used again. So you have the y-coordinates growing for each pixel and not for every next integer value of n. We should correct this unusual behaviour in order to be able to do better tracing. One way to do that would be use a procedure that applies the procedure "multiply by 2, divide by 3, add 50" n-1 times, n being the integer part of the current x-coordinates of the cursor. take a look at the following program:

```
<<
100                @Put 100 (initial value) on the stack
IF                 @If n>1
  n 2.
THEN               @then "cook" T(n) with 100 as the
  1 n IP           @initial value
START
  2. * 3. / 50 +
```



# Sequences, series and limits with the HP49G - Part 4

```

NEXT
END
>>

```

Store this in EQ and plot it with the same settings like the previous plot. And now the great moment of truth arrives. Press [TRACE] and trace freely back and forth.

The only thing that remains is to have an easy way to define many different initial values. Of course we could edit the program stored in EQ each time we want another initial value. But it would be just boring to leave the plotting environment, edit the program, draw again and so on. There must be an easier way. And guess what? There is. We can define different initial values by just using the graphics cursor, that is without having to leave the graphics environment! First edit the above program and change it to:

```

<<
T0          @Put T0 (initial value) on the stack
IF          @If n>1
  n 2.
THEN        @then "cook" T(n) with 100 as the
  1 n IP     @initial value
  START
    2. * 3. / 50. +
  NEXT
END
>>

```

Re-STOre that in EQ. Next enter the following program:

```

<<
DEPTH ->LIST @Make a list out of the stack objects
-> stack    @Store it locally
<<
ERASE DRAX
DO
  PICTURE   @Activate the graphics environment
  IF

```

```

DEPTH      @If then user has pressed [ENTER]
THEN
  IM        @Then store the imaginary part of
  'TO' STO  @the coordinates in T0,
  DRAW 0    @draw, return 0.
ELSE        @Else restore stack, return 1
  stack
  OBJ->
  DROP
  1.
END
UNTIL
END
>>
>>

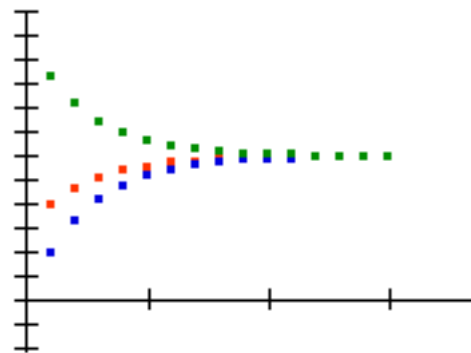
```

Store this program in RECINTERACT. The command PICTURE activates the plotting environment and all its comfort and the program that contained it just stays halted at this point. When you press [ON] the subsequent commands of the program are executed.

Run the program RECINTERACT with the same plot settings like before. First thing you see is an empty graphics screen. Press the key [+] to see the cursor coordinates and move the cursor using the arrow keys to an y-coordinate of about 233. (The x-coordinate doesn't matter.) Press [ENTER] to let the sequence be drawn again with the new initial value. When the plotting is done, move to some other y-coordinate, press [ENTER] and then [ON] to see another sequence. Play around as long as you wish. If you want to exit the program, don't press [ENTER] but only [ON] after a plot has finished. Pressing [ENTER] after the plot has finished and while in the plot environment, just puts the current cursor coordinates as a complex number on stack level 1. This complex number will be used by the program to get its imaginary part (y-coordinate) and use it as the new initial value. When you press [ON] you leave the plotting environment. This will let the program continue, draw the sequence for the new initial value, and stay at the plotting environment again. Notice that when a new initial value is entered by pressing [ENTER] in the

# Sequences, series and limits with the HP49G - Part 4

plotting environment, the program detects this by checking if the stack is empty or not. We can use this technique because we make a list of all stack objects and store it in the local variable "stack" before we start doing anything else. That is, when the graphs are drawn, we can assume that there is nothing else on the stack, except (eventually) coordinates entered by the user. If the stack contains something we draw again and also return a 0, which is used by the DO-UNTIL-END LOOP to decide that the user might want another plot. If you don't enter any additional initial values, that is if you don't press [ENTER] but only [ON], then the program detects that there is nothing on the stack and exits, putting first the original stack objects back on the stack and a 1, which is used to exit the DO-UNTIL-END loop. You can see the plot for three different initial values on the picture on the right. (This was Rcobo's HP49G, equipped with colour display ;-))



Another method? OK, we proceed to the more hidden features. Do you know STORE? No, not the command STO, I mean the function STORE. Yes, it is a function and it is allowed in algebraic objects! In stack syntax it takes any object from stack level 2 and a name from stack level 1. It stores the evaluated object in the name *and* returns the result of the evaluation to stack level 1. In algebraic syntax you write: STORE(object,name). Go on and enter STORE(X-1,Y) in the EQW. Put that on the stack, press EVAL. Now you have a new variable Y which contains X+1, and the content X+1 itself is on stack level 1. Which is exactly what we need to formulate plottable algebraic expression for recurrences that can't be transformed to analytic closed forms. Let's consider the recurrence

$$T(n) = \frac{2}{3} T(n-1) + 50 \quad \{T(0) = 100\} \quad \text{once more. (This can be}$$

transformed to an analytic closed form, but let's use it anyway.) In its

first incarnation let the corresponding algebraic be  
STORE  $\frac{2}{3} T0 + 50, T0$ . Enter that, and STEQ to put it in EQ.

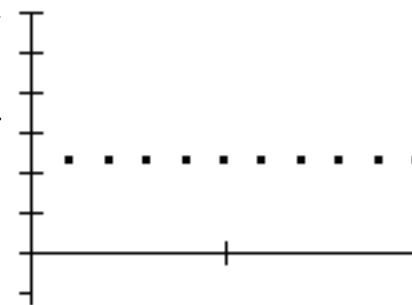
Store the initial value of 100 in T0. Still with the same plot settings, try to plot it. It simply evaluates  $\frac{2}{3} T0 + 50$ , stores the result in T0

and puts the same result in stack level 1. Since the function plotting software evaluates the object in EQ repeatedly, we would expect it to draw the same sequence again. But it doesn't! Instead of this it plots all points at the y-coordinate of about 116.67 which is the numeric result of the first evaluation of the algebraic  $\frac{2}{3} T0 + 50$ . (To see that, store

100 in T0 again, recall EQ and press [EVAL], [->NUM].) So the plot looks like the picture on the right. Why is that? Well, I am not sure about that, but through experimentation I believe that the explanation is the following. Have you ever noticed that there is a more or less observable delay between the key press [DRAW] and the real start of the drawing? You get a small (or big) pause and then the plot goes on faster than what you might have thought when the pause occurred. The question has been often asked in the news group, "what does the calculator do in this time?" I believe that one of the things that are done in this time is the complete numeric evaluation of the function, except of the independent variable, which takes different values later on, when the HP49G actually plots. Our function,

STORE  $\frac{2}{3} T0 + 50, T0$ , doesn't contain the independent variable

at all. It is first numerically evaluated to 116.67, since T0 already contained 100. It is this *evaluated* form that is used later to plot the points of the sequence. Since this is a plain simple real, nothing changes while n takes the values 1,2,... and so on. So, the only y-coordinate is this number, 116.67. We need a



# Sequences, series and limits with the HP49G - Part 4

mechanism that prevents this evaluation. And guess what? We have it!

Change the current EQ to  $\text{QUOTE STORE } \frac{2}{3} T_0 + 50, T_0$  and

re-store it in EQ. [ERASE] and [DRAW] again. This works! Store 50 in  $T_0$  and [DRAW]. The combination QUOTE/STORE does miracles here. If you want to the function itself to contain the initial value, then go to the EQW and enter the function to be plotted as:

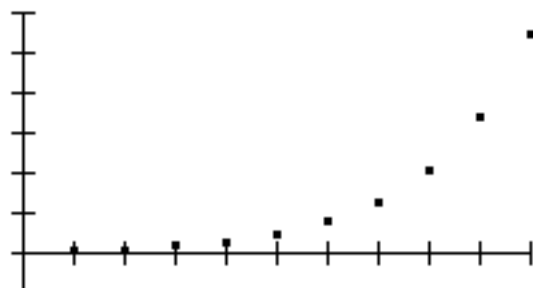
$\text{IFTE } n < 2, \text{STORE}(50, T_0), \text{QUOTE STORE } \frac{2}{3} T_0 + 50, T_0$  .

This automatically stores the initial value of 50 in  $T_0$  and returns 50 for  $n = 1$  (and all values of  $n$  less than 2). But for  $n \geq 2$  it calculates the next point out of the current point.

Using the same method, we can write for the Fibonacci sequence:

$\text{IFTE}(n < 1, \text{STORE}(0, F_0), \text{IFTE}(n < 2, \text{STORE}(1, F_1), \text{QUOTE}(\text{STORE}(F_0 + \text{STORE}(F_1, F_0), F_1))))$

Set "V-View:" from -10 to 60, "Indep Low:" to 0, "High:" to 10 and step to 1 (user units). Let it plot! Slow, but it does its work.



So we have yet another method to plot sequences given as recurrences. What a blessing in disguise this STORE was! And still we can do a little better. You have already noticed that the last couple of methods leave an additional variable in the current directory when the plot is done. Of course we can manually purge it, but we can also let the HP49G do that automatically when it finishes the plot. For the plotting programs given so far, it seems to be easier. We simply add PURGE at the end. For example the end of the program RECINTERACT can be changed to:

```
.....
UNTIL
END
>>
'T0'      PURGE @Additional code
>>
```

But what can we do if we use STORE in an algebraic? Well, there is also the function UNASSIGN. This function takes a variable name from the stack, it returns the unevaluated contents of the variable on the stack and purges the variable. It also allowed in algebraic objects (hurrah!) where it has the syntax:  $\text{UNASSIGN}(\text{name})$ . So if we want to get rid of  $T_0$  after, say the tenth point of

$T(n) = \frac{2}{3} T(n-1) + 50$  {  $T(0) = 100$  } has been plotted, we could

change the equation so that it uses UNASSIGN to purge the variable  $T_0$  that isn't needed anymore. I have tried to do that, with QUOTE and without

QUOTE, but no use. The variable  $T_0$  still remained there after the plot was done. So perhaps some of the people out there could experiment and tell us what to do. Anyway, we already have seen that STORE and UNASSIGN are not useless at all. They allow us to do something that has been thought impossible. Trabakoulas looks at the horizon and thinks of all the other hidden things waiting to be found.

While he wonders we still insist to find yet another way to get rid of the variables that hold the initial values after the plot is done. What about local variables? They are removed after some program has done its work, so they seem to fit. The code of RECINTERACT can be changed to:

```
<<
DEPTH ->LIST      @Make a list out of the stack objects
0.                @and enter a 0.
-> stack <-T0      @Store locally, <-T0 is a compiled
                  @local variable.
<<
```

# Sequences, series and limits with the HP49G - Part 4

```

ERASE DRAX
DO
  PICTURE      @Activate the graphics environment
  IF
    DEPTH      @If then user has pressed [ENTER]
  THEN
    IM          @Then store the imaginary part
    '<-T0' STO   @of the coordinates in <-T0,
    DRAW 0     @draw, return 0.
  ELSE
    @Else restore stack, return 1
    stack
    OBJ->
    DROP
    1.
  END
UNTIL
END

```

>>

>>

At the same time we change the program in EQ to:

```

<<
<-T0          @Put <-T0 (initial value) on the stack
IF            @If n>1
  n 2.
THEN          @then "cook" T(n) with 100 as the
  1 n IP      @initial value
  START
    2. * 3. / 50. +
  NEXT
END
>>

```

When RECINTERACT finishes now, there is no global variable T0 left in the current directory, because we never used one. (Why does the local variable have to be a *compiled* local variable? Why not an ordinary local variable?)

Perhaps you wonder if storing the following program in EQ would work:

```

<<
.666666666667 * 50. +      @Calculate
DUP                        @Copy for next evaluation
>>

```

Store that in EQ and enter 50., the initial value required to plot it. Set "Indep:" to 'n', "H-Tick:" to 1 and "V-Tick:" to 25, "H-View:" from 0 to 15 and "V-View:" from -50 to 300, "Indep Low:" to 1, "High:" to 15 and "Step:" to 1. If you ERASE and DRAW, then you see that it also works. But the initial value of 50. isn't on the plot. We must make the program better so that if we plot the first point, the value of the initial value will be used.

```

<<
IF
  n 2.                  @If we don't plot the
THEN                    first point, then
  .666666666667 * 50. + @Calculate
END
DUP                    @Copy for next evaluation
>>

```

Store this in EQ, enter 50. and re-ERASE, re-DRAW. This time the initial value is plotted too. (The value left on the stack after the plot is ready, is the copy created when DUP is executed for the last time by the program in EQ. Since nobody uses it any more, it stays all alone on the stack, poor number.)

Using the last program in EQ, we can change RECINTERACTIVE a little bit, so that we can use many different initial values interactively.

```

<<
DEPTH ->LIST    @Make a list out of the stack objects
-> stack
<<

```

# Sequences, series and limits with the HP49G - Part 4

```

ERASE DRAX
DO
  PICTURE      @Activate the graphics environment
  IF
    DEPTH      @If then user has pressed [ENTER]
  THEN
    IM         @Put the init. value on the stack
    DRAW DROP  @Draw and drop the poor number.
    0          @Return 0.
  ELSE        @Else restore stack, return 1
    stack
    OBJ->
    DROP
    1.
  END
UNTIL
END
>>
>>

```

This works just like before and lives no poor numbers on the stack when we finish plotting.

Until now we have used many different ways to plot sequences, but the plot type was always FUNCTION. Actually there is another plot type which can be used to plot sequences. This plot type is "Scatter" and it was initially thought for statistical plots, but we can use it to plot any sequence of points. First of all let's look at the description of the plot type and the usage of the plot parameters.

The HP49G uses a reserved variable named `DAT` to hold the data that are used for statistics. This is a matrix with the variable data in columns. To do a scatter plot we must first choose the plot type SCATTER. Obviously this can be done interactively in the PLOT-SETUP screen by choosing "Scatter" as the plot type. Programmatically the same can be done with the command SCATTER. Because `DAT` can have any number of columns (that is, any number of variables) we must also specify which of the columns will be used

as the horizontal variable and which will be used as the vertical variable. On the PLOT SETUP screen this can be done by entering the number of the column in the input fields "Cols:". If for example the data in the first column must be plotted against the data in the second column, we just enter 1 and 2 in these fields. The commands for doing the same in programs are XCOL, YCOL and COL. XCOL and YCOL both take a number  $n$  from the stack and assign the role of the

x- or y-variable to the corresponding column of the matrix in `DAT`. COL takes both numbers from stack level 2 (x) and 1 (y) and uses them to assign the roles of both the x- and y-variables to the corresponding columns of the matrix. Notice that

	var <sub>1</sub>	.....	var <sub>n</sub>
Obs.1	value <sub>1,1</sub>	...	value <sub>1,n</sub>
	⋮	⋱	⋮
Obs.n	value <sub>m,1</sub>	...	value <sub>m,n</sub>

these settings can be also made from other inform screens. For example, if you press [RED-SHIFT] and then [5] you are presented a pop-up menu. If you choose "3. Fit data..." from this menu, then you are presented the FIT DATA screen where you can set the x- and y-column. If you want to put the commands for statistics in programs and you prefer menu hunting than typing them yourself, then you can enter 96.01 and then MENU. You are then taken to the old fashioned main menu for statistics, which contains other menus with commands like XCOL, YCOL and so on.

Now, the parameters that affect the scatter (and any other statistics type) plot, are kept in another system reserved variable named `PAR`. This variable is a list:

{ x-col y-col intercept slope model }

For the time being it is of interest for us to know that the first two parameters in `PAR` are the numbers specifying the x- and y-column. (We will take a closer look to the other parameters in future.)

Of course the settings that affect the lookings of the plot are stored as parameters in `PPAR`. (Look also at the complex numbers marathon.) `PPAR` is also a list:

# Sequences, series and limits with the HP49G - Part 4

{ (xmin,ymin) (xmax,ymax) indep res axes ptype depend }

The meaning of these parameters for the scatter plot is:

(xmin,ymin): A complex number which specifies the lower left corner of the display range. The value used as default is (-6.5,-3.1). The programmable command for this parameter is PMIN.

(xmax,ymax): A complex number which specifies the upper right corner of the display range. The value used as default is (6.5,3.1). The programmable command for this parameter is PMAX.

indep: A name specifying the independent variable. Default is X. Programmable command: INDEP.

res: not used with this plot type.

axes : A list that has one or more of the following elements in order. A complex number specifying the coordinates of intersection of the axes, a list that specifies the tick marks of the axes and two strings that are used as labels for the X- and the Y-axes. Programmable commands for these parameters are: AXES, ATICK. AXES takes as argument either a complex number with the intersection coordinates of the axes or a list containing all the axes parameters. ATICK takes as parameters either a number specifying the tick marks annotation distance in user units for both axes, or a list with two numbers for separate tick marks settings for the x- and y-axes, or a binary integer specifying the tick marks annotation distance in pixels for both axes, or a list with two binary integers for separate tick marks settings for the x- and y-axes in pixels.

ptype : The command name SCATTER.

depnd: Name specifying dependent variable. Programmable command: DEPND.

Now that we know about the parameters for the plot, let's plot the simple sequence  $\frac{(-1)^n}{n}$  {n 1 } using a scatter type plot. We

first create the data to be plotted. Enter n, press [ENTER] to make a copy, then enter 1, 15, 1. Press SEQ to generate the list {1 2 ... 15}. These are going to be our x-column data. The command SEQ took an expression (in this case simply n) from stack level 5, the variable n from stack level 4, and the start, end and step values from stack levels 3, 2 and 1 respectively. It evaluated the expression n for different values of the variable n starting at 1 and

ending at 15 using a step of 1. Enter now  $\frac{(-1)^n}{n}$ , n, 1, 15, 1 and use

SEQ again to get the list  $-1 \frac{1}{2} \dots \frac{-1}{15}$ . These are going to be the y-column data. We must convert these lists to a matrix and store the matrix in DAT. This is an easy task with the HP49G. Enter 2 and press ->LIST to create the list {1 2 ... 15}  $-1 \frac{1}{2} \dots \frac{-1}{15}$

and use AXL to convert the list of lists to the matrix

$\begin{matrix} 1 & 2 & \dots & 15 \\ -1 & \frac{1}{2} & \dots & \frac{-1}{15} \end{matrix}$ . Now, the data are not in columns but in rows, so

we must transpose the matrix. Enter TRAN to get the transposed

matrix:  $\begin{matrix} 1 & -1 \\ 2 & \frac{1}{2} \\ \vdots & \vdots \\ 15 & \frac{-1}{15} \end{matrix}$ . (The command TRAN is in the second page of

# Sequences, series and limits with the HP49G - Part 4

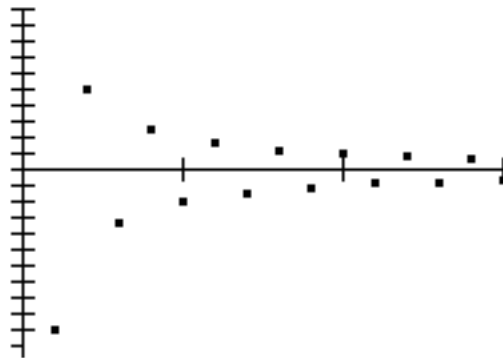
menu MATRICES/OPER). Now use the command STO to store the data in DAT. Go to the PLOT WINDOW - SCATTER screen and set H-View from 0 to 15 and V-View from -1.1 to 1. ERASE and DRAW to see the plot. Now, this was a rather simple plot, but how do we produce the data for, say the recurrence

$$T(n) = \frac{2}{3} T(n-1) + 50 \quad \{T(0) = 100\} \quad ? \text{ Well, one way is to use}$$

again SEQ, but a little bit... crazier. First of all let's do the x-column data. Enter  $n$ ,  $n$ , 1, 15, 1 and press SEQ to get the list  $\{1 \ 2 \ \dots \ 15\}$ . That was easy. For the y-column data we don't have an analytic closed form to use as expression on stack level 5. But SEQ accepts also programs instead of algebraic expressions. Enter first 100, the initial value of the sequence. Then, enter the program:

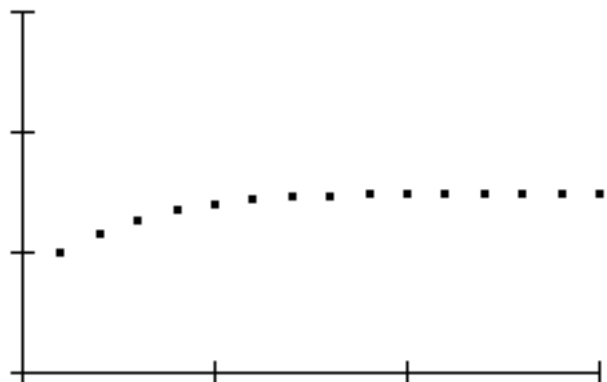
```
<<
  DUP                      @Copy for next evaluation
  .666666666667 * 50. + @Calculate sequence member
>>
```

This program works on a copy of the number on stack level 1, and creates the next member of the sequence. Enter now  $n$  or any other name. (This name doesn't appear in the program, so in this case we use it as a place holder, a dummy for the command SEQ, which varies  $n$  in the specified range of values, and then evaluates the program for each current value of  $n$ . Since the program doesn't contain  $n$ , the evaluation isn't affected by this value and only does what the program says.) Enter 1, 14, 1 and press SEQ. The results are 100. in stack level 2 and the list  $\{116.6667 \ \dots \ 149.828873918\}$  on stack level



1. Press  $[+]$  to include the initial value in the list. Now 2  $\rightarrow$  LIST, AXL, TRAN and STO. Set V-View from 0 to 300 and ERASE DRAW again.

So we have yet another way to plot sequences. There is really a wealth of different methods to plot even recurrences, for which perhaps one might think that they can't be plotted at all using the HP49G. Again we see very clearly that a flexible set of available commands allows to do much more than simply having for example a rich predefined but inflexible set of plot types.



Enough of one dimensional sequences but not enough of plotting sequences. We extend our experiments to two dimensions now. We are going to plot two dimensional sequences. Some of them give us such interesting plots that we can hardly believe what we see. First of all, let's describe briefly what is meant by a "two-dimensional" sequence. Consider for example the two sequences

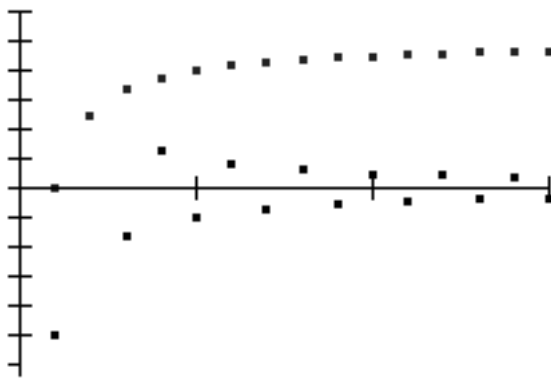
$$\frac{(-1)^n}{n} \{n \ 1\} \text{ and } \frac{n-1}{n} \{n \ 1\} \text{ simultaneously and}$$

think of the members  $\frac{(-1)^n}{n}$  and  $\frac{n-1}{n}$  as a pair. Many things can be done with this pair, like for example plotting them one over the other against  $n$ , or plotting them as a pair of coordinates and so on. We start with something similar to what we had in the complex numbers marathon. We plot two sequences against  $n$  like we did for the real and imaginary part of a complex quantity. To make an example, let's take the two sequences from above. Since they both start at  $n = 1$  and go to

# Sequences, series and limits with the HP49G - Part 4

$n = +$  , we can plot them together. Enter the list  $\left[ \frac{(-1)^n}{n}, \frac{n-1}{n} \right]$

which contains both general members of the sequences. Press STEQ. Many plot types can plot more than one expressions simultaneously when we put all expressions in a list and store this list in EQ. We are going to use again the plot type "Function" which behaves this way. Go to the PLOT SETUP screen, set "Type:" to Function, "Indep:" to  $n$ , check the option "\_Simoult", uncheck the option "\_Connect", set "H-Tick:" to 1, "V-Tick:" to 0.1 and uncheck the option "\_Pixels". Now, go to the PLOT WINDOW - FUNCTION screen and set "H-View:" from 0. to 15., "V-View:" from -1.2 to 1.1, "Indep Low:" to 1, "Step:" to 1 and uncheck the option "\_Pixels". ERASE and DRAW. The resulting plot contains both sequences. Nice, but you may already see the problem. If both sequences contain members near to each other then it will be hard to tell the one sequence from the other. If we connect the points (like in real function plots) then it gets a little bit better, but we lose the looks of the sequence and we can't see the actual points very well. One possible way to see which points belong to which sequence is to use the built-in capabilities of the HP49G in graphics. While you are in the plot environment, press the menu key [FCN] ([F4]). Press [NXT] to go to the second page of the menu. Press [+] to display the cursor coordinates and use the arrow keys to move the cursor to  $X=3$ . Press again [+] (or any menu key) to display the menu again and the press the menu key [NXEQ] ([F4]). The cursor moves to the upper chain of points and the sequence ' $(n-1)/n$ ' to which these points belong is displayed at the bottom of the screen. Press again [+] (or any menu key) to display the menu and press again [NXEQ]. Now the cursor goes down to the



sequence ' $(-1)^n/n$ ' and the sequence itself is displayed on the bottom of the screen. So we can see "where we are" at any time. Press any menu key to re-display the menu and then the menu key [PICT] ([F6]) to return to the plot environment. Press the menu key [TRACE] to start tracing. Unfortunately using the keys [ARROW-LEFT] and [ARROW-

RIGHT] while you trace the sequence  $\frac{(-1)^n}{n}$  has the undesired side effect of temporarily displaying the message "undefined" on the top of the screen, making the trace slow. This is because  $\frac{(-1)^n}{n}$  is only real

for integer values of  $n$ . There is a work around however. Press the menu key [(X,Y)] ([F2]) or [+] to display the cursor coordinates at the bottom of the screen. Now the message isn't displayed anymore, but instead of this the coordinate of Y remains "empty" for any  $n$  having non-integer values. You may also notice that the cursor doesn't "jump" on every point, but only to the points for  $n = 3$  ,  $n = 6$  ,  $n = 9$  ,  $n = 12$  . and  $n = 15$  . Why is that? Well, the horizontal coordinate of the cursor is calculated according to the current  $x_{\max}$  ,  $x_{\min}$  and the number of pixels available horizontally in the PICT, which is 131. With our current settings there is simply no pixel having horizontal coordinate 1, or 2 and so on. But there are pixels having horizontal coordinate 3, or 6 and so on. These are "jumped on" by the cursor

because the results  $\frac{(-1)^3}{n}$  ,  $\frac{(-1)^6}{n}$  , etc. are real numbers. Now, you

perhaps ask, how it comes that the other points were plotted? Well, for the plot the HP49G knew that it has to start at  $n=1$  and go to  $n=15$  in steps of 1. It calculated all pairs of *user* coordinates and then converted them to the pixels that are next to the calculated user coordinates. And this because we unchecked the option "\_Pixels" when we set up the step to 1. (If you check this option, then ERASE

and DRAW, you can see that almost no point of the sequence  $\frac{(-1)^n}{n}$  is plotted. The HP49G then finds first the horizontal coordinates of the pixels themselves, which under the current settings for  $x_{\max}$  ,  $x_{\min}$



# Sequences, series and limits with the HP49G - Part 4

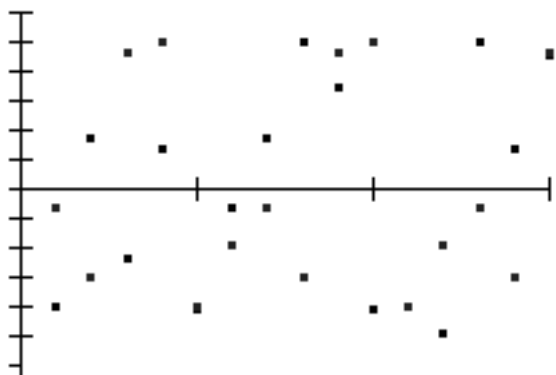
are almost never integer. So the expression  $\frac{(-1)^n}{n}$  is not real and so nothing is plotted.) Now let's change train and trace the other sequence. While in trace mode press [ARROW-UP] or [ARROW-DOWN]. While the keys [ARROW-LEFT] and [ARROW-RIGHT] move the cursor along some plotted curve while in trace mode, the keys [ARROW-UP] and [ARROW-DOWN] can be used to jump from one plotted curve to the other. The cursor goes up to the sequence  $\frac{n-1}{n}$ , which is always real for  $n \neq 0$ . While in PICT (even when in trace mode) you can press [BLUE-SHIFT] and then [ARROW-DOWN] to temporarily display the current expression on the top of the screen. This is an additional help to let you see which expression is currently traced.

Let's do another example now. Enter the list

$$\text{COS } \frac{n^2}{3} + \frac{3n}{5} - \frac{1}{7}$$

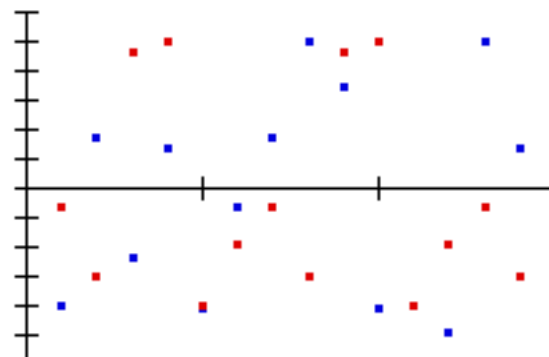
$$\text{SIN } \frac{n^2}{2} + \frac{4n}{6} - \frac{1}{8}$$

and press



STEQ. With the same settings as above ERASE and DRAW. On the resulting plot it is now really hard to keep the overview of the points of one sequence. As you can see, even if we had colours it would be hard to tell the behaviours of both sequences from each

other. The points of these sequences are too near to each other, they mix up in the plot, making it almost impossible to somehow follow the one or the other with a simple look. So we must somehow separate them but at the same time keep them on one plot if we want to compare them visually. We do the same thing like in the complex numbers



marathon. We are going to plot them one at the top half the other at the bottom half of the screen. Enter RCEQ to recall the list stored in EQ. Use the command OBJ-> to explode it and DROP to get rid of the element count. Store the expression

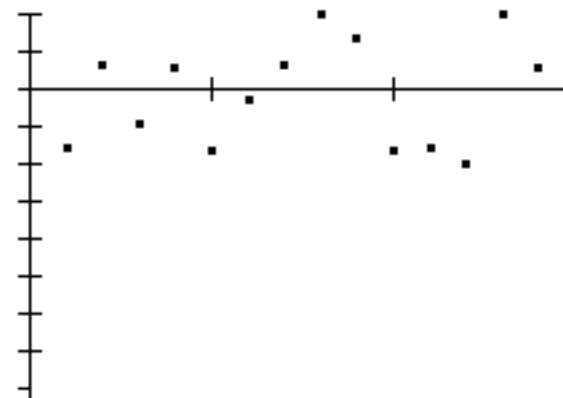
$$\text{SIN } \frac{n^2}{2} + \frac{4n}{6} - \frac{1}{8}$$

in 'TEMP' as we are going to use it later.

$$\text{Now use STEQ to store COS } \frac{n^2}{3} + \frac{3n}{5} - \frac{1}{7}$$

in EQ. We will

plot this sequence at the upper half of the screen, so we must set "V-View" from -4 to 1 to reserve space for the second plot. To avoid too many ticks on the vertical axes, we set "V-Tick:" to .5. Now, ERASE and DRAW to get the first half of the plot. Now recall TEMP and press STEQ to

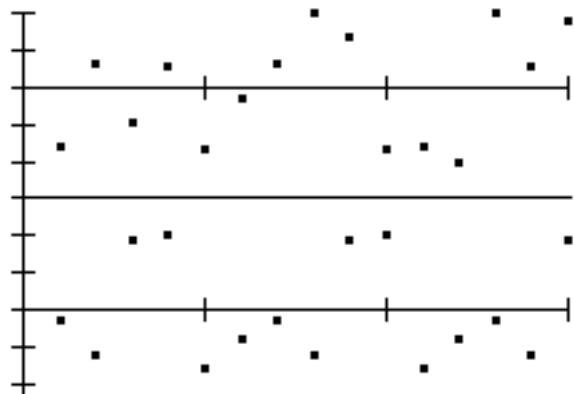


$$\text{store SIN } \frac{n^2}{2} + \frac{4n}{6} - \frac{1}{8}$$

in EQ. We want this expression to be

plotted under the first in the free space of the plot. So we set "V-View:" from -1 to 4. Press DRAW (*this time without ERASE*) to plot

# Sequences, series and limits with the HP49G - Part 4



the second expression. The plot now contains both sequences but separated from each other, so that we can see more clearly what is the one sequence and what is the other. We can add a dividing line between the two sequences for better separation.

While in the plot environment press [RED-SHIFT] and then [ARROW-LEFT]. This moves the cursor to the left of the display. Press the menu key [EDIT] ([F5]) to go to the menu with interactive commands for editing graphics. Press [×] (multiplication key). This puts a mark at the current position of the cursor. This mark is one of the two points that are necessary for drawing lines, circles and boxes. Press [RED-SHIFT] and then [ARROW-RIGHT] to move the cursor to the right. Now press the menu key [LINE] ([F3]) to draw a line between the mark and the current cursor position. The plot is now finished. (Except of course if you want to try your artistic capabilities by drawing boxes, circles and lines ;-)) We have now both sequences plotted against  $n$ . But we could also plot the one against the other. And as you can guess we do it, or else why should Nick say that? ;-)

Plotting the expression  $\text{SIN} \left( \frac{n^2}{2} + \frac{4n}{6} - \frac{1}{8} \right)$  against

$\text{COS} \left( \frac{n^2}{3} + \frac{3n}{5} - \frac{1}{7} \right)$  is essentially a parametric plot, with the

parameter  $n$ . The HP49G offers this plot type out of the box. The variable EQ contains then an algebraic expression or program that evaluates to a complex number. The real part is used as the horizontal coordinate while the imaginary part is used as the vertical coordinate.

That means that we must store in EQ the expression:

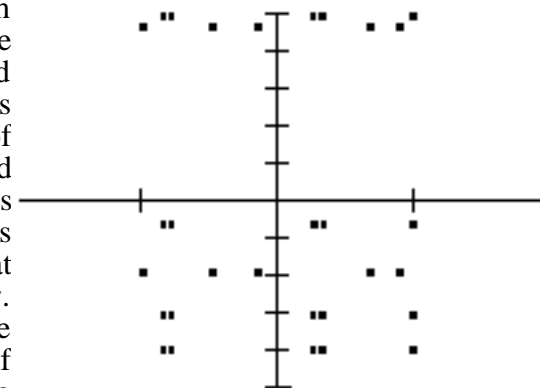
$$\text{COS} \left( \frac{n^2}{3} + \frac{3n}{5} - \frac{1}{7} \right) + i \text{SIN} \left( \frac{n^2}{2} + \frac{4n}{6} - \frac{1}{8} \right)$$

Do that, and then go to the PLOT SETUP screen and set "Type:" to Parametric. Set also "H-Tick:" and "V-Tick:" to 0.2. In the PLOT WINDOW - PARAMETRIC screen set "H-View:" from -2 to 2 and "V-View:" from -1 to 1. Since the dimensions of the PICT are 131x64 pixels, the width of the plot is twice its height. So the above settings for the horizontal and vertical view range preserve the aspect ratio. Set "Indep Low:" to 1 and "High:" to 50. Also set "Step:" to 1. Now ERASE and DRAW. You can see that the plot is interesting because it is... boring! Actually we would expect that many more different points would be plotted because the expression

$$\text{COS} \left( \frac{n^2}{3} + \frac{3n}{5} - \frac{1}{7} \right) + i \text{SIN} \left( \frac{n^2}{2} + \frac{4n}{6} - \frac{1}{8} \right)$$

was

plotted for 50 different values of  $n$ . But we have only about 30 plotted points. If you want, you can set "High:" to, say 100, so that the expression is evaluated and plotted for 100 different values of  $n$ . This will not change anything. (Except for the time that you spend waiting for the HP49G to finish the plot ;-)) The same points are plotted over and over again. The sequences have a finite number of points which are repeated periodically! This is perhaps interesting but it is nothing compared to what we are going to see now. Fasten your sit belts, the excursion to the depths of the universe is about to begin. (Trabakoulas says that we shouldn't forget our towels ;-))



# Sequences, series and limits with the HP49G - Part 4

We are going to plot the recurrence:

$$Y(n) = Y(n-1) + 0.7 \cdot X(n-1) + 0.7 \cdot Y(n-1) - \frac{(X(n-1) + 0.7 \cdot Y(n-1))^3}{25} \quad \{Y(0) = Y_0\}$$

against the recurrence  $\{X(n) = X(n-1) + 0.7 \cdot Y(n-1) \mid \{X(0) = X_0\}\}$ .

First we write a program that uses the global variable **P**, a complex number  $(X(n-1), Y(n-1))$ , to calculate the next point  $(X(n), Y(n))$ .

(We will see where the value in **P** comes from in a minute.) We use the fact that  $X(n-1) + 0.7 \cdot Y(n-1)$  appears more than once. We calculate this quantity only once and then we use stack commands to make copies of it which are used to complete the calculation.

```
<<
P RE .7 P IM * +      @calculate next X
P IM OVER DUP 3. ^    @calculate next Y
25. / - .7 * +
R->C DUP 'P' STO      @store a copy in P for the next
                      @calculation.
>>
```

Store that in EQ. Now, we use the same technique like in program RECINTERACT, to supply initial values for **P** interactively, that means out of the current coordinates of the graphics cursor.

```
<<
DEPTH ->LIST
-> stack
<<
ERASE DRAX
DO
  PICTURE
  IF
    DEPTH
  THEN
    'P' STO
```

```
DRAW 0.
ELSE
  stack OBJ->
  DROP 1.
END
UNTIL
END
>>
>>
```

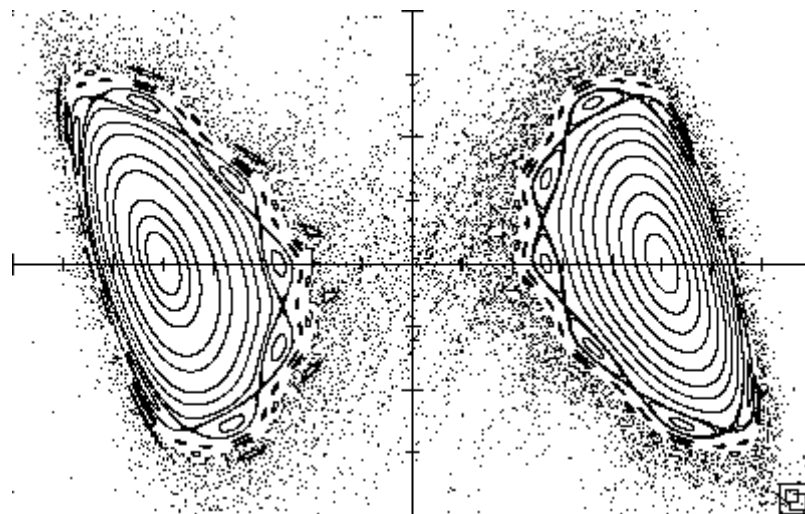
Store the program in REC2DINTRCT. Set parametric plot type, independent variable to **n** (or any other name - it is again a dummy), horizontal and vertical tick to 1 user unit, no connect, horizontal view from -8 to 8, vertical view from -4 to 4, independent variable low to 1, high to 500 and step to 1 user unit.

The check list is OK and we are ready for launch. Let REC2DINTRCT run. The first thing you see is the plot environment and only the axes are plotted. The program has just executed the command PICTURE and, as already mentioned, halts there until you leave the plot environment. While you are there you can move the cursor using the arrow keys, press **[+]** to show the cursor coordinates, and in general do anything that you can do when you manually enter the environment. Now, press **[+]** to see the coordinates and use **[ARROW-LEFT]** to move the cursor to **X=5, Y=0** (or as near to these coordinates as possible). Press the key **[ENTER]**. This puts the coordinates of the cursor on the stack as a complex number. It is exactly this complex number that is used as the initial value of the recurrences. Press **[ON]** to leave the plot environment and let the HP49G draw 500 points starting at the just entered point **(5,0)**. The sand clock appears and the HP49G seems to work but no much happens. (While the plotting is going on you can press **[ON]** to stop it, if you are not patient. This doesn't stop the whole program but only the plotting process.) When the HP49G is ready, be it because it plotted all 500 points or because you interrupted the plot, it shows again the plot environment. Press **[+]** to see the cursor coordinates, move the cursor to **X=4, Y=0** (or as near to this point as possible), press **[ENTER]** and then **[ON]**. Look! Some kind of loop materialises on the screen in front of your eyes. Let it

## Sequences, series and limits with the HP49G - Part 4

form nearly completely. Again, while the points are plotted you can press [ON] to jump to the plotting environment. Now that we know that starting at  $(5,0)$  almost doesn't plot anything and starting at  $(4,0)$  plots a loop, we wonder what might happen if we start at  $(4.5,0)$ . Move the cursor there, and again press [ENTER] and then [ON]. Another loop! It seems like we are orbiting around  $(5,0)$  if we start somewhere in the neighbourhood of  $(5,0)$ . But if we start at  $(5,0)$  we stay there. Move to  $(3.5,0)$ , press [ENTER] and then [ON]. Another orbit! Repeat starting at  $(3,0)$ . Wow! What is this? A chain-like orbit forms. Actually it is an orbit that consists of... orbits! Repeat starting at  $(2,0)$ . Watch the new orbit as it is being plotted. It starts like a "wavy" loop but after it has almost completely formed, the plotted points go to the left, keep moving for a while around a region which remains white, and then build-up something like space dust around the white region on the left and the "elliptical galaxy" on the right. We continue constructing the universe. Repeat starting at  $(1.5,0)$  and at  $(1.6,0)$ . The cosmic dust is getting denser. And are there new sub-galaxies forming at the outskirts of the "wavy boundary of the elliptic galaxy"? Go to the point  $(2.34, -1.71)$  and press again [ENTER] and [ON]. It really seems that the sub-galaxies exist. But let's move to the left, to the white region of the universe. Go to  $(-5,0)$  and draw another iteration. It looks as if  $(-5,0)$  were a point from where you can't escape. Initial points  $(-4.5,0)$  and  $(-4,0)$  form orbits.  $(-3.5,0)$  forms an orbit but very slowly.  $(-3,0)$  a somehow wavy orbit. But  $(-2.5,0)$  creates sub-galaxies at the boundary of the left galaxy. Wow, we have another galaxy on the left! Starting at  $(-2,0)$  puts a halo around the left galaxy and its sub-galaxies. And starting at  $(-1.5,0)$  makes cosmic dust even denser. You might have noticed that there are smaller white regions at the outskirts of the left galaxy. Go to  $(-1.6, -5.71E-1)$  and press [ENTER] and [ON]. Are these sub-sub-galaxies that form in the small white regions?

We already reach the limits of resolution of the HP49G screen at this scale. If the HP49G had better resolution we would see this:



But  $131 \times 64$  pixels are too few to see such details from such a distance. We must move nearer to the interesting part. If you want to keep this picture of the universe, just press [STO]. Doing this in the plot environment just puts a copy of the graph on stack level 1, which you can later store in some variable. But now, as Trabakoulas says in such cases: Zoom in, zoom in! Press the key [+] to see the cursor coordinates and move the cursor to  $(9.85E-1, 1.1.84)$ . Press again [+] to see the menu again. Press the menu key [ZOOM] ([F1]), and then the menu key [BOXZ] ([F2]). Press the key [+] to see the cursor coordinates again and move the cursor to  $(3.2, -6.98E-1)$ . Press [+] to switch to menu display and press the menu key [ZOOM] ([F6]). This will draw the horizontal axis and start the last iteration that we did again. Press [ON] to interrupt it. Go to  $(3,0)$  and again [ENTER], [ON]. You get a part of an orbit of the right galaxy. Start at  $(2.5,0)$ . Now you see sub-galaxies forming. Let them form well. Then go to  $(2.57, 6.72E-1)$  and let plot. A new wavy orbit forms just at the outskirts of the sub-galaxies. Start at  $(2.59, 6.72E-1)$ . A new orbit

# Sequences, series and limits with the HP49G - Part 4

seems to want to form between the sub-galaxies and the outskirts of the right galaxy. Start at  $(2.38, 6.32E-1)$  to get an orbit at the outskirts of the sub-galaxies, and at  $(2.26, 6.32E-1)$  to get new sub-sub-galaxies! Go to  $(2.69, 1.2)$  and  $(2.64, 1.16)$  and let plot. The inner world of the first sub-galaxies is formed. Start at  $(2.19, 6.32E-1)$  and  $(2.1775, 6.32E-1)$ . Again white wholes at the outskirts of the sub-sub-galaxies? Will they contain sub-sub-sub-galaxies? Let's see. Start at  $(2.14, 9.14E-1)$ . Yes, it looks like new sub-sub-sub-galaxies! Does it ever end? Start at  $(2.08, 5.92E-1)$ ,  $(2.02, 5.92E-1)$ ,  $(1.97, 5.92E-1)$  to decorate this part of the universe with glittering dust. Start at  $(1.75, 4.71E-1)$  and let plot. What? New galaxies in the middle of the cosmic void? And they somehow seem to be... triangles! You might be inspired enough to continue the construction of the universe, adding details and further zooming in to explore interesting regions. But the more you zoom in the more points you need to let the structures form well. And as the poor HP49G is not a CRAY you must be very very patient. On the next page you see what our zoomed part looks like and also some deeper zooms which are almost impossible on the HP49G. Isn't that wonderful? Two simple coupled recurrences construct a whole universe! And Trabakoulas wonders why in some zoom scales it looks like if the universe were... wooden! ;-)

And what happens if we change the recipe a little bit? Let's change the recurrence for Y to:

$$Y(n) = Y(n-1) + 0.5 \quad X(n-1) + 0.7 \quad Y(n-1) - \frac{(X(n-1) + 0.7 \quad Y(n-1))^3}{25} \quad \{Y(0) = Y0\}$$

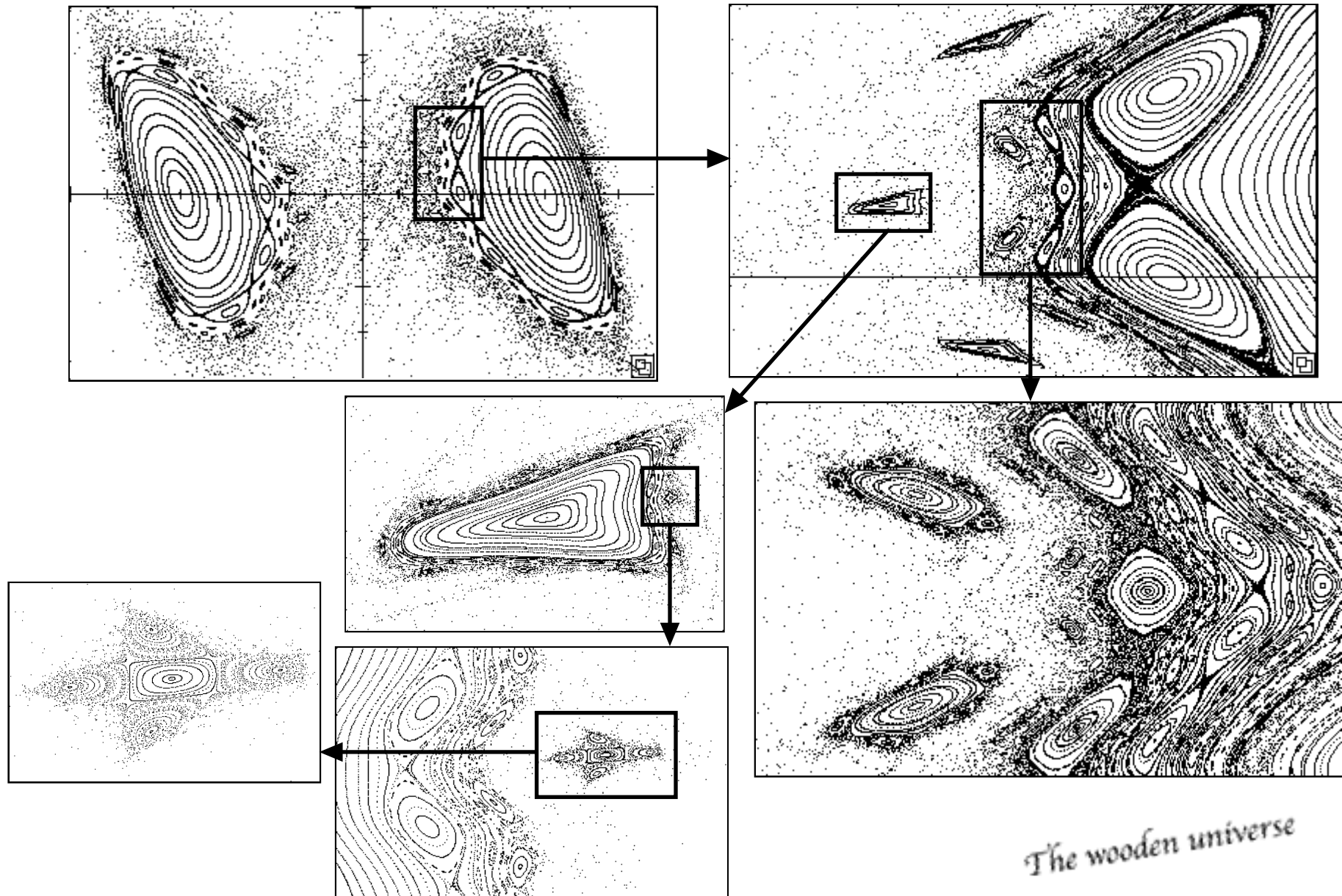
Change EQ to:

```
<<
P RE .7 P IM * +      @calculate next X
P IM OVER DUP 3. ^    @calculate next Y
25. / - .5 * +
R->C DUP 'P' STO      @store a copy in P for the next
                        @calculation.
>>
```

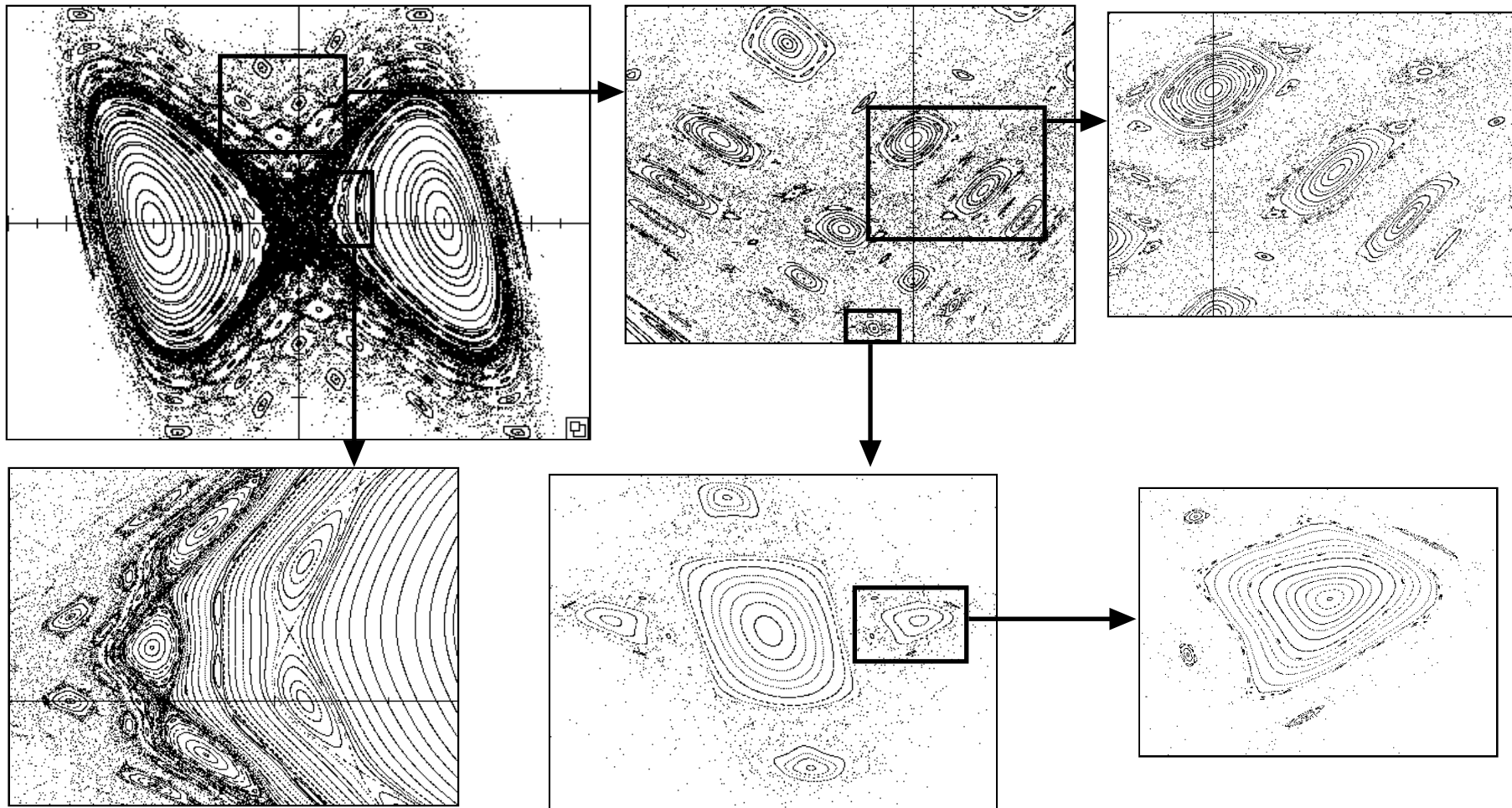
set the horizontal view range from -10 to 10 and the vertical from -5 to 5. Also set independent variable high to 2000. We need such a high number because some of the structures need many plotted points to form well. For the beginning try with starting values like  $(0,1)$ ,  $(0,2)$ ,  $(0,3)$ ,  $(0,3.5)$  and  $(0,4)$ . Remember that you can always press [ON] while the HP49G is plotting to interrupt the plot and continue with the next initial point. When you're done with the above initial points try  $(5,0)$ ,  $(4.5,0)$ ,  $(3,0)$ , and so on until  $(1,0)$ . When some initial point leaves some region more or less white, try starting somewhere inside that region. Quite often you get smaller structures that form in such regions. Zoom in interesting regions and look what new structures appear. The new recipe might be only just a bit different from the old, but the differences in the overall look of the plot are big. On the second next page you have again some pictures of the recurrences.

On the following pages we have some more universes and other strange products of such sequences. Storing the corresponding programs in EQ and running REC2DINTRCT you can explore them on the HP49G. (Until the batteries are dead, ;-)) Let it be said here that the following plots are made on a Mac and not on the HP49G. The plots on the HP49G will be less perfect.

## Sequences, series and limits with the HP49G - Part 4



## Sequences, series and limits with the HP49G - Part 4



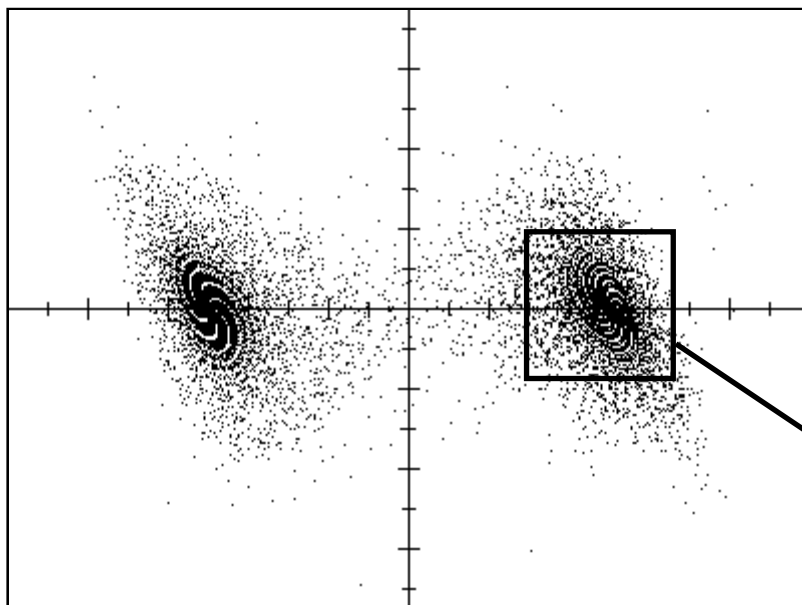
*Another wooden universe*

# Sequences, series and limits with the HP49G - Part 4

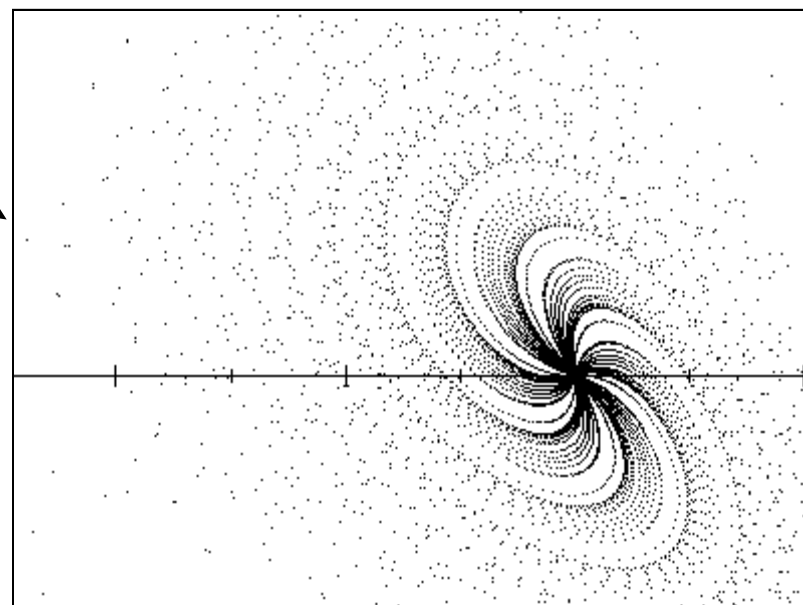
Sequences:

$$\{X(n) = X(n-1) + 0.01 \cdot Y(n-1)^3 \cdot \sin(3 \cdot X(n-1)) \quad \{X(0) = X_0\}\}$$

$$Y(n) = Y(n-1) + 0.71 \cdot X(n-1) + 0.7 \cdot Y(n-1) - \frac{(X(n-1) + 0.705 \cdot Y(n-1))^3}{25} \quad \{Y(0) = Y_0\}$$



*Spiral galaxies*



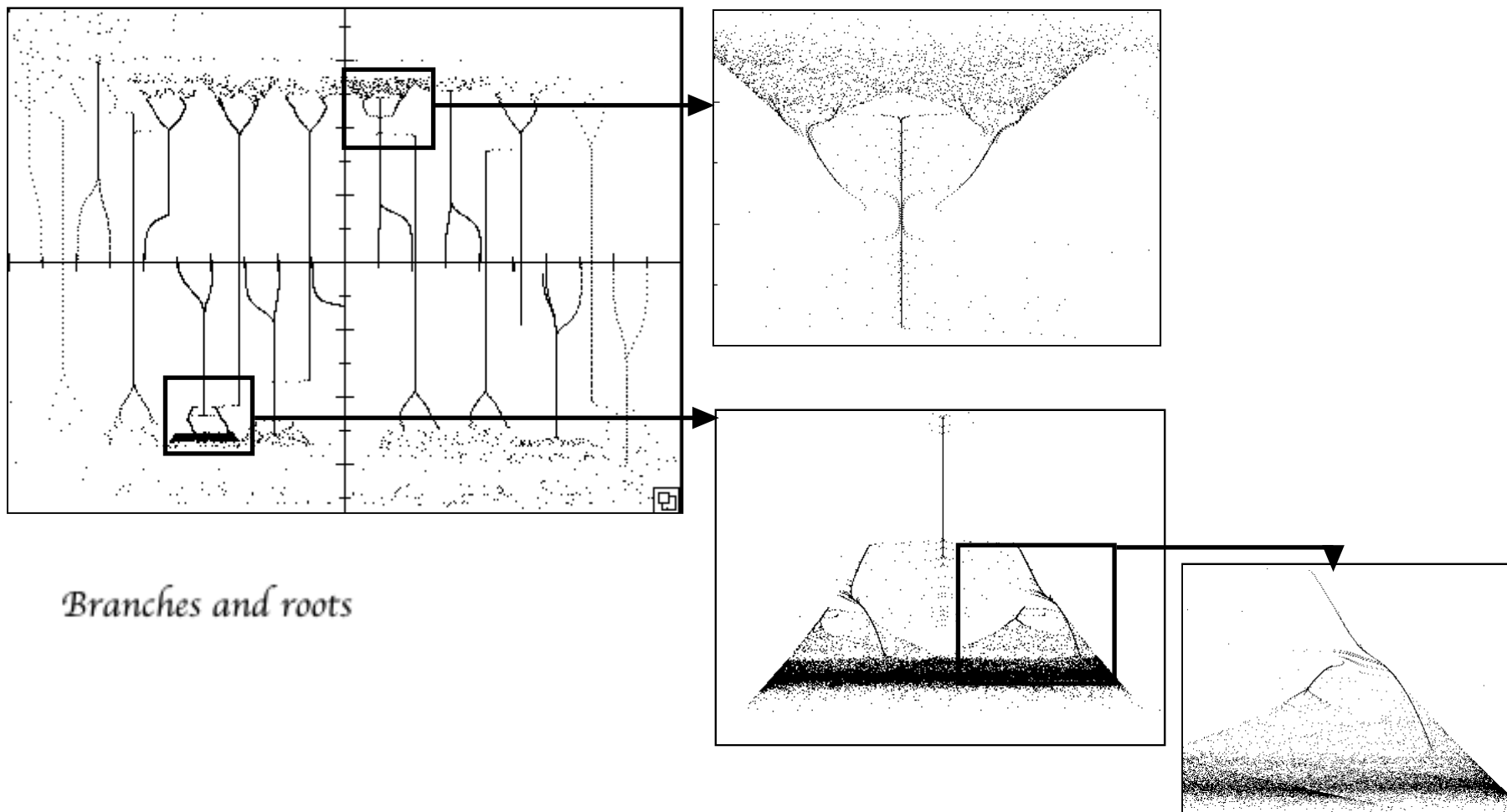


# Sequences, series and limits with the HP49G - Part 4

Sequences:

$$\{X(n) = X(n-1) + 0.02 \ Y(n-1)^2 \quad \{X(0) = X_0\}$$

$$Y(n) = Y(n-1) + 0.01 \ X(n-1) + 0.01 \ Y(n-1) - \frac{(X(n-1) + 0.09 \ Y(n-1))^3}{25} \quad \{Y(0) = Y_0\}$$



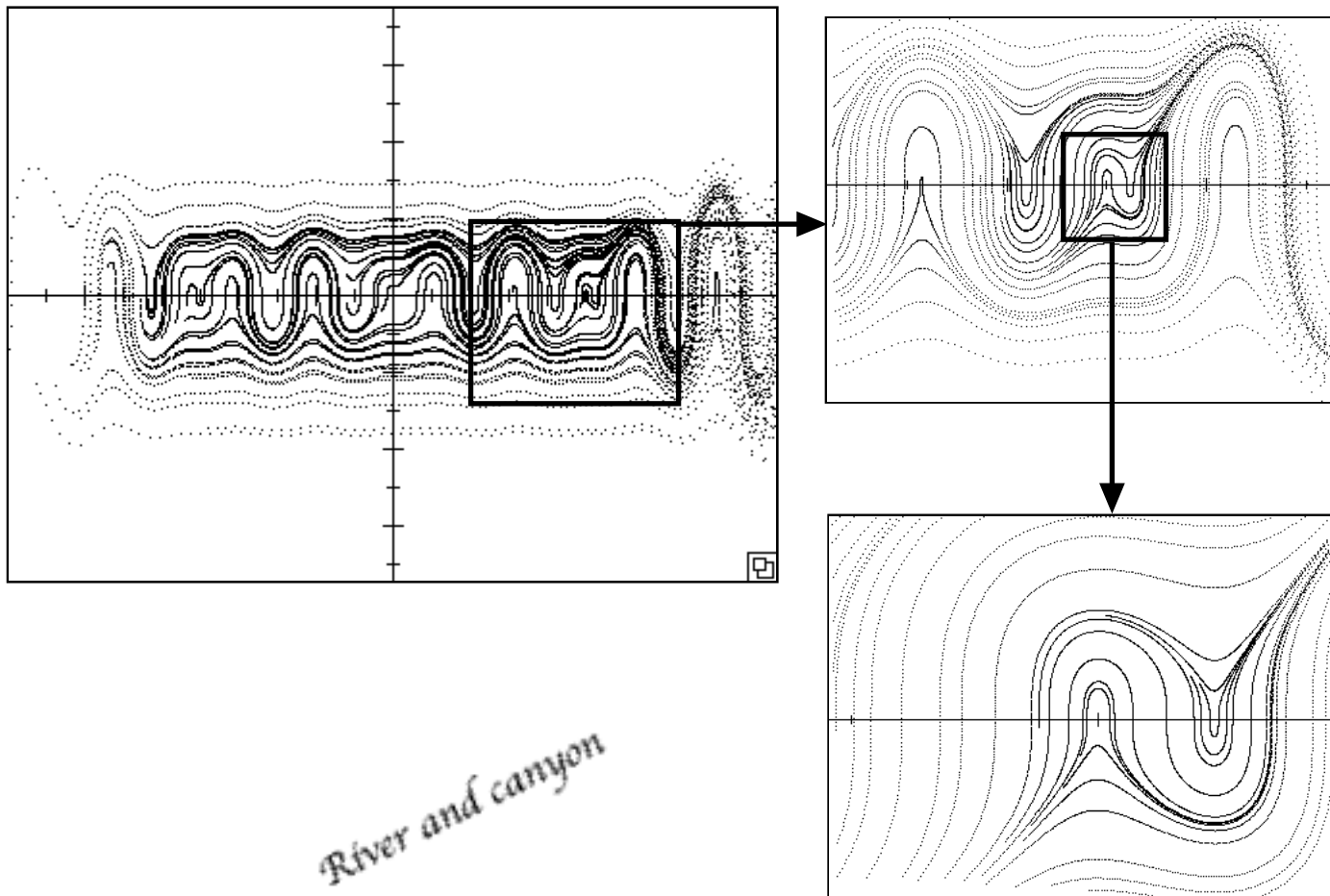
*Branches and roots*

# Sequences, series and limits with the HP49G - Part 4

Sequences:

$$\{X(n) = X(n-1) + 0.02 \quad Y(n-1)^2 \quad \{X(0) = X_0\}\}$$

$$Y(n) = Y(n-1) + 0.02 \quad X(n-1) + 0.07 \quad Y(n-1) - \frac{(X(n-1) + 0.02 \quad Y(n-1))^3}{25} \quad \text{SIN}(3 \quad X(n-1)) \quad \{Y(0) = Y_0\}$$

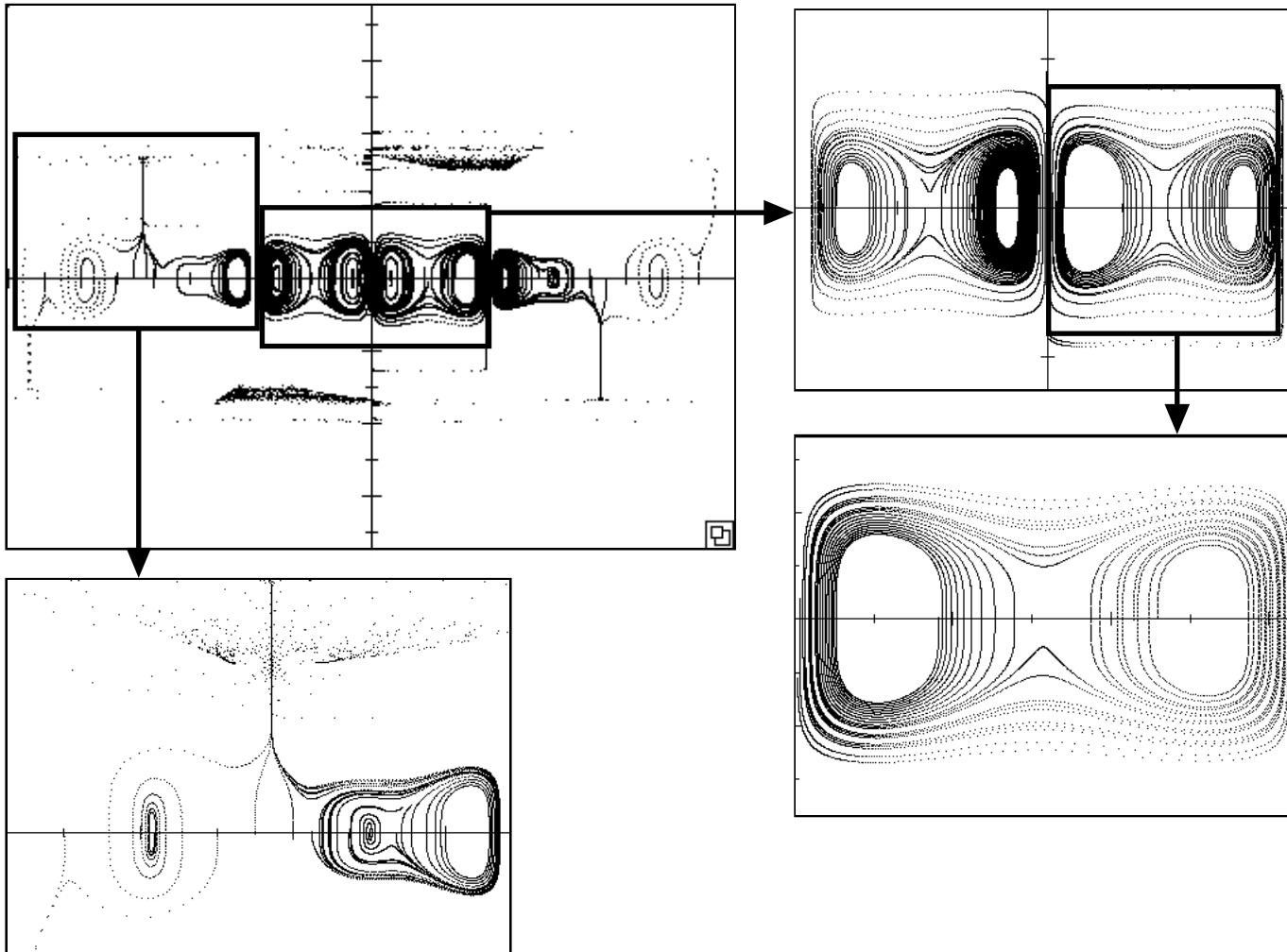


# Sequences, series and limits with the HP49G - Part 4

Sequences:

$$\{X(n) = X(n-1) + 0.03 \cdot Y(n-1)^3 \cdot X(n-1) \cdot \sin(X(n-1)) \mid \{X(0) = X_0\}\}$$

$$Y(n) = Y(n-1) + 0.01 \cdot X(n-1) + 0.01 \cdot Y(n-1) - \frac{(X(n-1) + 0.05 \cdot Y(n-1))^3}{25} \cdot \cos(3 \cdot X(n-1)) \mid \{Y(0) = Y_0\}$$



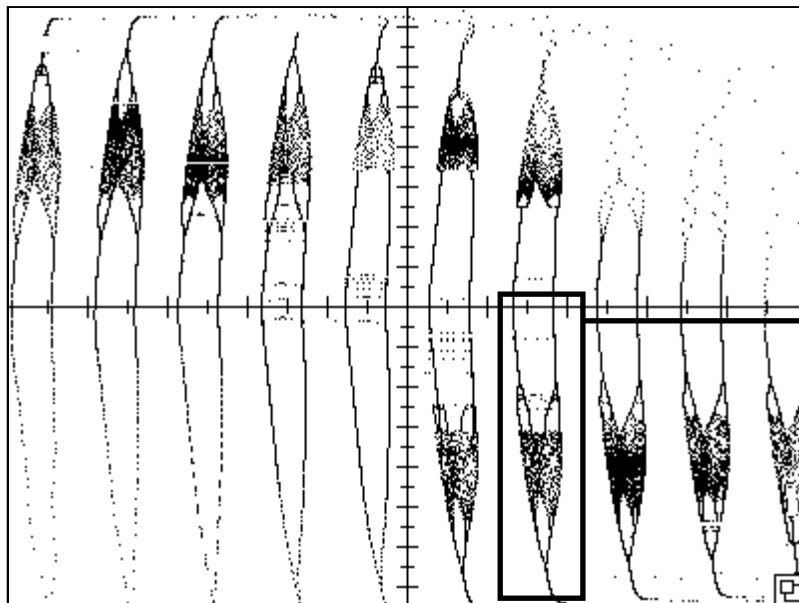
*Accelerator coils*

# Sequences, series and limits with the HP49G - Part 4

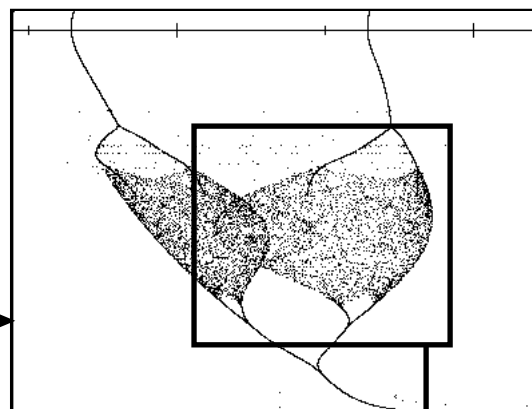
Sequences:

$$\{X(n) = X(n-1) + 0.07 \cdot Y(n-1)^2 \cdot \sin(3 \cdot X(n-1)) \mid \{X(0) = X_0\}\}$$

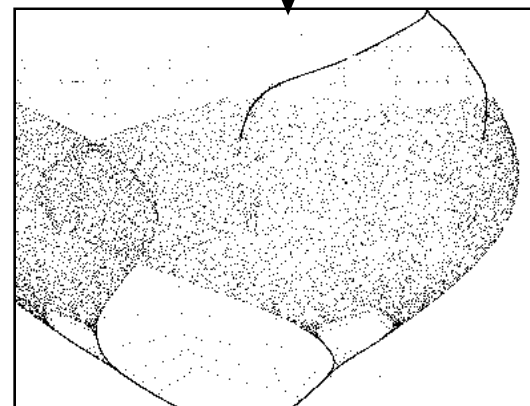
$$Y(n) = Y(n-1) + 0.07 \cdot X(n-1) + 0.7 \cdot Y(n-1) - \frac{(X(n-1) + 0.7 \cdot Y(n-1))^4}{325} \mid \{Y(0) = Y_0\}$$



*Mirrored women*



*Is that a bikini? ;-)*

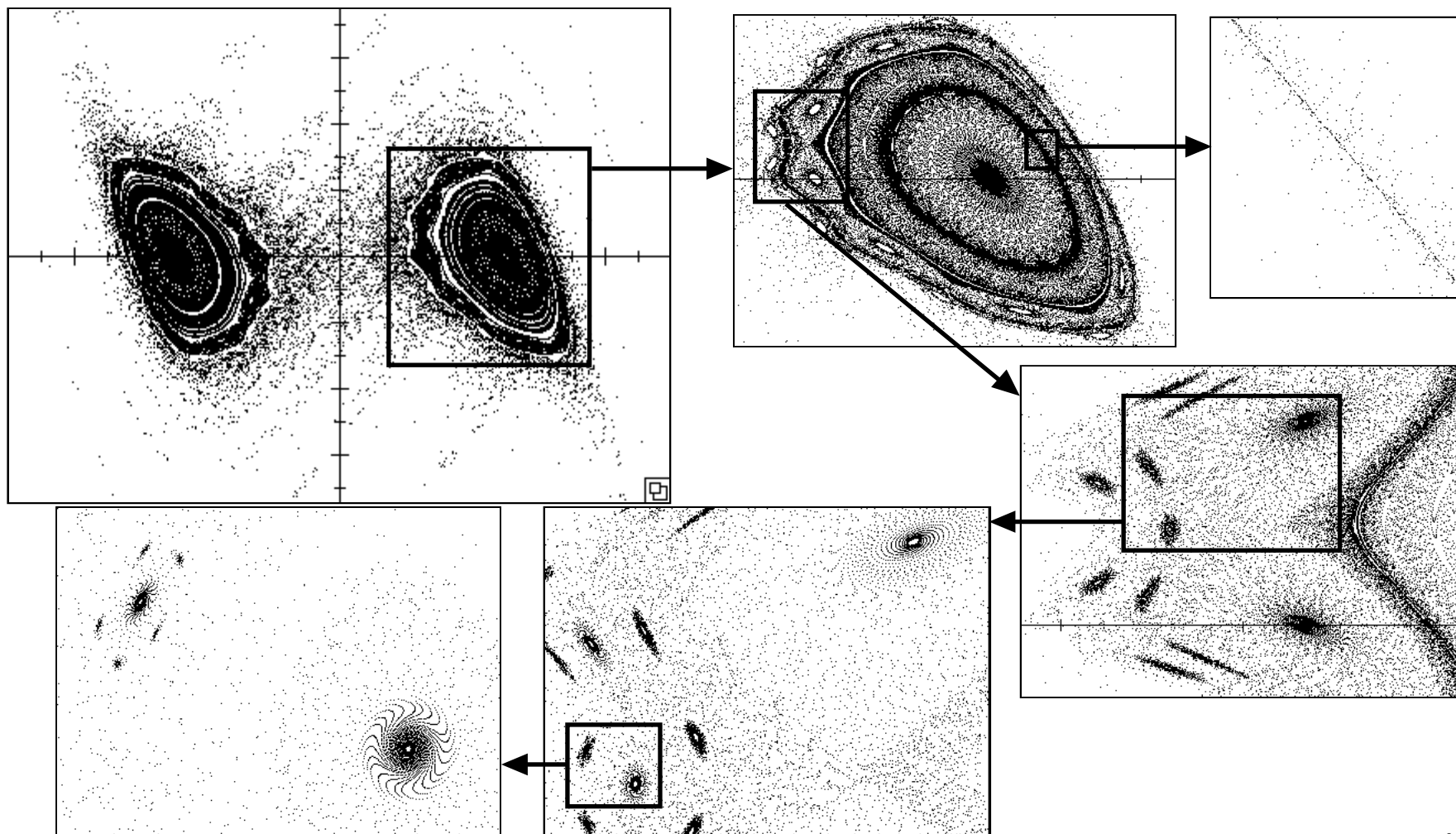


# Sequences, series and limits with the HP49G - Part 4

Sequences:

$$\{X(n) = X(n-1) + 0.7 \cdot Y(n-1) + 0.0001 \sin(2 \cdot X(n-1)) \quad \{X(0) = X_0\}$$

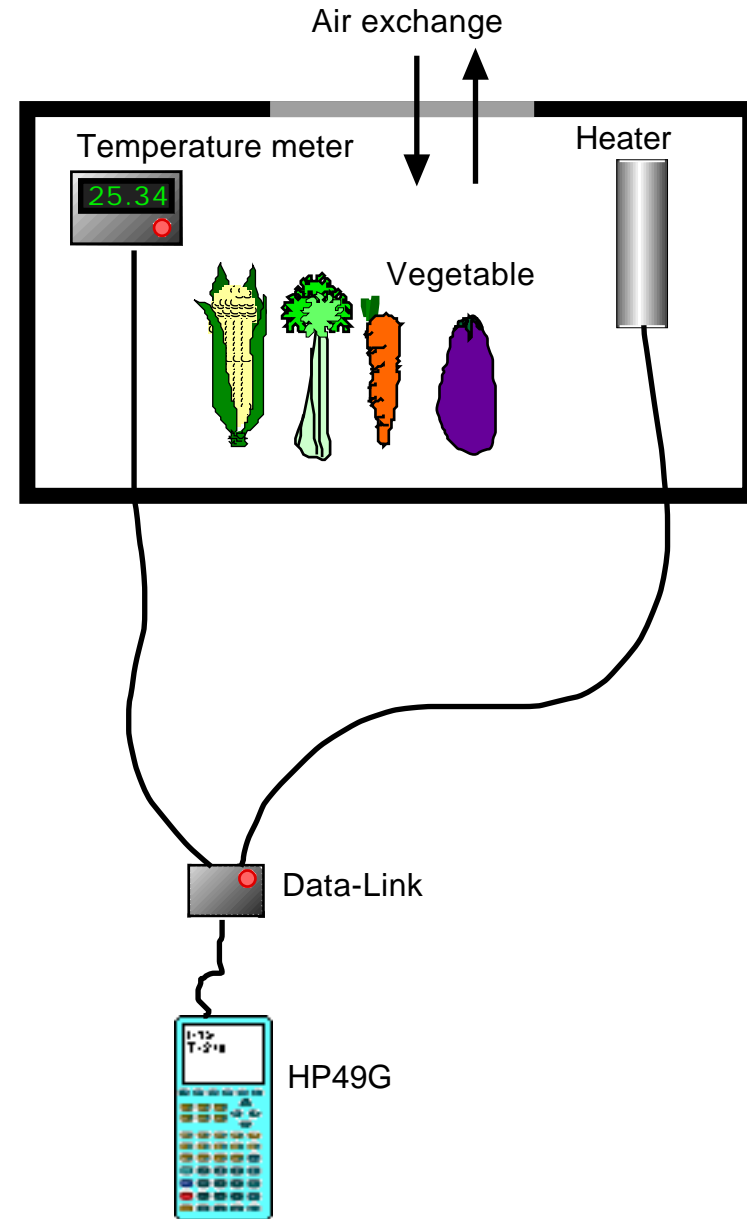
$$Y(n) = Y(n-1) + 0.7 \cdot X(n-1) + 0.7 \cdot Y(n-1) - \frac{(X(n-1) + 0.7 \cdot Y(n-1))^3}{25} + 0.001 \sin(3 \cdot X(n-1)) \quad \{Y(0) = Y_0\}$$



## Sequences, series and limits with the HP49G - Part 4

Wow! That was a journey to fascinating words! Perhaps you have already noticed that it was also a first look to chaos, which will be subject of a future marathon. Trabakoulas sits again and thinks about spiral distributions of goats and how they can be used in order to send them all to the centre point, the core ;-) Beside such highly scientific shepherd applications we have also other opportunities to apply what we have seen.

Take for example Nick and his vegetable. Once he asked himself "why do we have some vegetable in winter only, while some other vegetable comes only in summer?". He asked his wife this highly philosophical question and expected some kind of international conspiracy that for some secret reason forbids spring onions to be sold in winter. But he was quite amazed to hear that different plants grow at different times and that one couldn't expect to see ripe bananas growing on the Alps in the middle of winter. Disappointed from the unexpected simplicity of the answer (what? no conspiracy?), he decided to construct a small vegetable house isolated from its surroundings with only one air exchange window. The inner atmospheric conditions, like temperature, humidity and so on, should be controlled by the HP49G. He planned first the temperature control. The HP49G is connected to a data link, model Rcobo-DeLuxe. This device receives temperature readings from a temperature meter in the vegetable house every 10 seconds, and sends them to the HP49G. On the HP49G a program is running which compares the received temperature with a value that Nick has entered, 20°C. If the temperature inside the vegetable house is less than 20°C then the HP49G sends the command "Heat" to the link, which then, intelligent as it is, sends the command to the heater. The heater starts heating for a second and then stops heating, giving enough time for the air inside the house to be homogenised, so that the next temperature reading really measures the temperature of the whole house and not only at the spatial coordinates of the temperature meter. If the temperature is over 20°C, then the HP49G doesn't send any command and the air exchange does the cooling. So he thought that he could establish a constant temperature in the vegetable house, so that he could grow and sell swiss bananas in winter and greek spring onions in summer. Trabakoulas doubt that this was going to be that easy, but Nick nonetheless wanted to try it. The results were not at all



# Sequences, series and limits with the HP49G - Part 4

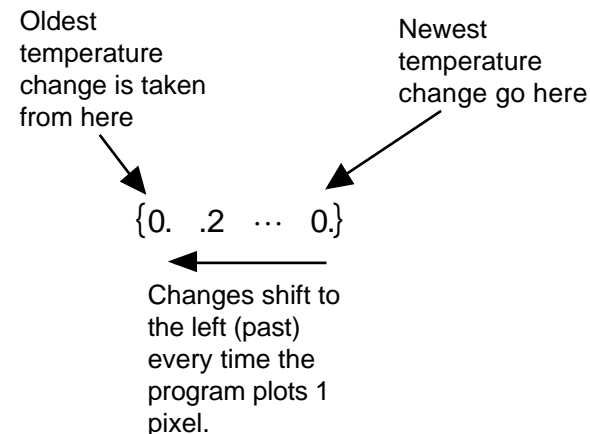
convincing. The bananas look rather like french fries and the spring onions were stinking so strongly that one could only get in the vegetable house wearing a diving mask with aqua lung. What went wrong?

We simulate the above situation on the HP49G with a program.

```
<<
{#0d #0d} PVIEW      @Show us the current PICT
DRAX
{0. 0. 0. 0. 0.
0. 0. 0. 0. 0.}      @List of past temp. changes
-1.                  @Counter (time)
-> Current Target h t @Store in locals
<<
DO
' h'
IF                  @If current temp. less
Current Target. < @target temp.
THEN
.4                  @return temp. change because of heater
ELSE
0.                  @else no heating.
END
STO+                @Add to list of past temp. changes
Current .2 -        @Temp. lowering air exchange
h HEAD +            @Add temp. raise from before 10 sec.
' Current' STO
h TAIL
' h' STO            @Store rest of past temp. changes.
't' INCR            @Increment time
Current R->C PIXON @Turn correspond. pixel on.
UNTIL
t 130 ==            @Until we plot at t=131 seconds
END
PICTURE             @We go to the plot environment
>>
>>
```

This simulation bases on the fact that the heat needs 10 seconds to be distributed homogeneously in the house by air circulation. That is, when the heater heats now, the temperature meter will read the increased temperature because of this heating after 10 seconds. The heater gives the amount of heat to the air in the house, that is enough for a temperature raise of  $0.4^{\circ}\text{C}$ . At the same time, if we heat or not, the air exchange with the outside is always present and lowers the temperature of the house at the rate of  $0.2^{\circ}\text{C}$  per second. (It's cold out there ;-))

Store the program in 'NICKOVEG1' and set up a horizontal plot view range from 0 to 130 and a vertical view from 10 to 25. Because the PICT is 131 pixels wide, setting H-View from 0 to 131 (or any other range that contains 131 user units) actually makes 1 pixel equal to one user unit. The PICT isn't restricted to  $131 \times 64$  pixels. It can be much bigger. The portion that you see at a time however is always  $131 \times 64$  pixels. We are going to see how we can change the dimensions of PICT in the next part of this marathon.

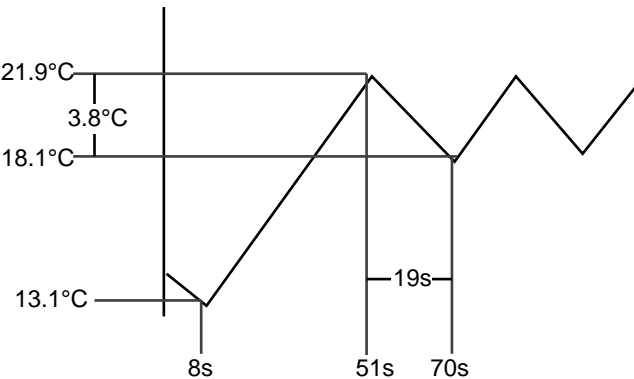


The list h plays the role of the "history of temperature changes due to heating". We always take the first (oldest) change and add it to the current temperature and at the same time we add the newest temperature change at the end of the list.

The program needs the current temperature and the target temperature, so enter 15. and 20., ERASE and run NICKOVEG1. The program starts plotting temperature against time. When it finishes the plot, it switches to the plot environment, to let us examine how the temperature changed from 0. to 130 seconds.

# Sequences, series and limits with the HP49G - Part 4

We see that the first 8 seconds the temperature falls until it reaches 13.1°C. Then it rises until it reaches 21.9°C at 51 seconds. And from this time on it oscillates between 21.9°C and 18.1°C every 19 seconds.



The poor vegetable never knew what kind of climate that was. ;-)

One reason for the bad controlling of temperature is that the heat used to make the house warmer is too much when the current temperature approaches the target temperature, because the thermometer reading lags 10 seconds. So perhaps we could improve the circumstances under the vegetable has to grow, if we take a better heater that doesn't always give a fixed amount of heat. For example we could take the model TH1 by TRABA-HEAT® which releases an amount of heat that can be adjusted, so that it that causes a temperature change of  $\frac{\text{target} - \text{current}}{10}$ .

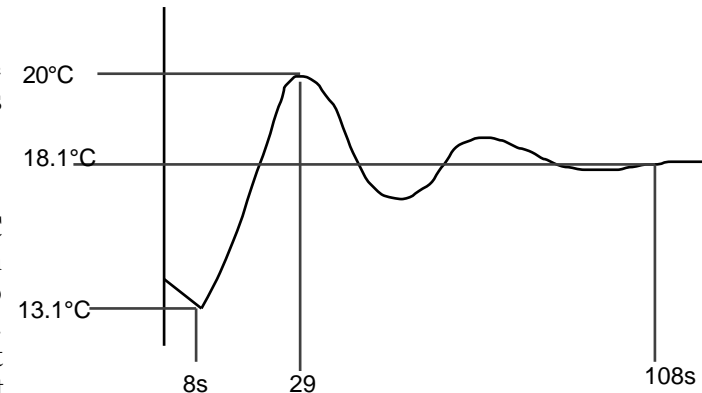
10

The program for the simulation must now be slightly changed:

```
.....
IF                               @If current temp. less
  Current Target. < @target temp.
THEN
  Target Current - @return temp. change
  10. / @because of heat
ELSE
  0. @else no heating.
END
.....
```

The changed code is in red. Store this in NICKOVEG2, enter again 15. and 20. and run it.

Now the behaviour is quite different. The temperature falls to 13.1°C at 8s but then rises up to 20°C at 29s. Then it oscillates but the oscillations



get weaker and weaker and from about 108s on it remains constant at 18.1°C. Better than before because the temperature remains constant after some time, but still not very good because we wanted 20°C and we stabilise temperature at 18.1°C. What would happen if we provide a target temperature of 22°C? Enter 15. and 22. and run NICKOVEG2 again. Yeah! Now, after all oscillations the temperature stays at 20°C. So we can adjust a target temperature of 22°C where we want 20°C. Or improve the system once more.

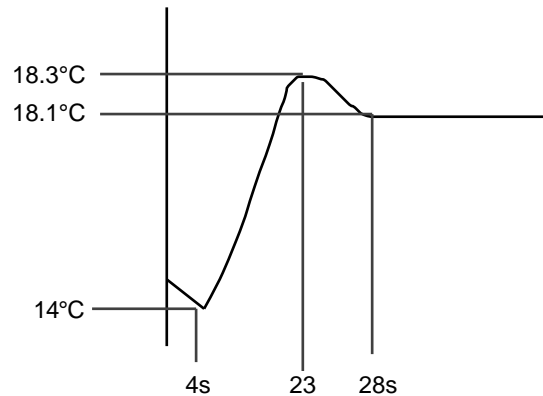
Another weakness of the whole system is that the heat simply needs to much time to be uniformly distributed around. We must somehow accelerate that. What about a ventilator? Let's install model KAROT-1 from Karagiaouoglou Rotors® and see what happens. The heat needs now 5. seconds for uniform distribution, so we change NICKOVEG2:

```
.....
DRAX
{0. 0. 0. 0. 0} @List of past temp. changes
-1. @Counter (time)
.....
```

making the history of temperature changes shorter. Store the new program in NICKOVEG3. Enter 15. 20. and run the new program.



## Sequences, series and limits with the HP49G - Part 4

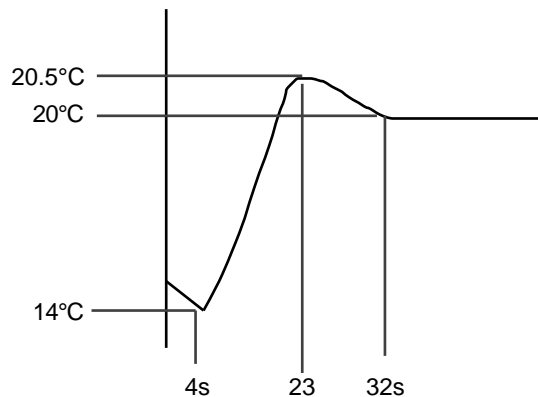


Now the results are better considering that constant temperature is achieved much faster and with almost no oscillations. But again we reach 18.1°C and not 20°C. So we adjust again a target temperature of 22°C and see what happens. Enter again 15. and 22. and run NICKOVEG3 again.

hope that the consequences of travelling to wooden universes were not very dramatic and that you don't decide to become a vegetable farmer ;-)

Vegi-Greetings,  
Nick.

That was it! After 32s we reach 20°C. Theoretically at least. The used model for simulation is too naive for accurate predictions, and I tell you the function `VegetableGrowth( )` can be very demanding ;-). Not to speak about all the other variables that are important, like humidity and the like. And not to speak about perturbations like for example Nick who forgot to shut the door of the vegetable house in the middle of winter, or the goats of Trabakoulas that ate up anything eatable in the house ;-).



That's why Nick thought again and decided not to become a vegetable farmer but an HP49G enthusiast instead. At least the HP49G never complained "Sequence Error: Too Low Temperature", though rumour has that the JYA wanted to include this feature ;-).

End of this part. Next time we'll continue this marathon with series and we'll see what the HP49G provides for working with them. I

# Sequences, series and limits with the HP49G - Part 5

Zeno the sophist was a rather seldom kind of human being. He liked thoughts that produce contradiction. And he liked being the enfant terrible of the deic establishment of ancient Greece. One day, he was sitting with his friend Trabakon<sup>6</sup> the shepherd at the sea side tavern drinking his ouzo, he found the situation quite boring. No problems what so ever, everything peaceful and quite. But the Trabakon noticed that a few tables away, Herakles the Hero the son of Zeus the God was sitting and enjoying a good meal. Trabakon told Zeno about the 12 heroic missions of Heracles and that was it! Zeno had just found a new victim. Smiling with singing voice he approached Herakles.

"Ahey! Son of Zeus, godly Herakles, a good appetite I wish you."

"Join me, oh wise Zeno, the man who brings us headaches!"

"I wish I could, oh Heracles, but studies I must do<sup>7</sup>. But before I go, a tiny thing that I must ask, oh hero of the 12 missions."

"Go ahead and ask wise man, and I'll be glad to answer."

"Can you the strongest of the Greeks run faster than Mara the turtle, if it starts 1 stadion ahead of you?"

"Wise man, of course I can, and I am going to prove that by just doing."

"Be careful, oh Herakles, be careful I tell you. This

<sup>6</sup> Ancestor of Trabakoulas

<sup>7</sup> Zeno spoke RPL sometimes

mission is more difficult than all the others taken together."

"Wise man, explain to me."

"Suppose the turtle runs 12 times more slowly than you, oh Heracles. Then, at the time you have just run to the point where she started, covering the starting distance to her, she is already exactly  $\frac{1}{12}$  of the same distance ahead."

"Yes, oh wise man, but then I'll reach her!"

"Not even then, oh Heracles! Because while you are covering that  $\frac{1}{12}$  of the starting distance, she again runs  $\frac{1}{12}$  of  $\frac{1}{12}$  of the same distance."



# Sequences, series and limits with the HP49G - Part 5

"Yes, but then...."

Herakles stopped and stood up. He understood the problem. He really put Mara one stadion ahead and started running, but the experiment didn't work because Mara had better things to do than running 12 times more slowly than Herakles. The implications of this event were enormous. Herakles went to his angry father, Zeus the God and told him about the story. Zeus asked in agony:

"And what did you find out experimentally my son?"

"Errhh, I mean.... Mara wouldn't run..., you know..."

"We sent you to the best hero schools, oh stupid, and you just neither know, nor you can find out?"

"Sir, I..."

"Shut up! The punishment terrible will be<sup>8</sup>. You will never be a God, you stay half a God for ever."

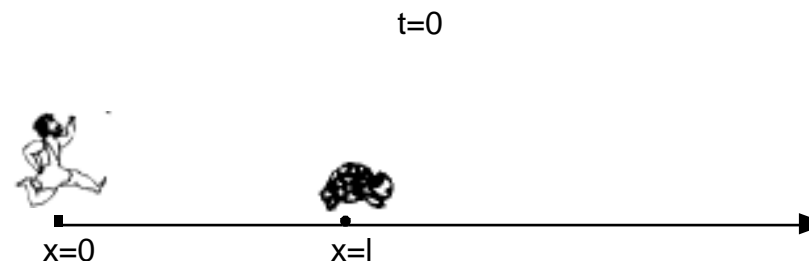
"Because I didn't run faster, Sir?"

"Of course not, you stupid! But rather because instead of sitting at the tavern, eating Souvlaki and Tzatziki and drinking ouzo, you preferred to let Zeno make a fool out of you!" Saying this Zeus blinked with the eyes, as he always did when he wanted to say that in reality the punishment was no punishment at all. Herakles didn't get that immediately but he did understand after some years, when he compared again his life with the life of Gods. Night after night at the sea side tavern, enjoying terrific meals and the presence of the most beautiful girls, instead of the monotonous nectar and ambrosia of the Gods. Being a fool has also advantages sometimes ;-)

Despite the big life that Herakles makes, let's try to prove Zeno wrong. We do it using simple physics first, and later we'll examine it also using series maths. Let the speed of Mara be  $v$ . If Heracles runs

<sup>8</sup> Zeus also spoke RPL sometimes

12 times faster than Mara, then his speed is  $12v$ . Let's further put the point 0 of the x-coordinate at the starting point of Heracles. Mara starts



at  $x=l$ ,  $l$  being in this particular case the length of 1 antique greek stadion about 185m. At  $t=0$  we have the situation as demonstrated on the above picture taken with a Trabon X1000 camera. Now, Zeno fires a signal and watches both athletes running. To his embarrassment Heracles does reach Mara. Let's denote the time when Heracles reaches Mara with  $t_1$ . At any time  $t$  the x-coordinate of Mara is  $l + vt$ . At the time  $t_1$ , that is when they meet, her x-coordinate is  $l + vt_1$ . Zeno enters this on his HP49G. The x-coordinate of Heracles at any time is  $12vt$ . When they meet his x-coordinate is  $12vt_1$ . Zeno enters this expression on his HP49G again. Because they meet both x-coordinates must be equal. Zeno presses [=] and gets the equation  $l + vt_1 = 12vt_1$ . Now he enters  $t_1$  and he presses [SOLVE]. His

HP49G returns  $t_1 = \frac{l}{11v}$ . This is the time when Heracles reaches

Mara. Zeno enters again the x-coordinate of Mara at the time  $t_1$  which is  $l + vt_1$ . He presses [OVER] to get a copy of  $t_1 = \frac{l}{11v}$  at stack

level 1 and then [SUBST], [EXPAND]. The result is  $\frac{12l}{11}$ , the x-

coordinate of Mara when Heracles reaches her. Very nervous, Zeno re-enters  $12vt_1$ , the x-coordinate of Heracles at  $t_1$ . He presses [ARROW-UP] to go to the interactive stack, and then presses again twice [ARROW-UP] to go to stack level 3. Then he presses the menu

# Sequences, series and limits with the HP49G - Part 5

key [ECHO] ([F1]) to take  $t_1 = \frac{l}{11 \cdot v}$  in the command line, and then he presses [ENTER] to leave the interactive stack. Now his command line contains ' $t_1 = l/(11 \cdot v)$ '. He presses [ENTER], [SUBST], [EXPAND] and gets the x-coordinate of Heracles  $\frac{12 \cdot l}{11}$  at the time  $t_1$ . The same x-coordinate as Mara's! Zeno turns red, he fumes, throws

$$t = t_1 = \frac{l}{11 \cdot v}$$



his HP49G to the ground and decides to become a vegetable farmer without HP49G. ;-) The moral of the story: Never mess up with a greek Hero when he sits at the tavern ;-)

But why did Zeno such a mistake? What was his error? Well, if we consider the expression that gives the x-coordinate of Mara, as he considered it, we have:

$$x_{\text{Mara}} = l + \frac{l}{12} + \frac{l}{12} + \frac{l}{12} + \dots = l + \frac{l}{12} + \frac{l}{12^2} + \frac{l}{12^3} + \dots = l \left( 1 + \frac{1}{12} + \frac{1}{12^2} + \frac{1}{12^3} + \dots \right)$$

with  $l$  the length of one stadion. The expression in the parenthesis is a

series for which Zeno believed that it can grow above any limit. This made him think that Heracles and Mara will meet at  $x = +\infty$ , that is never. He thought that adding  $\frac{1}{12^n}$  with  $n$  going from 0 to  $+\infty$  will give a number that grows and grows above any finite number. But adding something an infinite number of times and infinity itself are two different pairs of shoes in this case. As we know today, the sum  $1 + \frac{1}{12} + \frac{1}{12^2} + \frac{1}{12^3} + \dots = \sum_{n=0}^{+\infty} \frac{1}{12^n}$  converges to  $\frac{12}{11}$ . Let's see if the HP49G knows that. Enter  $\sum_{n=0}^{+\infty} \frac{1}{12^n}$  and press [EXPAND]. Voila! You get  $\frac{12}{11}$ . This sum is but an example of an infinite series (or briefly

series). In general we write for a series:  $a_0 + a_1 + a_2 + \dots = \sum_{n=0}^{+\infty} a_n$ .

The statements that we have for sequences are very useful when we examine series. Beginning with a series, we can construct the sequence

of its partial sums. We construct all *finite* sums,  $s_0 = \sum_{n=0}^0 a_n$ ,

Partial sums sequence of a series  $\sum_{n=0}^{+\infty} a_n$

$$\begin{aligned} s_0 &= a_0 = \sum_{n=0}^0 a_n \\ s_1 &= a_0 + a_1 = \sum_{n=0}^1 a_n \\ s_2 &= a_0 + a_1 + a_2 = \sum_{n=0}^2 a_n \\ &\dots \\ s_N &= a_0 + a_1 + a_2 + \dots + a_N = \sum_{n=0}^N a_n \end{aligned}$$

$$s_1 = \sum_{n=0}^1 a_n, s_2 = \sum_{n=0}^2 a_n,$$

and so on and we get a sequence  $s_0, s_1, s_2, \dots$ . If this sequence of partial sums converges, that is iff it has a limit, then the series also converges and has the same limit. Let's take for example

Zeno's series  $\sum_{n=0}^{+\infty} \frac{1}{12^n}$ .

We construct the

# Sequences, series and limits with the HP49G - Part 5

sequence of partial sums. It is easy to write some certain members of this sequence, but what does the general  $N^{\text{th}}$  member looks like? We can use the HP49G to find that. In real mode enter  $\sum_{n=0}^N \frac{1}{12^n}$  and press

[EXPAND]. The HP49G says:  $\frac{12}{11} \frac{12^{N+1} - 1}{12^{N+1}}$ . EXPAND again to get  $\frac{12}{11} \frac{12^N - 1}{12^N}$ . That is, if we use the notation that we used until now for sequences, we can enter the partial sums sequence of Zeno's series as  $\sum_{n=0}^N \frac{1}{12^n} \{N \ 0 \ + \}$  or  $\frac{12}{11} \frac{12^N - 1}{12^N} \{N \ 0 \ + \}$ . The HP49G can find the limit of this sequence in both forms. Enter  $\sum_{n=0}^N \frac{1}{12^n}$ , then  $N = +$ , and then press [lim]. After some seconds the

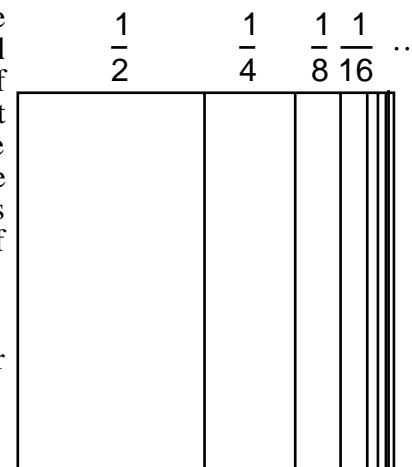
HP49G returns  $\frac{12}{11}$ . For the other form, enter  $\frac{12}{11} \frac{12^N - 1}{12^N}$ , then  $N = +$  and then press [lim]. Again the result is  $\frac{12}{11}$ . Notice that when

Partial sums sequence of Zeno's series $\sum_{n=0}^+ \frac{1}{12^n}$	
$s_0 = 1 =$	$\sum_{n=0}^0 \frac{1}{12^n}$
$s_1 = 1 + \frac{1}{12} =$	$\sum_{n=0}^1 \frac{1}{12^n}$
$s_2 = 1 + \frac{1}{12} + \frac{1}{12^2} =$	$\sum_{n=0}^2 \frac{1}{12^n}$
...	...
$s_N = a_0 + a_1 + a_2 + \dots + a_N =$	$\sum_{n=0}^N \frac{1}{12^n} = \frac{12}{11} \frac{12^N - 1}{12^N}$

finding the limit using the second form, the HP49G switches the current VX to N. So if your VX is something else, use STOVX to reset it to what you had before finding the limit. The HP49G is able to find many limits of series out of the box. That means, we don't need to program much. Hurrah! ;-)

Let's have another example. We take a square and divide it in two equal parts. Then we divide the right half in two equal parts, again the right half in two equal parts and so on. The picture on the right suggests that the sum of all parts has to be 1. This sum is an infinite sum, a series, of the form  $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = \sum_{n=1}^+ \frac{1}{2^n}$ .

Let's see if it converges. Enter  $\sum_{n=1}^+ \frac{1}{2^n}$  and press [EXPAND]. The



result is 1. Again, let's find the general member of the partial sums sequence of this series. Enter  $\sum_{n=1}^N \frac{1}{2^n}$  and expand twice. The result  $\frac{2^N - 1}{2^N}$  is the general member of

the partial sums sequence  $\frac{2^N - 1}{2^N} \{N \ 1 \ + \}$ . Of course, since

the HP49G can find the limit of this series, it isn't necessary to find the general member the partial sum sequence and then find its limit for  $N = +$ . But it might be useful for other cases to have a program that takes such a series and returns its partial sum sequence.

```
<<
->LST -> seqlst      @Turn to RPL List
<<
seqlst                @Replace with N
3 'N' PUT
```

# Sequences, series and limits with the HP49G - Part 5

```
->ALG EXPAND EXPAND @Turn to alg. and expand
'N'
seq1st 2 3 SUB      @Get sub list with low and high
+
1 ->LIST +          @Turn to sequence
>>
>>
```

Store the program in PARTSUMSEQ. Enter the previous series  $\sum_{n=1}^{\infty} \frac{1}{2^n}$  and run PARTSUMSEQ. The result is  $\frac{2^N - 1}{2^N} \{N \ 1 \ + \}$ , the partial sums sequence in our notation. Notice that the original series must not contain N in the summand expression.

Perhaps it is interesting for you to know that even in the 18<sup>th</sup> century there was not total clarity about series. For example the limit of the series

$1 - 1 + 1 - 1 + 1 - 1 \dots = (-1)^n$  was taken to be 0 if the series were written as  $(1 - 1) + (1 - 1) + (1 - 1) \dots$ , or 1 if the series were written as  $1 - (1 - 1) - (1 - 1) - \dots$ . But if we use the partial sums sequence,  $s_0 = 1, s_1 = 0, s_2 = 1, s_3 = 0, \dots$ , we see that it doesn't converge. The series has thus no limit, it also does not converge. What does

PARTSUMSEQ do if we give it the series  $(-1)^n$ ? Enter it and press

PARTSUMSEQ. The program returns  $-\frac{e^{(i N + i)} - 1}{2} \{n \ 0 \ + \}$

and the HP49G switches to complex mode. The expression  $-\frac{e^{(i N + i)} - 1}{2}$  is what the HP49G says when it wants to tell us that it found an expression which can be 0 or 1. Indeed if in the above expression we substitute  $N = 0$  and expand, we get a 1. If we

substitute  $N = 1$  and expand we get a 0. Can the HP49G find if the

series  $\sum_{n=0}^{\infty} (-1)^n$  converges or not? Let's see. Enter  $\sum_{n=0}^{\infty} (-1)^n$  and press EXPAND. Be prepared for some unusual things. After some seconds you get the message "Bounded var error" and you can continue by pressing the menu key [OK] ([F6]). If you press [OK] then you get another message, "SERIES remainder is O(1) at order 3". Again press the menu key [OK] to continue. The HP49G rattles some seconds and then presents you another "Bounded var error". Press again [OK]. Heavens! Another "SERIES remainder is O(1) at order 3" and another key press of the menu key [OK]. And the result is...?. This can have two different meanings. Either the HP49G *found* that the series has no limit or it *didn't find* anything any wants to say "Dunno!". Notice that the HP49G has switched to complex mode and that it set n as the current VX. Use STOVX to set your VX again. But the changes go further. The HP49G has added the assumption that n is real to the list REALASSUME. Enter 'n' and use the program RCASSM from part 3 of this marathon to see that. If you don't want to have that assumption, enter 'n' and use the function UNASSUME to remove it.

We can use the program CONVERGES? and the partial sums sequence

to get an answer to the question of convergence of  $\sum_{n=0}^{\infty} (-1)^n$ . Enter this series and use PARTSUMSEQ to get again the partial sums sequence

$-\frac{e^{(i N + i)} - 1}{2} \{N \ 0 \ + \}$ . Now, you remember that the

program SPCASES will find the special cases to which the expression for the general member of the sequence branches. But we didn't program any recognition of patterns like  $e^{i N}$  and so SPCASES will

do nothing with this sequence. We must first convert  $-\frac{e^{(i N + i)} - 1}{2}$  to trigonometric expressions. Press [DUP], [HEAD] and then [SINCOS]. The result is  $-\frac{\cos((N+1)) + i \sin((N+1)) - 1}{2}$ .

# Sequences, series and limits with the HP49G - Part 5

Enter a 1 and press [SWAP], [PUT]. Now the partial sums sequence is converted to:

$$-\frac{\cos((N+1)) + i \sin((N+1)) - 1}{2} \{N \ 0 \ + \}$$

Run the program CONVERGES? and in some seconds you get {1 0} on stack level 2 and 0 on stack level 1. The 0 says that the partial sums sequence and thus the series itself doesn't converge. The list at stack level 2 contains the values of the limits of the two "branches" of the sequence, those two values that in the 18<sup>th</sup> century were said to be the limits of the series  $\sum_{n=0}^{\infty} (-1)^n$  depending on how it was written. The program CONVERGES? has given the HP49G has historical knowledge. ;-)

One of the simplest series that one can examine is the arithmetic series

$\sum_{n=0}^{\infty} a_0 + n d$ , which results from the members of the arithmetic sequence  $\{a_0 + n d \mid n \in \mathbb{N}\}$ . The series of course does not converge, since the members of the arithmetic sequence grow and grow above any finite number. Enter  $\sum_{n=0}^{\infty} a_0 + n d$  and expand. The result is  $+$ , the series goes to infinity. But we can also examine the behaviour of the *finite* arithmetic series  $\sum_{n=0}^N a_0 + n d$ . It is not known for sure if the following really happened, but it is nonetheless worth telling it. When Gauss was 9 years old, his teacher Büttner gave the class the exercise to add all numbers from 1 to 100. He had hardly returned to his desk when the small Carl Friedrich Gauss (Kalle Fritz, as we would say today) put his arithmetic slate on Büttner's desk, and said: "Dar licht se"<sup>9</sup>. The teacher smiled but he was quite surprised

when he collected all the arithmetic slates from the other children and saw that on Gauss' one there was only the correct result, 5050. No additions, no calculations what so ever! How did Kalle Fritz calculated the result of  $\sum_{n=0}^{99} 1 + n$   $\sum_{n=1}^{100} 1 + (n-1)$   $\sum_{n=1}^{100} n$  so fast? Did he have an HP49G? Well, such people don't need that, though we can of course ask, what they would be able to do if they did have an HP49G. Gauss did the following:

$$\begin{array}{ccccccccccc} 1 & + & 2 & + & 3 & + & \dots & + & 50 & + & \\ 100 & + & 99 & + & 98 & + & \dots & + & 51 & + & \end{array}$$


---


$$\begin{array}{ccccccccccc} 101 & + & 101 & + & 101 & + & \dots & + & 101 & + & \end{array}$$

50 pairs.  
Each pair sums to 101  
50\*101=5050

His teacher, Büttner, recognised that small Kalle Fritz should learn much more than what was taught in that class and so he brought him from Hamburg a special book, Remers Arithmetica. Gauss' idea can be applied to any finite arithmetic series.

Does our small HP49G knows about finite arithmetic series? Let's try.

Enter  $\sum_{n=0}^N a_0 + n d$  and expand. After some seconds the HP49G

answers with  $\frac{(2 \ N + 2) \ a_0 + d \ N^2 + d \ N}{2}$ . For the special case of

Gauss, enter  $\sum_{n=0}^{99} 1 + n$  or  $\sum_{n=1}^{100} n$  and expand to get the result 5050. If you enter  $\sum_{n=0}^{N-1} 1 + n$  or  $\sum_{n=1}^N n$  for getting the sum of all integers from 1

<sup>9</sup> "There it lies"

# Sequences, series and limits with the HP49G - Part 5

up to  $N$ , you get the result  $\frac{N^2 + N}{2}$ . Press [COLLECT] to get  $\frac{N(N+1)}{2}$ , which says exactly what Gauss found for  $N = 100$ . We have  $\frac{N}{2}$  pairs, of which each pair sums to  $N+1$ .

We can also use this fact to solve problems like the following:

- 1) Solve  $\sum_{n=1}^N n = 5050$  for  $N$ .

Enter  $\sum_{n=1}^N n = 5050$ , enter  $N$ , press SOLVE. The result is  $\{N = 100 \quad N = -101\}$ . You get two solutions because the HP49G first finds the result for the sum, which is  $\frac{N^2 + N}{2}$ , and then solves the quadratic equation  $\frac{N^2 + N}{2} = 5050$  for  $N$ . This returns the above two solutions and you have to keep the right one. If you want the HP49G to automatically filter out the solution  $N = -101$ , then enter  $N = 1$  and then ASSUME, before solving the problem. The HP49G then will return only the first solution,  $N = 100$ .

- 2) A finite arithmetic series  $\sum_{n=1}^N a_n = \sum_{n=1}^N a_1 + (n-1)d$  has  $a_1=3$ ,  $a_N = 43$  and  $d = 5$ . Find  $N$ , and the sum of the series.

First we calculate for what value of  $N$  the summand of the form  $a_1 + (N-1)d$  equals 43. Enter  $a_1 + (N-1)d = 43$ ,  $a_1=3$  and press [SUBST]. Then enter  $d = 5$  and press [SUBST]

again. Now, enter  $N$  and press [SOLVE]. The result is  $N = 9$ , which means that we have the finite series is  $\sum_{n=1}^9 a_1 + (n-1)d$ .

Enter  $\sum_{n=1}^N a_1 + (n-1)d$ ,  $N = 9$  and press [SUBST]. This will not only do the substitution but also calculate the sum. The result is:  $9 \cdot a_1 + 36 \cdot d$ . Enter again  $a_1 = 3$  and press [SUBST], then enter  $d = 5$  and press [SUBST] again. Expand to get 207, the sum of the finite series.

- 3) For a finite arithmetic series  $\sum_{n=1}^N a_1 + (n-1)d$  we know that  $d = 12$ ,  $a_N = 60$  and that its sum is equal to 180. Calculate  $N$  and  $a_1$ .

Enter  $\sum_{n=1}^N a_1 + (n-1)d = 180$  and then  $d = 12$  and press [SUBST]. The HP49G calculates the sum and returns  $N \cdot a_1 + 6 \cdot N^2 - 6 \cdot N = 180$ . Enter  $a_1$  and press [SOLVE] to get  $a_1 = -\frac{6 \cdot N^2 - 6 \cdot N - 180}{N}$ . Now enter the  $N^{\text{th}}$  summand,  $a_1 + (N-1)d$ , press [OVER] and then [SUBST] to substitute for  $a_1$ . Enter  $d = 12$  and substitute again. Enter 60, the value of the  $N^{\text{th}}$  summand, and press [=]. Now you have the equation  $-\frac{6 \cdot N^2 - 6 \cdot N - 180}{N} + (N-1) \cdot 12 = 60$  which can be solved for  $N$ . Enter  $N$  and press [SOLVE]. You get two solutions in a list:  $\{N = 6 \quad N = 5\}$ . Both are valid and we will soon see why. Now press [ARROW-LEFT] to swap stack levels 1 and 2 and then press [OVER] to get a copy of the solutions list on stack level 1. Press [SUBST] to make the two substitutions  $N = 6$  and



# Sequences, series and limits with the HP49G - Part 5

$N = 5$  and then press [EXPAND]. The result is the list  $\{a1 = 0 \quad a1 = 12\}$ . The results tell us that there are two series are solutions to our problem. The first is  $\sum_{n=1}^6 0 + (n-1) \cdot 12$  and the second is  $\sum_{n=1}^5 12 + (n-1) \cdot 12$ .

The problem can also be solved using another way. Enter  $\sum_{n=1}^N a1 + (n-1) \cdot d = 180, d = 12$  and press [SUBST] to get  $\sum_{n=1}^N a1 + 6 \cdot N^2 - 6 \cdot N = 180$ . Now, enter  $a1 + (N-1) \cdot d = 60, d = 12$  and [SUBST] to get  $a1 + (N-1) \cdot 12 = 60$ . Enter 2 and press [->LIST] to get a list containing the two equations. Press [AXL] to convert the list to a vector. Now you have a vector with the two equations:

$$\left[ N \quad a1 + 6 \cdot N^2 - 6 \cdot N = 180 \quad a1 + (N-1) \cdot 12 = 60 \right]$$

Enter the vector of unknowns,  $[N \quad a1]$ . Press [GBASIS]. The command GBASIS takes a vector of simultaneous polynomial equations and a vector with the unknowns contained in the polynomials. It returns a vector of equations which can be used to find solutions. In our case the result is:

$$\left[ 12 \cdot N + a1 - 72 \quad a1^2 - 12 \cdot a1 \right]$$

The second equation,  $a1^2 - 12 \cdot a1 = 0$ , contains only one unknown,  $a1$ . Press [OBJ->] and [DROP] to explode the vector. Enter  $a1$  and press [SOLVE]. The result is the list  $\{a1 = 0 \quad a1 = 12\}$ . Press [ARROW-UP] to go to the interactive stack. Press [ARROW-UP] once more to go to stack level 2, and then press the menu key [ROLL] ([F5]) to roll the contents of stack level 2 to stack level 1, effectively swapping the two

stack levels. Press the menu key [PICK] ([F4]) to put a copy of the contents of stack level 2 on stack level 1. Press [ON] to leave the interactive stack. Now press [SUBST] to substitute the two solutions for  $a1$  in the equation  $12 \cdot N + a1 - 72$ . Enter  $N$  and press [SOLVE] to get the solutions  $\{N = 6 \quad N = 5\}$ .

The results of GBASIS can always be used to find first solutions for one unknown, substitute these solutions in the previous equation, find solutions for the next unknown, and so on, subsequently substituting and solving for all unknowns.

The next series that we examine is the finite geometric series  $\sum_{n=0}^N a_0 \cdot q^n$ . First of all, let's see if the HP49G can tell us what the sum

of the series is. Enter  $\sum_{n=0}^N a_0 \cdot q^n$  and expand. The HP49G needs some

seconds to return  $\frac{a_0 \cdot |q|^{(N+1)} - a_0}{q - 1}$ . What would happen if we give it

assumptions for  $q$ ? Enter  $q \neq 0$  and use ASSUME to make this assumption about  $q$ . Enter again  $\sum_{n=0}^N a_0 \cdot q^n$  and expand. This time the

result is  $\frac{a_0 \cdot q^{(N+1)} - a_0}{q - 1}$ , which means that the assumption was taken

into consideration. Remove the current assumption by entering  $q$

UNASSUME. Enter  $q \neq 0$  ASSUME. Enter again  $\sum_{n=0}^N a_0 \cdot q^n$  and

expand. The result is now  $-\frac{a_0 \cdot q^{(N+1)} \cdot e^{(i \cdot N + i)} - a_0}{q + 1}$  and the HP49G

has switched to complex mode. Remember that the expression  $e^{(i \cdot N + i)}$  is HP49G's way to say "either 1 or -1". You can demonstrate this.

# Sequences, series and limits with the HP49G - Part 5

Press [ARROW-DOWN] to get the result in the EQW and select the sub-expression  $e^{(N+1)}$ . Press [RED-SHIFT] and then [VAR] to copy the sub-expression. leave the EQW by pressing [ON]. Now, press [RED-SHIFT] and then [NXT] to paste the sub-expression on the command line. Press [ENTER] to put it on stack level 1. Now, enter  $N \ 0 \ 3 \ 1$  and press [SEQ] to make a sequence of  $e^{(N+1)}$  for  $N = 0$  to  $N = 3$ . The result is the list  $\{-1 \ 1 \ -1 \ 1\}$ .

For  $q = 1$  the series is  $\sum_{n=0}^N a_0 \cdot 1^n = \sum_{n=0}^N a_0 = a_0 + n \cdot 0 = a_0 \cdot (N+1)$ , that is a simple arithmetic series with  $d = 0$ .

Remove now all assumptions for  $q$ , by entering  $q$  UNASSUME.

Using the fact that the HP49G "knows" about finite geometric series, we can solve problems like the following:

- 1) A finite geometric series  $\sum_{n=0}^N a_0 \cdot q^n$  has  $a_0 = 2$ ,  $q = 5$  and its sum is 976562. Calculate  $N$ .

In real mode enter  $\sum_{n=0}^N a_0 \cdot q^n = 976562$  and  $q = 5$ . Press [SUBST]. The HP49G returns  $\frac{a_0 (e^{(N+1) \ln 5} - a_0)}{4} = 976562$ . If you don't like the exponential form, enter EXP2POW to convert the result to  $\frac{a_0 (5^{(N+1)} - a_0)}{4} = 976562$ . Enter  $a_0 = 2$  and press [SUBST] again to get  $\frac{2 (5^{(N+1)} - 2)}{4} = 976562$ . Enter  $N$  and press [SOLVE] to get  $N = 8$ .

2) The arab historian Ja'qubi reports: The Sheikh of Persia was very pleased by the game of chess, that he wanted to do the inventor of the game, Sessa Ebn Daher, any favour. Sessa Ebn was apparently much cleverer than the Sheikh. He answered that he wanted to have the amount of corn that results, if someone puts one grain of corn on the first square of the checkerboard, two on the next, four on the next next, and so on continuing always with twice the number of the previous square. The unlucky Sheikh accepted, but... he could never do Sessa that favour. Find out why.

Since the chessboard has 64 squares the total number of grains would have been  $\sum_{n=0}^{63} 1 \cdot 2^n$  which expanded on the HP49G returns  $2^{64} - 1$ . If we assume that the whole earth is a single corn field that with the approximate surface of  $5.1E10\_ha$ , that it produces  $40 \frac{dt}{ha \ yr}$  and each  $dt$  has  $2E6$  grains, then the whole earth produces the amount of:

$$5.1E10\_ha \cdot 40 \frac{dt}{ha \ yr} \cdot 2E6 \frac{grain}{dt} = 4.08E18 \frac{grain}{yr}.$$

That means that we can harvest the amount of  $(2^{64} - 1) \cdot 1\_grain$  in  $\frac{(2^{64} - 1) \cdot 1\_grain}{4.08E18 \frac{grain}{yr}}$ . Pressing ->NUM

this gives us about 4.5\_yr ! The moral of the story: Arab game inventors are much much cleverer than greek sophists. ;-)

# Sequences, series and limits with the HP49G - Part 5

After the finite geometric series we take a look at the infinite geometric series  $\sum_{n=0}^{\infty} a_0 q^n$ , or just geometric series for brevity. We have already

seen that the  $N^{\text{th}}$  member of the partial sums sequence of this series is  $\sum_{n=0}^N a_0 q^n = \frac{a_0 (q^{N+1} - 1)}{q - 1}$ . The fate of the series depends on  $q$ . If

$|q| < 1$  then it converges. If  $|q| > 1$  then it doesn't converge. Let's see

what the HP49G says for  $\sum_{n=0}^{\infty} a_0 q^n$  in the case  $|q| < 1$ . We can't make this assumption in this form. Instead of this, enter  $q$  UNASSUME,  $q - 1$  ASSUME,  $q + 1$  ASSUME. Now the list REALASSUME in

CASDIR contains  $q - 1$  AND  $q + 1$ . Enter  $\sum_{n=0}^{\infty} a_0 q^n$  and

expand. The HP49G returns  $-\frac{a_0}{q - 1}$  which is correct. For the case

$|q| > 1$  we must consider the cases  $q - 1$  and  $q + 1$  separately. Enter

$q$  UNASSUME  $q - 1$  ASSUME. Now, enter  $\sum_{n=0}^{\infty} a_0 q^n$  and expand again. The result now is  $+$  which is also correct. Enter  $q$

UNASSUME  $q + 1$  ASSUME. Enter  $\sum_{n=0}^{\infty} a_0 q^n$  and expand another

time. Now it takes much longer to get an answer. At the end after much rattling the HP49G says ?. When  $q = -1$  the summands  $a_0 q^n$  will alternative from positive to negative and vice versa, with always bigger absolute values. So the series doesn't converge as it will switch its sign from  $+$  to  $-$  with every new added member. Notice that the HP49G has switched VX to  $n$ . So use X (or what your VX is) STOVX to restore your current VX. Notice also that the HP49G has switched to complex mode, so you might want to restore real mode now. Last thing to do, enter  $q$  UNASSUME to get rid of the assumptions for  $q$ .

One of the practical applications of the geometric series is presumably part of XQ and  $\rightarrow Q$ . Consider for example the periodical decimal number .2525... This can be represented as  $\frac{25}{100} + \frac{25}{10000} + \dots$ . The

latter is the geometric series with  $a_0 = \frac{25}{100}$  and  $q = \frac{1}{100}$ . That is, the

number .2525... is equal to  $\sum_{n=0}^{\infty} \frac{25}{100} \left(\frac{1}{100}\right)^n$ . As we already found

out, the series converges to  $-\frac{a}{q - 1} = -\frac{\frac{25}{100}}{\frac{1}{100} - 1} = \frac{25}{99}$ . Enter

$\sum_{n=0}^{\infty} \frac{25}{100} \left(\frac{1}{100}\right)^n$  and expand to get the result  $\frac{25}{99}$ . If you now press the key  $[-\rightarrow \text{NUM}]$  you get the original number .2525...

Another example. Suppose that you have six lines passing through point O and having all an angle of  $\frac{\pi}{6}$  to each other. From point P0 of one of the lines we draw one line segment orthogonal to the next line until point P1. From there we draw a line segment orthogonal to the next line until P2. And so on. What is the length of the "spiral"?

For the length  $l_0$  of first segment we have:  $l_0 = a \sin \frac{\pi}{6}$ . The

length of the segment  $\overline{OP_1}$  is  $a \cos \frac{\pi}{6}$ . Thus for the length of  $l_1$

we have  $l_1 = a \cos \frac{\pi}{6} \sin \frac{\pi}{6}$ . The length of  $\overline{OP_2}$  is

$a \cos^2 \frac{\pi}{6} \cos \frac{\pi}{6} = a \cos^3 \frac{\pi}{6}$ . Thus for the length of  $l_2$

# Sequences, series and limits with the HP49G - Part 5

we have  $l_2 = a \cos^2 \frac{\pi}{6} \sin \frac{\pi}{6}$ . The infinite sum  $\sum_{n=0}^{\infty} l_n$  is a geometric series with  $l_0 = a \sin \frac{\pi}{6}$  and  $q = \cos \frac{\pi}{6}$ . The series is  $\sum_{n=0}^{\infty} a \sin \frac{\pi}{6} \cos^n \frac{\pi}{6}$ . Because  $-1 < q = \cos \frac{\pi}{6} = \frac{\sqrt{3}}{2} < 1$  the series must converge. Enter  $\sum_{n=0}^{\infty} a \sin \frac{\pi}{6} \cos^n \frac{\pi}{6}$  and press [EXPAND] to get the length of the spiral,  $(2 + \sqrt{3}) a$ .

We already see that the build-in  $\sum$  is quite powerful and can find the sum of many series. But there are also series for which  $\sum$  returns a question mark. Examples:

$$\sum_{n=1}^{\infty} \frac{1}{\sqrt{n}}$$

The series doesn't converge. The result should be +∞.

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$$

The series converges.

$$\sum_{n=1}^{\infty} \frac{\cos n \left( \frac{\pi}{2} + \frac{\pi}{4} \right)}{n}$$

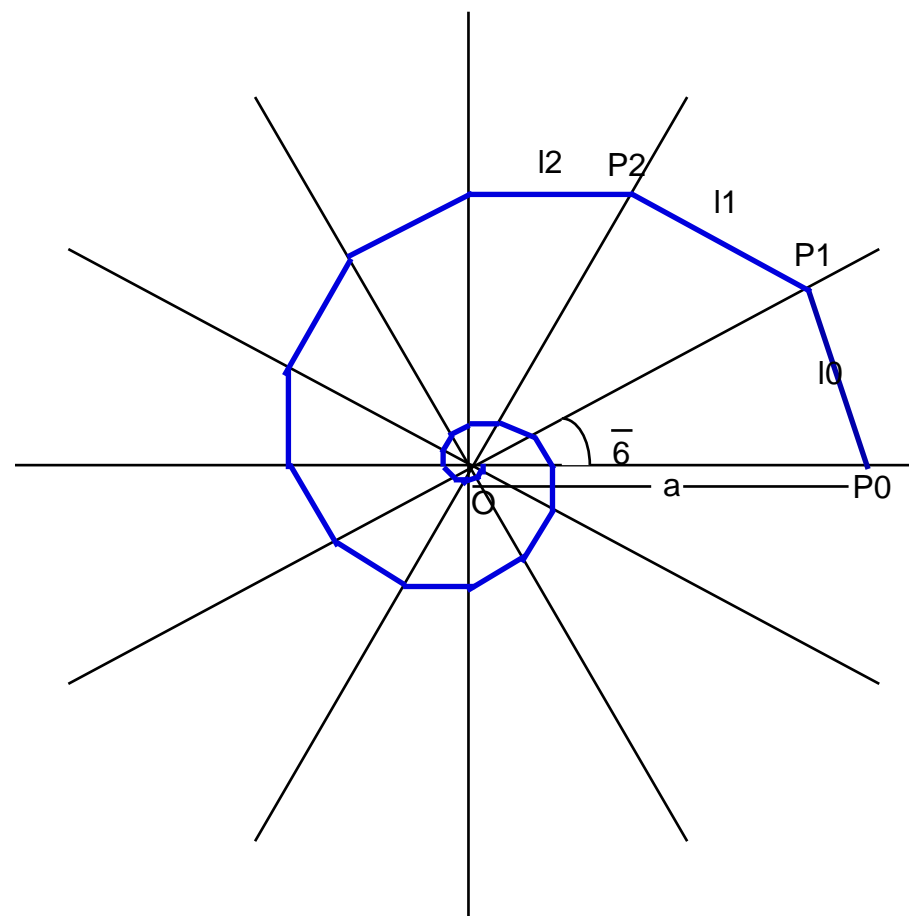
Doesn't converge, goes to +∞.

$$\sum_{n=1}^{\infty} 1 - \frac{1}{n^n}$$

Doesn't converge, goes to +∞.

$$\sum_{n=1}^{\infty} \frac{\ln(n)}{2^n}$$

Converges.



$$\sum_{n=1}^{\infty} \frac{n!}{n^n}$$

The series converges. If you try to expand this the HP49G says "Operator not implemented (SERIES)" and returns a ?.

First of all we need some criterion that lets us distinguish between series that we should further examine for convergence, and series for which we can say that they don't converge and so no further checking

# Sequences, series and limits with the HP49G - Part 5

is necessary. An easy implementable criterion would be:

**For convergence of a series  $\sum_{n=1}^{\infty} a_n$  it is necessary (but inconclusive) that the sequence  $a_n$  of the summands converges to 0.**

The above means that if we find for some series  $\sum_{n=1}^{\infty} a_n$  that the limit  $\lim_n a_n > 0$  or that the limit doesn't exist, then we can say that the series diverges. If we find that  $\lim_n a_n = 0$  then the series  $\sum_{n=1}^{\infty} a_n$  can converge but it doesn't have to, and so we must further examine it. This criterion (or convergence test) is useful because it lets us filter out all series for which no further testing is necessary, diminishing thus the amount of work that the HP49G has to do. Take for example the series  $\sum_{n=1}^{\infty} 1 - \frac{1}{n}$ . If you expand that then you get ? as the result.

The built-in functionality of `lim` is not enough even for saying if the series converges or not. But we can examine  $1 - \frac{1}{n}$ . Entering that and then  $n = \frac{1}{e}$  and pressing `[lim]`, returns  $\frac{1}{e}$ . Because  $\frac{1}{e} > 0$  we know that the series diverges and that no further tests are necessary.

Another example, let's consider  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$ . If you expand this, you get a ?. Enter  $\frac{(-1)^{n+1}}{n}$ ,  $n =$  and press `[lim]`. The HP49G returns... "lim Error: Mode switch not allowed here". So we have a problem. But hey! We have the program `CONVERGES?`. Let's try that. Enter

the sequence  $\frac{(-1)^{n+1}}{n}$  `{n 1 }` and press `CONVERGES?` The result after some seconds is the list `{0 0}` on stack level 2 and a 1 on stack level 1 which says that  $\frac{(-1)^{n+1}}{n}$  does converge. It branches to two special cases, each of them having the limit 0 when  $n =$ . For us this means that we have to further examine the series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  to see if it converges or not. It also means that we should rather use the program `CONVERGES?` and not the built-in `lim`. A program for enhancing the built-in capabilities of the HP49G regarding series, would in general:

- 1) Take a series  $\sum_{n=1}^{\infty} a_n$  from the stack.
- 2) Check if expanding returns a result different than ?.
- 2) If it does, then we're done.
- 3) If the result is ?, then check convergence of the sequence  $\{a_n \{n 1 \}\}$  using the program `CONVERGES?`
- 4) If `CONVERGES?` finds that the sequence  $\{a_n \{n 1 \}\}$  converges to 0, or if it can't find if the sequence converges, then do further tests.
- 5) If `CONVERGES?` finds that the sequence  $\{a_n \{n 1 \}\}$  converges to a limit different than 0, or that it diverges, then we are done and the series  $\sum_{n=1}^{\infty} a_n$  diverges.

Perhaps you ask yourself now, why we don't use the partial sums sequence. If this sequence converges to some limit, then the series itself converges to the same limit. But the problem is that if the HP49G can't do anything with the series  $\sum_{n=1}^{\infty} a_n$ , then it is likely that it will not

# Sequences, series and limits with the HP49G - Part 5

be able to do much with the partial sums sequence  

$$\sum_{n=1}^N a_n \{N-1\}$$
. Sometimes it will be able to transform the finite  

$$\sum_{n=1}^N a_n$$
 to some analytic closed form as a function of N, but then it  
will not be able to find if limit of this analytic closed form exists for  
N. Our program in construction for enhancing the built-in is:

```
<<
  PUSH                      @Save flags and current dir.
  DUP OBJ-> DROP2           @Explode series
  { HOME CASDIR
  REALASSUME } RCL          @Recall current assumptions
  RCLVX                     @Recall current VX
  -> series ivar lo hi
  sumnd assmlst vx          @Store in locals
  <<
    IF
      "Trying built-in
methods..." 1 DISP
      series EXPAND          @Try expanding
      -103 CF                @Sometimes expanding a series
                              @will switch to complex mode,
                              @so re-switsh to real mode.

      DUP series SAME
      NOT                    @If result is not the series
      OVER ? SAME NOT       @itself and not ?
      AND
      THEN
        IF                  @Then if result is +/-
          DUP ABS
          SAME
        THEN                @Then return 0 (divergence)
          0
        ELSE                @Else return 1 (convergence)
          1
        END IF
      END THEN
    END IF
  END IF
END
```

```
END
ELSE
  DROP
  "Testing summands
sequence -> 0" 1 DISP
  sumnd ivar lo hi          @Construct summands sequence
  3 ->LIST 2 LIST
  CONVERGES?                @Use CONVERGES?
  IF
    ? DUP SAME              @If CONVERGES? can't handle this
    ROT HEAD XQ             @or if it finds that sequence
    0 SAME ROT              @converges to 0
    AND OR
  THEN
    CASE
      ...Convergence test 1
      THEN
        ...Result 1
      END
      ...Convergence test 2
      THEN
        ...Result 2
      END
      ...Further tests
      series ?              @If all tests failed, return
                           the series and a ?
    END
  ELSE                      @else (summands seq. diverges or
    series 0                @it doesn't converge to 0) return
  END                      @series and 0 (for no convergence)
END
vx STOVX                   @Clean up and restore everything
{ HOME CASDIR } EVAL
assmlst 'REALASSUME'
STO
>>
POP
>>
```

# Sequences, series and limits with the HP49G - Part 5

The code in light grey is the place where further convergence tests will be made. The command `DISPXY` is similar to `DISP`, except that it can display at any screen coordinates using any font. It takes the object to display from stack level 3, a list with the pixel coordinates from stack level 2 and 0,1,2,3 (the font size) from stack level 1, and displays the object starting at the specified coordinates of the screen and using the specified font size. Notice that when the object to be displayed is a GROB (G**R**aphics **O**bject), then the command will not display the GROB itself but the description of the GROB, which reads `Graphic n × m` and means a GROB with size  $n \times m$ . If you want to display some GROB on the screen, then take a look at the programs of the previous parts of this marathon.

Now, what convergence tests can we use on the HP49G? What criteria are easily implementable? We have for example the ration test (Cauchy or d'Alembert ratio test) which we can use. It says:

**The series  $\sum_{n=1}^{\infty} a_n$  converges if  $\lim_{n \rightarrow \infty} \left| \frac{a_{n+1}}{a_n} \right| < 1$ , it diverges if  $\lim_{n \rightarrow \infty} \left| \frac{a_{n+1}}{a_n} \right| > 1$ . The case  $\lim_{n \rightarrow \infty} \left| \frac{a_{n+1}}{a_n} \right| = 1$  is non-conclusive.**

This is very easy to implement and we can again use CONVERGES? in order to be able to handle more expressions than these that the HP49G handles out of the box. We have to first construct the

sequence  $\left\{ \frac{a_{n+1}}{a_n} \right\}$ , then give this to CONVERGES? and

use the returned results. So, the light grey part in the above program becomes:

```

....
CASE
  "Assuming" 1 DISP      @Construct assumpt.
  "" ivar UNASSUME +    @and make assumpt.
  " " + lo + "" +

```

```

DUP {#50h #0h} 2
DISPXY OBJ->
ASSUME DROP
"Testing ratio
1 DISP
sumnd DUP ivar DUP
1 + = SUBST SWAP
/ EXPAND
-103 CF
ABS EXPAND ivar
lo hi 3 ->LIST
2 ->LIST CONVERGES?
DUP2 -> 1 r
<<
CASE
    r ? SAME
    THEN
        0
    END
    r 0 == SAME
    THEN
        0
    END
    r 1 ==
    1 HEAD 1
    == AND
    THEN
        0
    END
    1
END
>>
THEN @If we have a 1 on stack
IF @If sequence converges
THEN
    HEAD
    IF
        1 @to limit less than 1

```

# Sequences, series and limits with the HP49G - Part 5

```

THEN      @then return series and 1 (for
          series 1      @convergence)
ELSE      @else return series and 0 (for
          series 0      @divergence)
END
ELSE      @else series doesn't converge
DROP      @return series and 0 (for
          series 0 @divergence)
END
END DROP2
...Convergence test 2
THEN
...Result 2
END
...Further tests
series ?   @If all tests failed, return
           the series and a ?
END
....

```

At this point some strange properties of the HP49G should be explained. Enter  $\frac{1}{2}$ , then 1, and then press [ ]. The result is a 1,

clearly because  $\frac{1}{2}$  is less than 1. Yes, but now re-enter  $\frac{1}{2}$  and 1 and

press [ < ]. The result is...  $\frac{1}{2} < 1$ ! Press [EXPAND], press

[COLLECT], do anything you want. The darn thing won't return 1,

though  $\frac{1}{2}$  is less than 1. The only thing that makes the HP49G give us

a result, is to press [->NUM]. Then we get a 1. For me this is a rather strange behaviour, but, as Trabakoulas says, "It is, as it is". Accepting that we must find ways out of the troubles. Oh, I didn't say what the

trouble is? Well, say if the sequence  $\left| \frac{a_{n+1}}{a_n} \right| \{n \ 1 \}$  converges,

we must find if  $\lim_n \left| \frac{a_{n+1}}{a_n} \right| < 1$ . But the comparison < doesn't seem to work as we would expect. That's why the above code tests first if the limit of  $\left| \frac{a_{n+1}}{a_n} \right|$  for n is equal to 1 using the function ==. If it

isn't, then using the function and finding that  $\lim_n \left| \frac{a_{n+1}}{a_n} \right| = 1$  is true, is

the same like finding that  $\lim_n \left| \frac{a_{n+1}}{a_n} \right| < 1$  is true, because we already

have filtered out the possible case  $\lim_n \left| \frac{a_{n+1}}{a_n} \right| = 1$ .

The code of above tests if the sequence  $\left| \frac{a_{n+1}}{a_n} \right| \{n \ 1 \}$

converges using the program CONVERGES? If a question mark is returned on stack level 1, or if a 1 is returned on stack level 1 and a list containing only 1s is returned on stack level 2, or if it is found that the sequence diverges, then the code proceeds with the next convergence test (which we didn't programmed yet). But if a 1 is returned on stack level 1, and a list containing something different than only 1s on stack level 2, then the code compares the contents of the list against 1, to say if the sequence converges or not.

We move on to the second convergence test. We use the root test which says:

**The series  $a_n$  converges if  $\lim_n \sqrt[n]{|a_n|} < 1$ , it diverges if**

**$\lim_n \sqrt[n]{|a_n|} > 1$ . The case  $\lim_n \sqrt[n]{|a_n|} = 1$  is non-conclusive.**



# Sequences, series and limits with the HP49G - Part 5

The code for this test is almost the same with the code of the previous test, except for the construction of the sequence  $\left\{ \sqrt[n]{|a_n|} \quad \{n \ 1 \} \right\}$  that we give to CONVERGES?. So, to save paper and the trees, we take a look only at the part that is different.

```

...                               @End of first test
END DROP2
"Assuming" 1 DISP                @Construct assumpt.
...                               @Same like above
"Testing root                    @Build up
criterion" 1 DISP                @XROOT(n,ABS(An))
sumnd ABS
-103 CF
EXPAND
ivar XROOT
EXPAND
ivar lo hi                      @etc like above
.....
                               @end of second test

END DROP2
...Further tests
series ?                        @If all tests failed, return
                               the series and a ?

END

```

You may have noticed that at some points we switch to real mode using -103 CF. Why that? Well, some operations, like ABS, have the nasty habit to switch to complex mode. This can make things more complicated than they need to be because then many operations will ignore the assumptions that we have made for n. So we insert the instruction -103 CF to make sure that we are in real mode and so the assumptions are taken into consideration. This can play a significant role when expanding absolute values and roots.

Let's add a third convergence test, the test of Leibniz for series with an alternating sequence of summands. Such a series is for example

$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$ . Its summands are  $1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \dots$ . Their signs are alternating  $+, -, +, -, \dots$  and thus the name "alternating sequence". The test says:

**The alternating series  $\sum_{n=1}^{\infty} a_n$  converges if the sequence of the absolute values of the summands is a monotonically decreasing sequence with the limit 0.**

This test is only for this particular kind of series, but we will see later on some interesting implications that it brings. We can implement this test, using CONVERGES? and SEQMONTY, the program that returns the monotony behaviour of a sequence. We also use SPCASES to find out how many special cases there are, and check if we have an even number of them, to make sure that alternating behaviour is possible. (What would be if we have an odd number of special cases? Can we use this test then?)

```

....                               Previous tests
END DROP 2
"Testing Leibniz                @Build up sequence
criterion" 1 DISP                @An
sumnd ivar lo hi
3 ->LIST 2 ->LIST
SPCASES                          @Find sp. cases
IF                                @If even number
  DUP SIZE 2 MOD                 @of special cases
  NOT
THEN
  1                               @Do to each sp. cas
  <<
    "Finding sign                @Find its sign
    " NSUB + " of " + ENDSUB + 1 DISP
    OBJ-> DROP
    OBJ-> DROP2

```

# Sequences, series and limits with the HP49G - Part 5

```

        SWAP UNASSUME
        "' SWAP +
        " " + SWAP +
        "' + OBJ->
        ASSUME DROP
        SIGN EXPAND
    >> DOSUBS
    DUP "Checking sign    @Find object types
result types" 1 DISP 1 << TYPE >>    @of results of SIGN
    DOSUBS
    9 POS NOT            @No alg. objects
    "Checking alternating
signs" 1 DISP SWAP 2 << NEG == >>    @Check each sign
    DOSUBS 1 +          @is neg. of next
    << AND >> STREAM AND
    "Checking ABS(summand)
sequence -> 0" 1 DISP
    "' ivar UNASSUME
    + " " + lo + "' +
    OBJ-> ASSUME DROP    @Build up sequence
    sumnd ABS -103 CF    @ABS(An)
    EXPAND ivar lo hi
    3 ->LIST 2 ->LIST DUP SEQMONTY
    -2 SAME SWAP CONVERGES? SWAP
    HEAD XQ 0 SAME AND
    AND AND
    ELSE                @Odd number of sp. cases
        DROP 0
    END
    THEN    @Leibniz true, series converges
        series 1
    END
        series ?    @No test true, series might
    END        @converge or not
    ....        @All tests done

```

This is the program CONVERGES? that comes with this document. Before we test it, some words on modifications of the other programs

that we have so far. The programs SPCASES, RCR->ANL, SEQMONTY, and BOUNDS are modified. They also save the current assumptions and restore them when they finish. You should transfer the new versions to your HP49G, or else the examples below might not work.

And now for the tests. First we give it some series that the HP49G can deal with out of the box. Enter  $\sum_{n=1}^{\infty} n$  and press [ CONVERGES?]. In about 4 seconds the HP49G returns + on stack level 2 and a 0 on stack level 1, to say that this series diverges to positive infinity. Enter

$\sum_{n=1}^{\infty} \frac{1}{n^2}$  and press [ CONVERGES?]. Again in about 4 seconds the

HP49G returns  $\frac{2}{6}$  on stack level 2 and a 1 on stack level 1. This

means that the series converges to  $\frac{2}{6}$ .

And now some series for which the HP49G can't even find if they converge. Enter the alternating series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  and press DUP to make a copy on stack level 2. Press first [EXPAND]. The HP49G needs about 5 seconds to return ? (for "I don't know") and switch to complex mode. Switch back to real mode, DROP the question mark and press [ CONVERGES?]. This time, after 180 seconds, the HP49G returns the series itself on stack level 2 and a 1 on stack level 1. This means that the series does converge, but we don't know what it converges to. Well, at least we know that it converges. We will see later, if and how it is possible to find also what the series converges to.

Enter  $\sum_{n=1}^{\infty} \frac{1}{2^n \sqrt{n}}$ . This is a series which the HP49G can't deal with

out of the box. Press [ CONVERGES?]. After 63 seconds the HP49G returns the series back on stack level 2 and 1 on stack level 1.

# Sequences, series and limits with the HP49G - Part 5

Again, we know that the series does converge but we don't know what it converges to.

Enter  $\sum_{n=1}^{\infty} \frac{\cos n}{2 + \frac{1}{4}}$ , press [ CONVERGES?]. This time, after 10 minutes (!) the HP49G says that it can't find if this series converges or not. Well, we can't always win.

Enter  $1 - \frac{1}{n}$  and press [ CONVERGES?]. (If you EXPAND this the HP49G returns ?.) The HP49G returns the series at stack level 2 and a 0 at stack level 1, that is, it says that the series diverges after 101 seconds.

Enter  $\sum_{n=1}^{\infty} \frac{\ln(n)}{2^n}$ , another series that the HP49G can't deal with out of the box. Press [ CONVERGES?]. The HP49G needs about 3.5 minutes to find that the series does converge, returning the series on stack level 2 and a 1 on stack level 1.

Enter  $\sum_{n=1}^{\infty} \frac{n!}{(n+1)!}$ . Expanding this you would get a ? quite quickly, though this series is the same like  $\sum_{n=1}^{\infty} \frac{1}{n+1}$ , for which the HP49G can find that it diverges to + out of the box. But pressing [EXPAND] with  $\sum_{n=1}^{\infty} \frac{1}{n+1}$  on stack level 1 also changes the current VX to n. If you give the series  $\sum_{n=1}^{\infty} \frac{n!}{(n+1)!}$  to [ CONVERGES?], then after 3.2 seconds you get + on stack level 2, a 0 (for divergence) on stack level 1, and the current VX remains the same.

Last example: Enter  $\sum_{n=1}^{\infty} \frac{n!}{n^n}$  and press [ CONVERGES?]. After some seconds the HP49G flashes the message "Operator not implemented (SERIES)" and presents you a menu with [OK] ([F6]) for continuing. Press [OK] and wait until you get the message "SERIES remainder is O(1) at order 3". Press again the menu key [OK]. After a long time the message "Operator not implemented (SERIES)" again. Press again [OK]. After this adventurous story, the HP49G returns the series on stack level 2 and a ? on stack level 1. This time it was work for nothing, but we already have so many examples where the program did its job OK.

Of course there are many more convergence tests that you can add to CONVERGES? for further expanding its capabilities. As we already have noticed, it is possible with a little programming work to give the HP49G more power.

We now move on to some rules that may be helpful for working with series on the HP49G. The first of them is:

**The convergence behaviour and the limit of a convergent series don't change if the summands are grouped in parentheses in some arbitrary way.**

For example, the convergent series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$  remains convergent if we write it as  $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$  or  $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \dots$  or in any other way, grouping the

summands in parentheses. That means, *if we already have checked that the series converges*, we can start grouping the summands in many different ways. The limit of the sequence remains the same. How can this be useful for us? Well, let's consider the grouping

# Sequences, series and limits with the HP49G - Part 5

$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$ . Since we know that the series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  converges, we know that we can use  $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$

instead of  $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$  in any of our considerations. Now,

we could of course consider the groups  $1 - \frac{1}{2}$ ,  $\frac{1}{3} - \frac{1}{4}$  and so on,

as summands of a series. That is, we consider the series  $\sum_{n=1}^{\infty} \left( \frac{1}{n} - \frac{1}{n+1} \right)$ .

But we must be careful because we can't use the same index values.

For example enter  $\sum_{n=1}^{10} \frac{(-1)^{n+1}}{n}$  and press [->NUM]. The result is

.645634... Now enter  $\sum_{n=1}^{10} \left( \frac{1}{n} - \frac{1}{n+1} \right)$  and press [->NUM] again to get

.909090... Obviously something is going wrong. We can see what is going wrong if we consider the first sum, which is

$$\sum_{n=1}^{10} \frac{(-1)^{n+1}}{n} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \frac{1}{7} - \frac{1}{8} + \frac{1}{9} - \frac{1}{10}$$

and compare it to the second sum:

$$\sum_{n=1}^{10} \left( \frac{1}{n} - \frac{1}{n+1} \right) = 1 - \frac{1}{2} + \frac{1}{2} - \frac{1}{3} + \dots + \frac{1}{9} - \frac{1}{10} + \frac{1}{10} - \frac{1}{11} = 1 - \frac{1}{11}$$

They are not the same because except for the first summand  $\frac{1}{1} = 1$  and

the last  $\frac{1}{10+1} = \frac{1}{11}$  each value of  $n$  constructs the same summands twice, once with positive and once with negative sign as shown in the table.

$n$	1	2	3	...
$\frac{1}{n} - \frac{1}{n+1}$	$\frac{1}{1} - \frac{1}{2}$	$\frac{1}{2} - \frac{1}{3}$	$\frac{1}{3} - \frac{1}{4}$	...

What we need is to somehow transform the series  $\sum_{n=1}^{\infty} \left( \frac{1}{n} - \frac{1}{n+1} \right)$  in a way that each summand of the original series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  appears only

once with the right sign. Let's look again at the grouping that we

chose:  $\frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$ . Each group consists of two

members. We want these two members to appear both only once for a *single* value of  $n$ . Of each two members in a group the first has

always an odd denominator, that is it has the form  $\frac{1}{2n+1}$  if we start

at  $n=0$  instead of  $n=1$ . The second has the form  $\frac{1}{2n+2}$  if we

start at  $n=0$ . That means, we can write the summand  $\frac{1}{n} + \frac{1}{n+1}$  in the

series  $\sum_{n=0}^{\infty} \left( \frac{1}{2n+1} - \frac{1}{2n+2} \right)$ , provided that we start at

$n=0$ . And so the series becomes now:  $\sum_{n=0}^{\infty} \left( \frac{1}{2n+1} - \frac{1}{2n+2} \right)$ . This

series is the same like  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  except that it creates two summands

with every single value of  $n$ . We could say that "it grows twice as fast".

You could test this, by entering for example  $\sum_{n=1}^{10} \frac{(-1)^{n+1}}{n}$ , then pressing

[->NUM] to get .645634... Then enter  $\sum_{n=0}^4 \left( \frac{1}{2n+1} - \frac{1}{2n+2} \right)$  and

press [->NUM] again to get the same result, .645634... Notice that the first sum goes from  $n=1$  to  $n=10$ , while the second goes from  $n=0$  to  $n=4$ . This is because, as already said, the second

# Sequences, series and limits with the HP49G - Part 5

sum (and also the series) creates two summands of the first series by a single value of  $n$ .

$n$	1	2	3	4	...
$\frac{(-1)^{n+1}}{n}$	$\frac{1}{1}$	$-\frac{1}{2}$	$\frac{1}{3}$	$-\frac{1}{4}$	...
$\frac{1}{2n+1} - \frac{1}{2n+2}$	$\frac{1}{1} - \frac{1}{2}$	$\frac{1}{3} - \frac{1}{4}$	...		

If we expand the summand  $\frac{1}{2n+1} - \frac{1}{2n+2}$  in the series  $\sum_{n=0}^{\infty} \left( \frac{1}{2n+1} - \frac{1}{2n+2} \right)$ , then we get  $\frac{1}{4n^2+6n+2}$  and the series becomes then  $\sum_{n=0}^{\infty} \frac{1}{4n^2+6n+2}$ . The convergence and the limit of

this series is exactly the same like those of  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  because the new series was created through grouping and "renumbering" of the summands. But the second series has two big advantages. The first advantage is that the HP49G can work much better with the second series. Enter  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$  and press [EXPAND], to try to find what the general member of the partial sums sequence looks like. Result is, nada! The HP49G has only switched to complex mode. But enter  $\sum_{n=0}^N \frac{1}{4n^2+6n+2}$  and press [EXPAND] again. Aha! Now you get

the result  $\frac{\text{Psi } N+1 - \frac{-1}{2} - \text{Psi}(N+1+1) + \text{Psi} - \frac{-1}{2} - \text{Psi}(1)}{2}$ . In order to get the  $N^{\text{th}}$  member of the partial sums sequence of the original series  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$ , you have to enter  $\sum_{n=0}^{\frac{N}{2}-1} \frac{1}{4n^2+6n+2}$  and press [EXPAND] because of the altered "numbering". The result is then  $\frac{\text{Psi } \frac{N}{2} - \frac{-1}{2} - \text{Psi } \frac{N}{2} + 1 + \text{Psi} - \frac{-1}{2} - \text{Psi}(1)}{2}$ . This is the

partial sum  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$  of the series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  but only for  $N = 2, 4, 6, 8, \dots$ . The found function just can't reproduce the partial sums of with an odd number of summands, because the sum from which it was derived creates *two* summands of  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  for each single value of  $n$ . To check this, enter  $\sum_{n=1}^{100} \frac{(-1)^{n+1}}{n}$  and press [->NUM]. You get .688172... in about 2.3 seconds. Now, enter the function with  $N = 100$ , that is

$\frac{\text{Psi } \frac{100}{2} - \frac{-1}{2} - \text{Psi } \frac{100}{2} + 1 + \text{Psi} - \frac{-1}{2} - \text{Psi}(1)}{2}$ , and press

again [->NUM] to get the same result in 0.9 seconds! Do the same with  $N = 101$ , that is, enter  $\sum_{n=1}^{101} \frac{(-1)^{n+1}}{n}$  and press [->NUM]. You get .698073... Enter the function with  $N = 101$ , that is

# Sequences, series and limits with the HP49G - Part 5

$$\frac{\text{Psi} \frac{101}{2} - \frac{-1}{2} - \text{Psi} \frac{101}{2} + 1 + \text{Psi} - \frac{-1}{2} - \text{Psi}(1)}{2}, \text{ and press}$$

again [->NUM]. The result is now .688221...! So the expression with the Psi functions is only good for even N. But nonetheless it is

perfectly right to use it instead of  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$ , that is for N .

For even bigger values of N than those that we already used, the time difference between calculation of  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$  on the one hand and

$$\frac{\text{Psi} \frac{N}{2} - \frac{-1}{2} - \text{Psi} \frac{N}{2} + 1 + \text{Psi} - \frac{-1}{2} - \text{Psi}(1)}{2} \text{ on the other hand,}$$

will be huge because the expression with the sum needs much more time to complete. For example with N = 1000 it takes 19 seconds to calculate the sum, but only 0.9 seconds to calculate with the expression for the N<sup>th</sup> member of the partial sums sequence. And here comes the second advantage of the new created series. If you want to

approximately calculate the limit of the series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  by entering

some partial sum  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$ , where N is a big number, then you have to wait a loooong time. The series converges very, very slowly. On the HP49G each summand  $\frac{1}{n}$  will contribute to the partial sum a

significant amount, an amount that is enough to cause a change in the calculated approximation. For example, let's say that we take  $\sum_{n=1}^{10000} \frac{(-1)^{n+1}}{n}$ . The last summand,  $-\frac{1}{10000}$  is equal to -.0001, a value

that is within the 12 digits approximation of the HP49G. To calculate

the best possible approximation using 12 digits, you should enter  $\sum_{n=1}^{1.E12} \frac{(-1)^{n+1}}{n}$  and press [->NUM]. Imagine how long you would have

to wait. (And how big the power generator for the HP49G must be;-)).

$$\text{But using } \frac{\text{Psi} \frac{1.E12}{2} - \frac{-1}{2} - \text{Psi} \frac{1.E12}{2} + 1 + \text{Psi} - \frac{-1}{2} - \text{Psi}(1)}{2}$$

and pressing [->NUM] you get the result .69314718055 again in 0.9 seconds. By the way, if you apply XQ on .69314718055, you get as result LN(2). Quite silly way to find a limit, I know, but in this case it works.

$$\text{If still you have } \frac{\text{Psi} \frac{N}{2} - \frac{-1}{2} - \text{Psi} \frac{N}{2} + 1 + \text{Psi} - \frac{-1}{2} - \text{Psi}(1)}{2} \text{ on}$$

the stack, enter N = , and press [lim], the HP49G says "Operator not implemented (SERIES)", returns a ?, and sets N as the current VX. So it seems that we don't have any alternative for calculating the

limit of  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$ .

This kind of transformation of a series to another can be used also for

other series. Consider for example  $\sum_{n=1}^{\infty} \frac{\cos \frac{2n}{3}}{n}$ . The HP49G

can't do much with it. But using SPCASES we can find that

$\cos \frac{2n}{3}$  can be  $\frac{-1}{2}$ ,  $\frac{-1}{2}$ , or 1. If we can find that the series

converges, *only if we can do that*, then we can group the summands, so that a new series is generated, that converges to the same limit. This series does converge, even if the HP49G can't find that. Following the same steps we find this new series to be:

# Sequences, series and limits with the HP49G - Part 5

$$\sum_{n=0}^{\infty} \left( \frac{-\frac{1}{2}}{3n+1} + \frac{-\frac{1}{2}}{3n+2} + \frac{1}{3n+3} \right) = -\frac{9}{54} \sum_{n=0}^{\infty} \frac{n+5}{n^3+108n^2+66n+12}.$$

The  $N^{\text{th}}$  partial sum of  $\sum_{n=1}^{\infty} \frac{\cos \frac{2n}{3}}{n}$  can be found by entering

$$\sum_{n=0}^{\frac{N}{3}-1} -\frac{9}{54} \frac{n+5}{n^3+108} \frac{1}{n^2+66} \frac{1}{n+12}$$
 and expanding. The result is:

$$-\frac{\Psi\left(\frac{N}{3}-\frac{2}{3}\right)+\Psi\left(\frac{N}{3}+\frac{1}{3}\right)-2\Psi\left(\frac{N}{3}+1\right)+\Psi\left(-\frac{2}{3}\right)+\Psi\left(-\frac{1}{3}\right)-2\Psi(1)}{6}$$

The HP49G can't find the limit of this expression for  $N$ , but at least we have a fast formula for plugging some big value for  $N$  and see what happens, instead of waiting for results of summations. (For what values of  $N$  can the above expression with Psi functions be used?)

You now may wonder why and how we find the new values between which the index varies. How and why did we find that the last sum

$$-\frac{9}{54} \frac{n+5}{n^3+108} \frac{1}{n^2+66} \frac{1}{n+12} \text{ goes from } n=0 \text{ to } n=\frac{N}{3}-1,$$

where  $N$  stands for the  $N^{\text{th}}$  partial sum  $\sum_{n=1}^N \frac{\cos \frac{2n}{3}}{n}$  ? That is not hard to see. Let's try to find that out considering the series

$$\frac{\cos \frac{2n}{3}}{n} \Big|_{n=3}.$$

The expression  $\cos \frac{2n}{3}$  with  $n = 3, 4, 5, 6, 7, 8 \dots$  can actually

be only  $\cos \frac{2\pi}{3} = 1$  for  $n=3$ ,  $\cos \frac{2\pi}{3} = -\frac{1}{2}$  for

$n = 4$ , and  $\cos \frac{2\pi}{3} = -\frac{1}{2}$  for  $n = 5$ . For  $n = 6$  we have

again 1, and the other values keep repeating themselves. That means

that our series is actually:  $\frac{1}{3} + \frac{-\frac{1}{2}}{4} + \frac{-\frac{1}{2}}{5} + \frac{1}{6} + \frac{-\frac{1}{2}}{7} + \frac{-\frac{1}{2}}{8} + \dots$ . This series does converge, even if the HP49G can't find that, so we accept its convergence as given. Because it does converge we can group all summands for  $n = 3$  to  $n = 5$ , then for  $n = 6$  to  $n = 8$ , and so on in

parentheses:  $\frac{1}{3} + \frac{-\frac{1}{2}}{4} + \frac{-\frac{1}{2}}{5} + \frac{1}{6} + \frac{-\frac{1}{2}}{7} + \frac{-\frac{1}{2}}{8} + \dots$ . Now, each

group, written as a single summand of a series, will give us the series:

$$\sum_{n=0}^{\infty} \frac{1}{3^{n+3}} + \frac{-\frac{1}{2}}{3^{n+3+1}} + \frac{-\frac{1}{2}}{3^{n+3+2}}.$$
 We can see that if we want

for example the first group to be equal to  $\frac{1}{3} + \frac{-\frac{1}{2}}{4} + \frac{-\frac{1}{2}}{5}$ , the second

$\frac{1}{6} + \frac{-\frac{1}{2}}{7} + \frac{-\frac{1}{2}}{8}$ , then we must replace in the first summand of the group  $n$  by  $nSpc \ n + nStart$ , in the second summand of the group  $n$  by  $nSpc \ n + nStart + 1$  and in the third summand of the group  $n$  by  $nSpc \ n + nStart + 2$ , where  $nSpc$  is the number of distinct special outcomes of  $COS \ \frac{2 \ n}{3}$ , and  $nStart$  the starting value of

# Sequences, series and limits with the HP49G - Part 5

$n$  in the original series. When we do this replacement, the new series will start at  $n=0$  always. In the new series the ending value of  $n$  is still  $N$ . But not in the partial sum. Let's say that the original partial sum in the above example goes up to  $n = N$ ,  $N$  being some arbitrary

integer. That means, we have the partial sum 
$$\sum_{n=3}^N \frac{\cos \frac{2n}{3}}{n}$$
.

Since the new partial sum generates three summands of the original sum with only one single value of  $n$ , its ending value must be different. The last replacement for  $n$  in the group of summands will be in general  $n_{\text{Spc}} n + n_{\text{Start}} + n_{\text{Spc}} - 1$ . If we want the last expression to be equal to  $N$ , then we have to solve the equation  $n_{\text{Spc}} n + n_{\text{Start}} + n_{\text{Spc}} - 1 = N$  for  $n$ . This gives us:

$n = \frac{N+1-n_{\text{Start}}}{n_{\text{Spc}}} - 1$ . So if the original partial sum goes up to  $N$ , the

new grouped partial sum has to go up to  $\frac{N+1-n_{\text{Start}}}{n_{\text{Spc}}} - 1$ . In the

above example the original  $N^{\text{th}}$  partial sum is 
$$\sum_{n=3}^N \frac{\cos \frac{2n}{3}}{n}$$
.

The corresponding new grouped partial sum has to be:

$$\sum_{n=0}^{\frac{N-5}{3}} \frac{1}{3n+3} + \frac{-\frac{1}{2}}{3n+3+1} + \frac{-\frac{1}{2}}{3n+3+2},$$
 or if we expand the

summand: 
$$\sum_{n=0}^{\frac{N-5}{3}} \frac{9n^2 + 13n + 12}{54n^3 + 216n^2 + 282n + 120}.$$
 This is the same as

the original partial sum but only for  $N = 5$ ,  $N = 8$ , and so on, that is only for values of  $N$  that correspond to integer number of groups of summands in the original partial sum. (And for  $N = 11$ .) In the above example, for  $N = 5$  we get the first group of summands, for  $N = 8$  the sum of the first and the second, and so on. To see an example with  $N = 5$ :

Original partial sum:

$$\sum_{n=3}^5 \frac{\cos \frac{2n}{3}}{n} = \frac{\cos \frac{2 \cdot 3}{3}}{3} + \frac{\cos \frac{2 \cdot 4}{3}}{4} + \frac{\cos \frac{2 \cdot 5}{3}}{5} = \frac{1}{3} + \frac{-\frac{1}{2}}{4} + \frac{-\frac{1}{2}}{5} = \frac{13}{210}$$

The transformed partial sum:

$$\sum_{n=0}^{\frac{5-5}{3}} \frac{1}{3n+3} + \frac{-\frac{1}{2}}{3n+3+1} + \frac{-\frac{1}{2}}{3n+3+2} = \frac{1}{3 \cdot 0+3} + \frac{-\frac{1}{2}}{3 \cdot 0+3+1} + \frac{-\frac{1}{2}}{3 \cdot 0+3+2} = \frac{1}{3} + \frac{-\frac{1}{2}}{4} + \frac{-\frac{1}{2}}{5} = \frac{13}{210}$$

In general the values of  $N$ , for which the partial sum of the transformed series will be equal to the partial sum of the original series, are:  $N = n_{\text{Spc}} n + n_{\text{Start}} + n_{\text{Spc}} - 1$ , where  $n_{\text{Spc}}$  is the number of special cases,  $n$  some arbitrary integer and  $n_{\text{Start}}$  the starting value of  $n$  in the original series.

All these things are good and nice, but what can we do with them? Well, as we already found out some pages ago, it has advantages to know some analytic closed form of a partial sum, even if the HP49G can't find what happens when  $N \rightarrow \infty$ . (For example, Kalle Fritz



# Sequences, series and limits with the HP49G - Part 5

could immediately find the sum  $\sum_{n=0}^{100} n = 5050$  because he found the

analytic closed form of its partial sum  $\sum_{n=0}^N n = \frac{N(N+1)}{2}$ .)

It is easy to make a small program that takes a series and returns the transformed series along with its  $N^{\text{th}}$  partial sum, if possible to find it. The program should also return the number of special cases because it could be important for some other program that calls it.

```
<<
PUSH
DUP OBJ-> DROP2
{ HOME CASDIR REALASSUME } RCL
RCLVX
-> sum var lo hi exp assmlst vx
<<
expr var lo hi                @Construct part. sum
3 ->LIST 2 ->LIST              @sequence
SPCASES DUP SIZE              @Find special cases
R->I
-> spclst nspc
<<
IF                              @If more than 1 special
  nspc 1. >                     @cases
THEN                            @Then transform series
  sum
  "Converting to sum of
special cases" 1 DISP
  spclst 1
  <<
    OBJ-> DROP OBJ-> DROP2
    OVER nspc * + = SUBST
  >>
  DOSUBS LIST EXPAND var 0      3 ->LIST
  SWAP + { } + DUP 'N' 1 + lo - nspc / 1 -
  EXPAND PUT SWAP ->ALG SWAP ->ALG
ELSE                            @Else return series
  sum DUPDUP ->LST
  3 'N' PUT ->ALG
END
  "Finding Nth partial
sum sequence" 1 DISP EXPAND
  3 ->LIST nspc
  >> vx STOVX { HOME CASDIR } EVAL
  assmlst 'REALASSUME' STO
  >> POP
>>
```

# Sequences, series and limits with the HP49G - Part 5

Store that in TRANSER and let's test it. Enter  $\frac{\cos n \frac{-}{2} + \frac{-}{4}}{n^2}$  and press [TRANSER]. The program returns the list

$$\begin{array}{l} \frac{\cos n \frac{-}{2} + \frac{-}{4}}{n^2} \\ + \\ \frac{2048 \sqrt{2} n^5 + 6400 \sqrt{2} n^4 + 7808 \sqrt{2} n^3 + 4640 \sqrt{2} n^2 + 1348 \sqrt{2} n + 155 \sqrt{2}}{32768 n^8 + 163840 n^7 + 348160 n^6 + 409600 n^5 + 290944 n^4 + 127360 n^3 + 33440 n^2 + 4800 n + 288} \\ 6 \sqrt{2} \text{Psi} \frac{N}{4} - \frac{-1}{2}, 1 - 6 \sqrt{2} \text{Psi} \frac{N}{4} - \frac{-4}{4}, 1 - 6 \sqrt{2} \text{Psi} \frac{N}{4} - \frac{-1}{4}, 1 - 6 \sqrt{2} \text{Psi} \frac{N}{4} + 11 + 6 \sqrt{2} \text{Psi} - \frac{-1}{2}, 1 - 6 \sqrt{2} \text{Psi} - \frac{-3}{4}, 1 - 6 \sqrt{2} \text{Psi} - \frac{-1}{4}, 1 - \sqrt{2}^2 \end{array}$$

192

on stack level 2 and the number of special cases 4 on stack level 1. Seems to work. Notice that the original sum must not contain the name N, as it is used to find the N<sup>th</sup> partial sum.

Now that we have TRANSER, we can use it in CONVERGES?. When we test the Leibniz criterion we can call TRANSER to transform the alternating series and at least give its N<sup>th</sup> partial sum if it can't be found what it converges to. We add a couple of bytes in CONVERGES?:

```
.....
THEN  @Leibniz true, series converges
      series TRANSER + 1
END
.....
```

Now CONVERGES? will use the Leibniz criterion of convergence, and if it finds that the alternating series converges, it will return a list containing the original series, the transformed series, its N<sup>th</sup> partial sum and the number of special cases. Enter for example  $\frac{(-1)^{n+1}}{n}$  and press [ CONVERGES?]. The program rattles and rolls and at the

end it returns the list

$$\begin{array}{l} \frac{(-1)^{n+1}}{n} \\ + \\ \frac{1}{4 n^2 + 6 n + 2} \\ \text{Psi} \frac{N}{2} - \frac{-1}{2} - \text{Psi} \frac{N}{2} + 1 + \text{Psi} - \frac{-1}{2} - \text{Psi}(1) \end{array}$$

2

on stack level 2 and a 1 on stack level 1. We could use the N<sup>th</sup> partial sum along with N = to find its limit, but unfortunately the HP49G seems to be allergic against this partial sum. It says "Operator not implemented (SERIES)" and returns a ?. It also leaves N as the current VX, so if you did this, use STOVX to set your VX again. But there are many cases where the HP49G can find the limit of the partial sum for N , and that's why we include it in the output list.

Now we face another inconvenient question. The whole time we are talking about series whose summand is in some analytic closed form. But what about recurrences? Imagine for example that you want to find what happens if you add all Fibonacci numbers. It's easy to say "hey,

# Sequences, series and limits with the HP49G - Part 5

this just can converge since the sequence itself diverges." Granted, but what about, say, the members of the sequence

$$P(n) = P(n-1) + \frac{3}{(n+1)^2} \quad \{P(0) = 0\} \quad ? \text{ If we have the series}$$

$$P(n) \text{ where } P(n) = P(n-1) + \frac{3}{(n+1)^2} \text{ and } P(0) = 0 \text{ and}$$

$n=0$   
 $n = 0, 1, 2, 3, \dots$ , will it converge or not? And is it possible to construct such sums on the HP49G? Is there any notation that we can use?

Let's see. We start using the simple recurrence sequence  $\{A(n) = A(n-1) + 1 \quad \{A(0) = 0\}\}$ . In its analytic closed form this sequence is  $\{n \quad \{n \quad 0 \quad \}\}$ . That means that if we somehow find

how to write  $A(n)$  on the HP49G, then this must be the same with  $n=0$

the series  $n$ . The first candidate that we check is the function  $|$   $n=0$

(where). As an example, go to the EQW and enter  $X^2 - X|_{X=2}$ . Expanding this, you get 2, the result of  $X^2 - X$  for  $X = 2$ . The stack usage is a bit more flexible as it allows more than one substitutions to be made at once. For example enter  $X + Y^2$ . Now enter the list  $\{X \quad 2 \quad Y \quad 1\}$  and press  $[[]]$ . (This is [red-shift], then [TOOL].) The result is  $1 + 2^2$ , that is the expression  $X + Y^2$  with  $X = 1$  and  $Y = 2$ . Enter  $X + Y^2$  again and then  $\{X \quad Y \quad Y \quad 1\}$ . Now, if you press  $[[]]$  again, you might think that this would substitute first  $X = Y$  resulting in  $Y + Y^2$ , and then  $Y = 1$  resulting in  $1 + 1^2$ . But the result is  $Y + 1^2$ , which shows that this function makes the substitutions isolated from each other. To get the result  $1 + 1^2$ , you should first enter the list  $\{X \quad Y\}$ , use  $|$ , then enter  $\{Y \quad 1\}$  and then use  $|$  again. The same can be done in the EQW. You would enter  $X + Y^2|_{X=Y|_{Y=1}}$ . If you expand

this you get 2, the result of substituting  $X = Y$  in  $X + Y^2$  and then substituting  $Y = 1$  in the expression which has been the result of the first substitution.

Now go to the EQW again and enter  $X(n) + 1|_{\square}$ , move the cursor to the place holder for the substitution and try to finish the expression, entering  $X(n) = 1$ , so that you get  $X(n) + 1|_{X(n)=1}$ . When you try to enter the parentheses of  $X(n) = 1$ , the HP49G doesn't even allow you to do that! This syntax is not allowed and suppressed by the build-in syntax checker. But hey! We have  $\rightarrow$ LST and  $\rightarrow$ ALG. Enter the valid expression  $X(n) + 1|_{X=1}$ . Press  $\rightarrow$ LST to get the list  $\{'X + Y^2' \quad 'X' \quad 1 \quad \}$ . Now we are going to put  $X(n)$  at position 2 in the list. Enter 2, then  $X(n)$  and press put. The list now is:  $\{'X + Y^2' \quad 'X(n)' \quad 1 \quad \}$ . Press  $\rightarrow$ ALG. Now you see  $X(n) + 1|_{X(n)=1}$ , which says that the HP49G doesn't allow you to enter such a thing directly but it can display it correctly. What about using it? Press [EVAL] and you get  $1+1$ , the right result! If you use [EXPAND] on  $X(n) + 1|_{X(n)=1}$  you get the completely expanded expression, that is a nice 2.

Let's try this with a series. We want to enter  $A(n)|_{A(n)=n^2}$ . Enter

$A(n)$ . Press  $\rightarrow$ LST to get the list  $\{n \quad 0 \quad A(n) \quad \}$ . Now we must put the "illegal" expression  $A(n)|_{A(n)=n^2}$  where  $A(n)$  is in the list, at position 4. Enter 4 and then  $A(n)|_{A=n^2}$ . Press  $\rightarrow$ LST to get the list  $\{'A(n)' \quad 'A' \quad n \quad 2 \quad ^ \quad \}$ . Enter 2, then  $A(n)$  and press [PUT]. Now use  $\rightarrow$ ALG to get  $A(n)|_{A(n)=n^2}$ . Press [PUT] and then

# Sequences, series and limits with the HP49G - Part 5

[→ALG] again to get the "illegal" sum  $\sum_{n=0} A(n)|_{A(n)=n^2}$ . It can be displayed! Can it be used? Press [ENTER] to make a copy and then press [arrow down] and you see that taken in the EQW! Change it to

$\sum_{n=0} A(n)|_{A(n)=n^2}$ , press [ENTER] and [EVAL] to get 55, the right

result. Should I start singing the same old song again? I mean the top of the HP49G-pops, "why can't I enter that directly in the EQW?. Why the adventurous input with →LST, →ALG?" Let's go further.

Drop the number 55, change the copy of  $\sum_{n=0} A(n)|_{A(n)=n^2}$  to  $\sum_{n=0}^N A(n)|_{A(n)=n^2}$  and expand. You get the result  $\frac{2N^3 + 3N^4 + N}{6}$  which is also correct.

Now we use the recurrence  $P(n) = \frac{P(n)}{2} \{P(0) = 1\}$  and try to

somehow write the series  $\sum_{n=0} P(n)$ . Unfortunately there is good and

bad news. The good news is that we can write  $\sum_{n=0} P(n)|_{P(n)=\frac{P(n-1)}{2}}|_{P(0)=1}$  using →LST and →ALG more than once. The bad news is that evaluating or expanding this, doesn't do the right thing. So we must look for another way to denote such series. Since we used lists to denote recurrence sequences, let's do that also for such series. The program CONVERGES? can then check the type of argument. If it finds a list, then it will assume a recurrent series and try to use the program RCR→ANL to turn the summand to its analytic closed form. We use the notion  $\{\{\text{recurrence} \{\text{initialConditions}\}\}$ .

is the series which has as summand the general recurrence sequence

member like for example  $\sum_{n=0} P(n)$ . The inner list  $\{\text{recurrence} \{\text{initialConditions}\}\}$  is exactly the same like what we used for denoting recurrence sequences, like for example  $P(n) = \frac{P(n-1)}{2} \{P(0) = 1\}$ . The whole thing in this example is

then  $\sum_{n=0} P(n) = \frac{P(n-1)}{2} \{P(0) = 1\}$ .

We add code at the start of CONVERGES? that checks if the argument is a list and tries to find the analytic closed form in this case.

```
<<
IF                                @IF we have a recurrence series
  DUP TYPE 5 ==
THEN
  OBJ→ DROP
  RCR→ANL                        @Try to find the analytic closed form
  HEAD SWAP                      @And re-construct the sum
  →LST 4 ROT
  PUT
END
PUSH DUP OBJ→ @Rest of code exactly like it was
...
```

We test our example. Enter  $\sum_{n=0} P(n) = \frac{P(n-1)}{2} \{P(0) = 1\}$  and press [CONVERGES?]. After deepest thoughts the HP49G returns 2 on stack level 2 and 1 on stack level 1. The series converges to 2. If you enter  $P(n) = \frac{P(n-1)}{2} \{P(0) = 1\}$  and use the program

# Sequences, series and limits with the HP49G - Part 5

RCR->ANL, then the result is  $1 - \frac{1}{2} \sum_{n=0}^{\infty} \{n \cdot 0 + \dots\}$ . That shows that the series was the recurrence form of the geometric series  $\sum_{n=0}^{\infty} \frac{1}{2} \cdot 2^n$ , which converges to 2. (Look also at the geometric series, page 5-8 of this marathon.) It works!

Another example: Enter  $a(n) \{a(n) = a(n-1) + n \mid \{a(0) = 0\}\}$  and press [CONVERGES?]. The HP49G returns + and 0 to tell you that the series doesn't converge. Indeed the analytic closed form of the series is  $\sum_{n=0}^{\infty} \frac{n^2 + n}{2}$  which goes to +.

Another example which also gives us some additional insight about the complex phenomenon called "Modes" on the HP49G. In real rigorous mode enter the recurrence series:

$$P(n) = \frac{P(n-1) + P(n-2)}{3} \quad \{P(0) = 0 \mid P(1) = 1\}$$

Press [ENTER] twice as we are going to need the same thing again later. Now, press [CONVERGES?] and wait for a loong time until the HP49G returns the result  $\frac{13 + 4\sqrt{13}}{13}$  on stack level 2 and 1 on stack level 1. Hurrah! We have all possible reasons to celebrate, except that the result is... wrong!!!. Let's follow what happened. Drop the 1 and press [arrow right] to swap stack levels 1 and 2. Press [OBJ->] and drop the 2 from stack level 1. What you have now on stack level 1 is a recurrence sequence:

$$P(n) = \frac{P(n-1) + P(n-2)}{3} \quad \{P(0) = 0 \mid P(1) = 1\}$$

The program CONVERGES? used first RCR->ANL to convert this sequence to its closed analytic form. So press [RCR->ANL]. The result is:

$$\frac{-(3\sqrt{13})}{13} - \frac{-1 + \sqrt{13}}{6}^n + \frac{3\sqrt{13}}{13} \frac{1 + \sqrt{13}}{6}^n \{n \cdot 0 + \dots\}$$

which is OK. That means, our series can be written as:

$$\sum_{n=0}^{\infty} \left( \frac{-(3\sqrt{13})}{13} - \frac{-1 + \sqrt{13}}{6}^n + \frac{3\sqrt{13}}{13} \frac{1 + \sqrt{13}}{6}^n \right)$$

Press [OBJ->], [DROP2] to put the expression for the summand,  $\frac{-(3\sqrt{13})}{13} - \frac{-1 + \sqrt{13}}{6}^n + \frac{3\sqrt{13}}{13} \frac{1 + \sqrt{13}}{6}^n$ , on stack level

1. Press [SWAP], [->LST], enter a 4, press [ROT], [PUT], [->ALG] to get the series on the stack. Press [DUP] to make a copy of this series. Switch to real rigorous mode and expand, to get the wrong result  $\frac{13 + 4\sqrt{13}}{13}$ . (The current VX has been changed to n, so use

STOVX to restore your VX. Also, n has been added to REALASSUME, so if you don't want it there, use the command UNASSUME.) Switch to complex rigorous mode, swap and expand again. This time you get the result 3 which is correct. (The current VX has been re-changed to n, and n has been added to REALASSUME again.)

What is the reason for this shitty behaviour? Well, switch back to real mode and enter the expression  $-\frac{-1 + \sqrt{13}}{6}^n$ . Press [ENTER] to copy it to stack level 2. Now, enter the list {n 1} and press [[]]. The

# Sequences, series and limits with the HP49G - Part 5

result is  $\frac{e^{1 \text{ LN}(-1+\sqrt{13})}}{e^{1 \text{ LN}(6)}}$ . Press [TEXPAND] and you get  $\frac{-1+\sqrt{13}}{6}$ . That

is, the HP49G has found out that  $-\frac{-1+\sqrt{13}}{6}^1 = \frac{-1+\sqrt{13}}{6}$  !!!

Ouch! Swap and switch to complex mode. Enter again {n 1} and

press []. Now you have the result  $e^{1 \text{ LN} -\frac{-1+\sqrt{13}}{6}}$ . Pressing [TEXPAND] you get  $-\frac{-1+\sqrt{13}}{6}$ , the correct result. Switch to real

rigorous mode again. Enter X and press [LN]. You get  $\text{LN}(|X|)$ . See what happened in our example? The expression  $-\frac{-1+\sqrt{13}}{6}^n$ ,

which appears in the series, has been converted to  $\frac{e^{n \text{ LN}(-1+\sqrt{13})}}{e^{n \text{ LN}(6)}}$ . This

is equivalent to  $e^{n \text{ LN}(-1+\sqrt{13})-n \text{ LN}(6)}$ , and the last expression is

equivalent to  $e^{n \text{ LN} \frac{-1+\sqrt{13}}{6}}$ . That shows us that the HP49G in real rigorous mode, wanted to use the absolute value of  $-\frac{-1+\sqrt{13}}{6}$  as

argument for the function LN. So it constructed  $e^{n \text{ LN} \frac{-1+\sqrt{13}}{6}}$  which is equal to  $\frac{-1+\sqrt{13}}{6}^n$ , which in this case was the completely wrong thing. No further comments ;-)

What can we do to avoid this? Well, we could switch to complex mode before we try to use the command EXPAND with the series

$\sum_{n=0}^{\infty} \frac{-(3 \sqrt{13})}{13} - \frac{-1+\sqrt{13}}{6}^n + \frac{3 \sqrt{13}}{13} \frac{1+\sqrt{13}}{6}^n$ . We can also add a couple of bytes to the program CONVERGES?.

```
.....
-> series ivar lo hi
sumnd assm1st vx      @Store in locals
<<
IF
  "Trying built-in
methods..." 1 DISP
-103 SF
series EXPAND      @Try expanding
-103 CF
vx STOVX
.....
```

Let's hope that these changes of the program will not cause other problems. Drop all objects until the remaining copy of the series

$$P(n) = \sum_{n=0}^{\infty} \frac{P(n-1)+P(n-2)}{3} \quad \{P(0)=0 \quad P(1)=1\}$$

comes to stack level 1. Press CONVERGES? and wait until you get the results 3 and 1. It works. But who knows what other unexpected problems this changes of the program might cause.

As a last example let's look at the series  $\sum_{n=0}^{\infty} \frac{1}{n(n-1)}$ . We might think

that this series diverges because for  $n=0$  and  $n=1$  the expression  $\frac{1}{n(n-1)}$  goes to infinity. But as we will see in this examples, infinity can be a source of quite unexpected surprises. Enter the series, press [ CONVERGES?] and after some seconds you get a -1 on stack level 2 and a 1 on stack level 1, which means "yes, the series converges to -1". How can that be? Well, since we know that the series converges,

# Sequences, series and limits with the HP49G - Part 5

we can group summands in parentheses as we wish. We group the first two summands:

$$\frac{1}{n(n-1)} = \frac{1}{n(n-1)} \Big|_{n=0} + \frac{1}{n(n-1)} \Big|_{n=1} + \frac{1}{n(n-1)} \Big|_{n=2}$$

We keep in mind that for every summand with index  $n$  the next summand will have the index  $n+1$ . So we can write for the first two summands:

$$\begin{aligned} \frac{1}{n(n-1)} \Big|_{n=0} + \frac{1}{n(n-1)} \Big|_{n=1} &= \\ \frac{1}{n(n-1)} + \frac{1}{(n+1)((n+1)-1)} \Big|_{n=0} &= \\ \frac{2}{n^2-1} \Big|_{n=0} &= -2 \end{aligned}$$

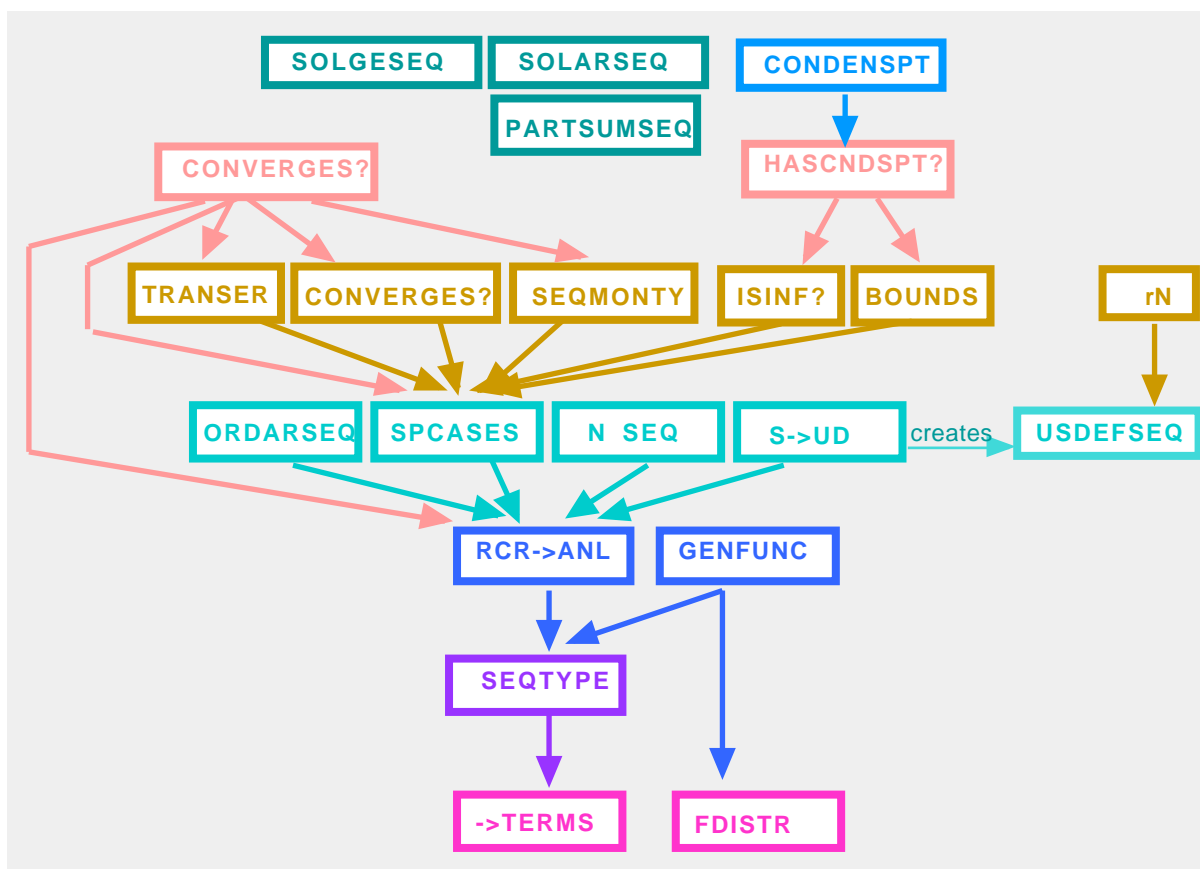
Enter  $\frac{1}{n(n-1)}$ , press [ENTER] to make a copy, enter  $n = n + 1$  and press [SUBST].

Press [+] and [EXPAND] to get  $\frac{2}{n^2-1}$ .

This expression is the sum of the first two summands for  $n = 0$ . That means, enter  $n = 0$  and press [EXPAND] to see that the sum of the first two summands is  $-2$ .

Enter the rest of the series,  $\frac{1}{n(n-1)}$ , and press [CONVERGES?]. The result is 1 (the limit) and 1. That means that the series converges to  $-2 + 1 = -1$ . Unexpected perhaps, but correct.

Let's take a look again at our programs and how they depend on each other. It is crowded around the programs SPCASES and RCR->ANL. Looks like they are the heart of the building, doesn't it? Notice also that CONVERGES? uses other programs. Actually this program can be made much better. (It is intentionally written that way, so that the interested user might think how to make it better ;-)) For example, many of the programs that it calls, call SPCASES themselves. Look if it is really necessary to do that. Also notice that CONVERGES? calls CONVERGES? many times. Is that really necessary? Can we avoid that?



# Sequences, series and limits with the HP49G - Part 5

We move on to plotting series. Actually when we say that we plot a series, we don't mean that we plot the series itself. This would mean to plot just a single point, the limit of the series, and this only if it exists. What we rather mean when we speak about plotting a series of

the form  $\text{summand}(n)$ , is that we plot the points

$\text{start}, \text{summand}(n), \text{start} + 1, \text{summand}(n),$   
 $\text{start} + 2, \text{summand}(n),$  and so on. We plot the *partial sums*

$\sum_{n=\text{start}}^N \text{summand}(n)$ , for  $N = \text{start} + 1, \text{start} + 2, \text{start} + 3, \dots$ . This can be easily done on the HP49G using the plot type function. Let's have an example. We plot the series  $\sum_{n=1}^N \frac{1}{\sqrt{n}}$ . Press first simultaneously [red-shift] and [F4]. Select plot type "Function". In the input field "EQ:", enter the partial sum  $\sum_{n=1}^N \frac{1}{\sqrt{n}}$ . Enter N as independent variable.

This means that we actually plot the expression  $F(N) = \sum_{n=1}^N \frac{1}{\sqrt{n}}$  as a function of N. Enter 1 for both horizontal and vertical axes ticks, and uncheck the option "\_Pixels". Now press simultaneously [red-shift] and [F2]. Enter "H-View:" from 0 to 20, "V-View:" from -2 to 8, enter 1 for "Indep Low:" and 20 from "High:", enter 1 for "Step:" and uncheck the option "\_Pixels". Now press the menu keys [ERASE] and [DRAW] and watch how the HP49G plots the series. When the plot is ready you can start tracing it using the menu keys [TRACE] and [(X,Y)]. Perhaps you have already noticed that the HP49G gets more and more slowly while it plots  $\sum_{n=1}^N \frac{1}{\sqrt{n}}$  for higher and higher values of N. This is not hard to understand. The first point

has the coordinates

$X = 1$  and

$Y = \sum_{n=1}^1 \frac{1}{\sqrt{n}}$ . The

Y-coordinate is calculated quite fast, as the HP49G has to find

$\frac{1}{\sqrt{1}}$ . The second point has the

coordinates  $X = 2$  and  $Y = \sum_{n=1}^2 \frac{1}{\sqrt{n}}$ . The calculation of the Y-

coordinate takes now a little more time, as the HP49G has to find

$\frac{1}{\sqrt{1}} + \frac{1}{\sqrt{2}}$ . When we are at  $X = 20$  the HP49G has to calculate the

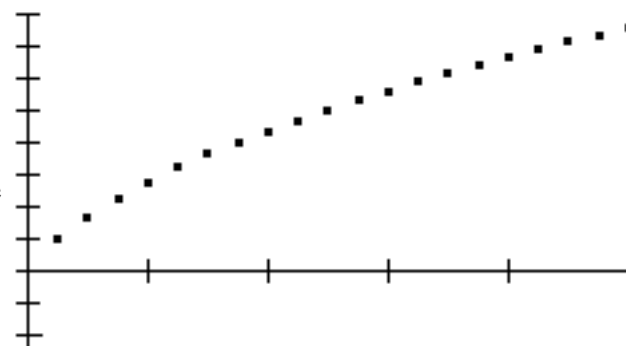
sum  $\frac{1}{\sqrt{1}} + \frac{1}{\sqrt{2}} + \dots + \frac{1}{\sqrt{20}}$ , which of course needs a considerable

amount of time. The same happens when tracing. The HP49G feels not very responsive when we trace for higher values of the X-coordinate. Notice also how the plot indicates that this series doesn't converge. You can try to plot for higher values of the X-coordinates. The overall looking of the plot remains the same. The sequence of plotted points doesn't get "flat".

Another example? OK, graphics is always fun. Let's do  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$ .

Enter  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$  as the expression to be plotted and .1 as vertical tick

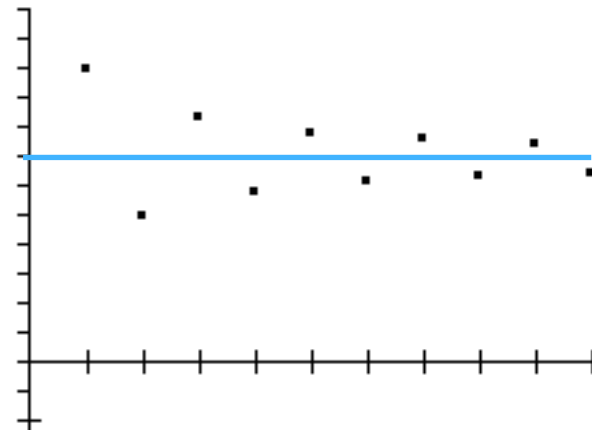
distance. "H-View:" from 0 to 10, "V-View:" from -2 to 1.2, "Low:" and "High:" are 1 and 10 respectively. Press [ERASE], [DRAW]. Notice how the plot indicates that we have convergence. As N gets higher and higher, the partial sums points come nearer and





# Sequences, series and limits with the HP49G - Part 5

neener to  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$  which is the limit of this series as we already found out. It will be interesting to put another series on the same plot, the series to which  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$  has been transformed, when the program CONVERGES? has performed the Leibniz test of convergence. Return to the stack, enter  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$  and press [TRANSE]. When the program finishes, press [BACKSPACE] to drop the returned 2 (number of special cases) and then enter 3 GET to get the third element of the returned list. This is the analytic closed form of the transformed sequence. It returns the same results like



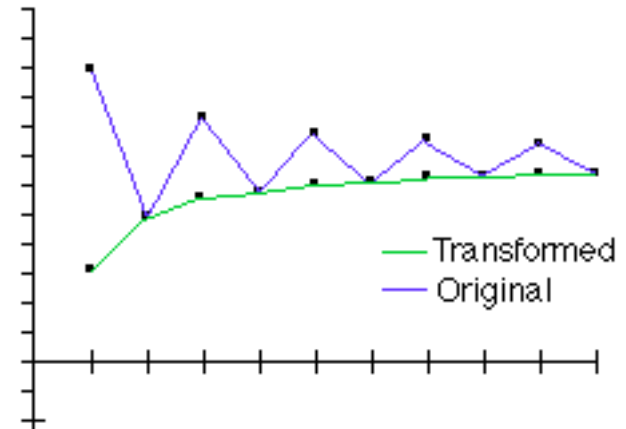
$\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$ , but only when  $N = 2, 4, 6, \dots$  and also for  $N$ . We already have seen the reasons for this. Now you have  $\text{Psi}\left(\frac{N}{2}\right) - \frac{-1}{2} - \text{Psi}\left(\frac{N}{2} + 1\right) + \text{Psi}\left(-\frac{-1}{2}\right) - \text{Psi}(1)$  on stack level 1.

Enter STEQ, and then DRAW. Press [arrow left] to go to the plotting environment. Oh no, don't search for coloured lines and for annotations. I only put them there for focusing the two series. The points connected with the blue lines belong to the original series. Their partial sums were  $\sum_{n=1}^N \frac{(-1)^{n+1}}{n}$ . Those points that are connected with the green lines are the points of the transformed series and they have the

partial sums that TRANSER found:  $\text{Psi}\left(\frac{N}{2}\right) - \frac{-1}{2} - \text{Psi}\left(\frac{N}{2} + 1\right) + \text{Psi}\left(-\frac{-1}{2}\right) - \text{Psi}(1)$ . This partial sum belongs

to the transformed series that also found:

$\sum_{n=0}^N \frac{1}{4n^2 + 6n + 2}$ . You see that both series converge to the same limit and that they have the same partial sum for  $N = 2, 4, 6, \dots$ .



Speaking about partial sums, if the HP49G can find an analytic closed expression for the partial sum of a sequence, then we can plot the partial sum instead of the series itself,

to speed up plotting and tracing. For example consider the series  $\sum_{n=1}^N n$ .

If you enter  $\sum_{n=1}^N n$  and press [EXPAND], then the HP49G returns

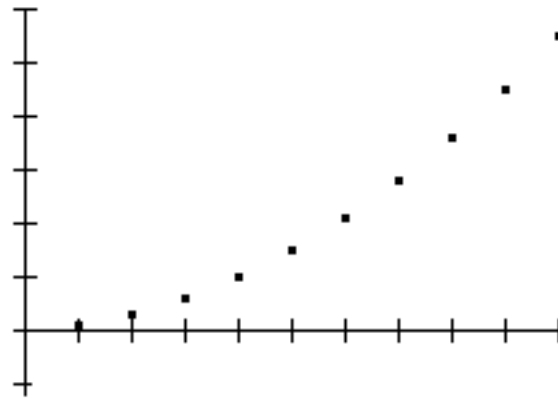
$\frac{N^2 + N}{2}$ . You can use this expression instead of  $\sum_{n=1}^N n$  to plot the series.

If you plot it you get exactly the same plot like for the series. Don't confuse this with the previous example. In this example the  $N^{\text{th}}$  partial sum is valid for any value of  $N$ . In the previous example we first transformed the original series to another series, whose partial sum is the same like that of the original series for some distinct cases and for  $N$ . We did that in order to help the HP49G find if the series converges because it couldn't do anything with the original series. The advantage is now that the plot is much faster. Also, if you trace this

# Sequences, series and limits with the HP49G - Part 5

plot you will find that the HP49G is much more responsive now, as it doesn't have to calculate sums for every plotted point that is traces. So whenever you have to plot a series

summand(n),  
 $n = \text{start}$   
 enter first its partial  
 $\sum_{n=\text{start}}^N \text{summand}(n)$ ,  
 and try to find its  
 analytic closed form. If  
 you succeed, plot this  
 instead of  
 $\sum_{n=\text{start}}^N \text{summand}(n)$  to  
 speed things up.



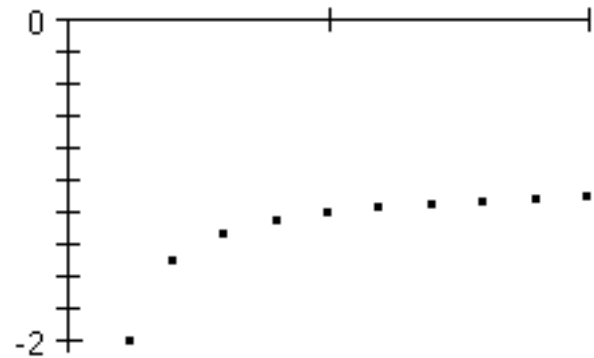
Sometimes the series itself will not be plotted. Consider the series

$\sum_{n=0}^N \frac{1}{n(n-1)}$ . If you try to plot this, the HP49G will not draw a single point. Why? Well, when the HP49G plots, it numerically evaluates the expression to plot for values between the plot parameters "High:" and "Low:". For example, when it plots the tenth point of  $\sum_{n=0}^N \frac{1}{n(n-1)}$ , it

tries to numerically find out hat the value of  $\sum_{n=0}^{10} \frac{1}{n(n-1)}$  is. If you

enter  $\sum_{n=0}^{10} \frac{1}{n(n-1)}$  and press [->NUM], then the HP49G errors out because a division by 0 occurs for  $n = 0$ . But as we have seen the value of this this series exists for any  $n \neq 1$ . We can only plot this series if we find its partial sum as an analytic expression. Enter

$\sum_{n=0}^N \frac{1}{n(n-1)}$  and  
 expand. The result is  
 $-\frac{N+1}{N}$ , which can be  
 plotted like any other  
 function of  $N$ . Only  
 when  $N = 0$  the series  
 is  $-$ , that is only  
 this point can't be  
 plotted.

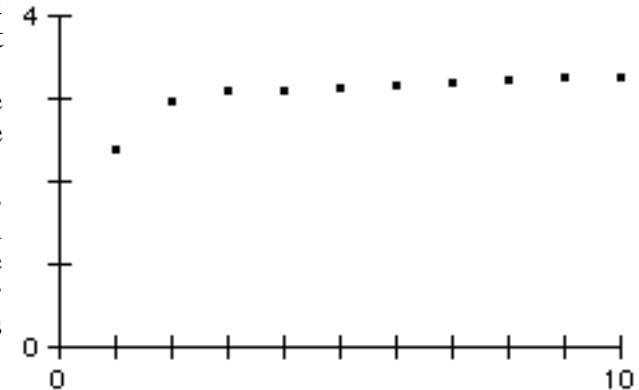


Plotting series can also be helpful for the investigation of the behaviour of series regarding their convergence. We use the example

$\sum_{n=1}^N \frac{e^{\sin \frac{2n}{6}}}{n^2}$  for which the program CONVERGES? can't find if it

converges. Enter  $\sum_{n=1}^N \frac{e^{\sin \frac{2n}{6}}}{n^2}$  and then STEQ to store that as the current expression to be plotted. Let it plot and you can see that it looks like converging to about 3.3. Having "guessed" that the series

converges we can go further and find exactly what it converges to. Because we assume that the series converges, we can group its summands in parentheses as we wish. So we enter the series



# Sequences, series and limits with the HP49G - Part 5

$\sum_{n=1}^{\infty} \frac{e^{\sin \frac{2n}{6}}}{n^2}$  and press [TRANSER]. After several seconds the program returns the huge list on stack level 2 and 6 on stack level 1.

$$\frac{\sum_{n=1}^{\infty} \frac{e^{\sin \frac{2n}{6}}}{n^2}}{\frac{(839808 n^{10} + \dots) e^{\frac{\sqrt{3}}{2}^2} + \dots}{(15116544 n^{12} + \dots) e^{\frac{\sqrt{3}}{2}}} - \frac{6 e^{\frac{\sqrt{3}}{2}} \text{PSI} \left( \frac{N}{6} - \frac{-1}{2}, 1 \right) + \dots}{216 e^{\frac{\sqrt{3}}{2}}}}$$

Drop the 6, press [OBJ->] and drop the element count. Now you have

$$6 e^{\frac{\sqrt{3}}{2}} \text{PSI} \left( \frac{N}{6} - \frac{-1}{2}, 1 \right) + \dots$$

the huge expression  $-\frac{6 e^{\frac{\sqrt{3}}{2}} \text{PSI} \left( \frac{N}{6} - \frac{-1}{2}, 1 \right) + \dots}{216 e^{\frac{\sqrt{3}}{2}}}$  on stack level 1.

This is the expression that converges to the same limit like the original series when N . Press [arrow right] to swap stack levels 2 and 1. Stack level one now contains the huge series

$$(839808 n^{10} + \dots) e^{\frac{\sqrt{3}}{2}^2} + \dots$$

$\frac{(839808 n^{10} + \dots) e^{\frac{\sqrt{3}}{2}^2} + \dots}{(15116544 n^{12} + \dots) e^{\frac{\sqrt{3}}{2}}}$ . This series converges to the

same (assumed) limit like the original series. Press [CONVERGES?] and wait some seconds. The program returns the limit:

$$\frac{6 e^{\frac{\sqrt{3}}{2}} \text{PSI} \left( -\frac{1}{2}, 1 \right) + 6 \text{PSI} \left( -\frac{2}{3}, 1 \right) + 6 e^{\frac{\sqrt{3}}{2}^2} \text{PSI} \left( -\frac{1}{3}, 1 \right) + 6 \text{PSI} \left( -\frac{5}{6}, 1 \right) + 6 e^{\frac{\sqrt{3}}{2}^2} \text{PSI} \left( -\frac{1}{6}, 1 \right) + 2 e^{\frac{\sqrt{3}}{2}}}{216 e^{\frac{\sqrt{3}}{2}}}$$

on stack level 2 and 1 on stack level 1. The series converges. Drop the 1 and press [ENTER] to make a copy of the limit. Press [->NUM]. The result is 3,37503588204, quite near that value which we observed by looking at the plot of the series. So there is very strong

evidence that the original series  $\sum_{n=1}^{\infty} \frac{e^{\sin \frac{2n}{6}}}{n^2}$  converges.

Some of the examples of series that we had so far, contained expressions like for example  $(-1)^n$  or

$\sin \frac{n}{2}$ . The HP49G can't deal with such series out of the box,

and that's why we used SPCASES, to just replace such expressions with all their possible outcomes. For example the HP49G can't deal

with  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$ , but calling SPCASES from CONVERGES? we can

find that the series converges. It should be mentioned however that for some series the time consuming procedures of SPCASES can be

avoided. Consider for example the series  $\sum_{n=0}^{\infty} (-1)^n$ . This series doesn't

converge. But the HP49G can find an expression for its N<sup>th</sup> partial sum. Enter  $\sum_{n=0}^N (-1)^n$  and expand. The HP49G returns  $-\frac{e^{i(N+i)} - 1}{2}$ .

Let's have another example. We consider

$\sum_{n=0}^{\infty} \sin \frac{n}{3}$ . This series also doesn't converge.

But the HP49G can find an expression for its N<sup>th</sup> partial sum, if we use complex exponentials instead of

# Sequences, series and limits with the HP49G - Part 5

trigonometric functions. Enter  $\sum_{n=0}^N \sin \frac{n}{3}$  and press [EXPLN].

$$-i e^{i \frac{n}{3}} + i$$

The HP49G answers with  $\sum_{n=0}^N \frac{2 e^{i \frac{n}{3}}}{2}$ . If you expand now the HP49G will return the same sum. But press [LIN] to linearize the exponentials. The result is  $\sum_{n=0}^N -\frac{i}{2} e^{-i \frac{n}{3}} + 2 i \frac{n}{3} + \frac{i}{2} e^{-i \frac{n}{3}}$ .

Expanding this you get the  $N^{\text{th}}$  partial sum of the series,

$$\frac{(-i + \sqrt{2}) e^{\frac{(i N + i)}{3}} - 2 \sqrt{3} e^{\frac{(i N + i)}{3}} + i + \sqrt{3}}{4 e^{\frac{(i N + i)}{3}}}$$

. Now we return to trigonometric functions. Switch to real mode, press [ENTER] to make a copy, and press [RE] to get the real part of the expression. Press [TCOLLECT] and [EXPAND] to get the result

$$\frac{\sin \frac{(N+1)}{3} + \sqrt{3} \cos \frac{(N+1)}{3} - \sqrt{3}}{2}$$

. Swap and press [IM], [TCOLLECT], [EXPAND] to get 0, which shows that for the  $N^{\text{th}}$  partial sum of the sequence we have the relation:

$$\sum_{n=0}^N \sin \frac{n}{3} = -\frac{\sin \frac{(N+1)}{3} + \sqrt{3} \cos \frac{(N+1)}{3} - \sqrt{3}}{2}$$

This technique of using EXPLN to turn trigonometrics and powers to complex exponentials, then find the partial sum, and then return to the real domain and find the real and imaginary parts, can be used in quite a few cases. But there are also many many cases, in which it will not work.

Regarding alternating series that converge, like  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$ , we can ask if the series of the absolute values of the summands also converges. If it does, then the series *converges absolutely*. The series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  for example, converges but not absolutely, since the series

$$\sum_{n=1}^{\infty} \left| \frac{(-1)^{n+1}}{n} \right| = \sum_{n=1}^{\infty} \frac{1}{n}$$

diverges to  $+\infty$ . Now that we have

CONVERGES? it is easy to make a program that checks series for absolute convergence.

```
<<
IF                               @IF we have a recurrence series
  DUP TYPE 5 ==
THEN
  OBJ-> DROP
  RCR->ANL                       @Try to find the analytic closed form
  HEAD SWAP                      @And re-construct the sum
  ->LST 4 ROT
  PUT
END
PUSH { HOME CASDIR REALASSUME } RCL
OVER OBJ-> 4 DROPN
SWAP UNASSUME SWAP ASSUME DROP
OVER ->LST 4 OVER 4 GET ABS EXPAND PUT ->ALG
{ HOME CASDIR } EVAL SWAP 'REALASSUME' STO
POP
SWAP CONVERGES?
ROT CONVERGES?
ROT AND
>>
>>
```

Store that in ABSCONVERGES? Let's make some examples again.

# Sequences, series and limits with the HP49G - Part 5

Enter  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  and run ABSCONVERGES? After 3.5 minutes in agony the HP49G returns on stack level 3 the list

$$\left[ \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}, \sum_{n=0}^{\infty} \frac{1}{4n^2 + 6n + 2}, \text{Psi} \left( \frac{N}{2} - \frac{-1}{2}, 1 \right) - \text{Psi} \left( \frac{N}{2} + 1, 1 \right) + \text{Psi} \left( -\frac{-1}{2}, 1 \right) - \text{Psi}(1) \right], \text{ on stack}$$

level 2 + , and on stack level 1 a 0. The series is not an absolutely converging one. Its convergence behaviour results in the list on stack level 3, while the convergence behaviour of the series of the absolute value results in + .

Enter  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2}$  and run ABSCONVERGES? again. After 4.5 minutes the results are, on stack level 3 the list

$$\left[ \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2}, \sum_{n=0}^{\infty} \frac{4n+3}{16n^4 + 48n^3 + 52n^2 + 24n + 4}, 6 \text{Psi} \left( \frac{N}{2} - \frac{-1}{2}, 1 \right) - 6 \text{Psi} \left( \frac{N}{2} + 1, 1 \right) + 6 \text{Psi} \left( -\frac{-1}{2}, 1 \right) - 2 \right], \text{ on stack}$$

level 2  $\frac{2}{6}$  and on stack level 1 a 1. This series converges absolutely. Press [ROT], [2], [GET] and then [ CONVERGES?], to find that the

limit of the first series is  $\frac{6 \text{Psi} \left( -\frac{-1}{2}, 1 \right) - 2}{24}$ . The two limits are not the same but that doesn't matter. They both converge and that's enough for calling  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2}$  absolutely convergent.

We stay at the series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$  to examine another important property of series. This series converges, as we have already seen, to LN(2). Now we consider the series

$$1 + \frac{1}{3} - \frac{1}{2} + \frac{1}{5} + \frac{1}{7} - \frac{1}{4} + \dots$$

which is produced from the series  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$  through rearrangements of the summands. The new series

can be written as  $\sum_{n=1}^{\infty} \frac{1}{2(2n-3)} + \sum_{n=1}^{\infty} \frac{1}{2(2n-1)} - \sum_{n=1}^{\infty} \frac{1}{(2n)}$ . What does this expression converge to? You might think that since we only rearranged summands and since addition is commutative, the new series has to converge to LN(2) too. But this is not true. The program

CONVERGES? can't find even if this series converges at all. But still we can use the (dangerous) method of "plugging the numbers". Remember the advantages of the N<sup>th</sup> partial sum? OK, enter  $\sum_{n=1}^N \frac{1}{2(2n-3)} + \sum_{n=1}^N \frac{1}{2(2n-1)} - \sum_{n=1}^N \frac{1}{(2n)}$  and press [EXPAND].

After some seconds the HP49G returns the result  $\text{Psi} \left( N+1, -\frac{1}{4} \right) + \text{Psi} \left( N+1, -\frac{3}{4} \right) -$

$$\frac{\text{Psi} \left( 1, -\frac{1}{4} \right) + \text{Psi} \left( 1, -\frac{3}{4} \right) + 2 \text{Psi}(N+1) - 2 \text{Psi}(1)}{4}.$$

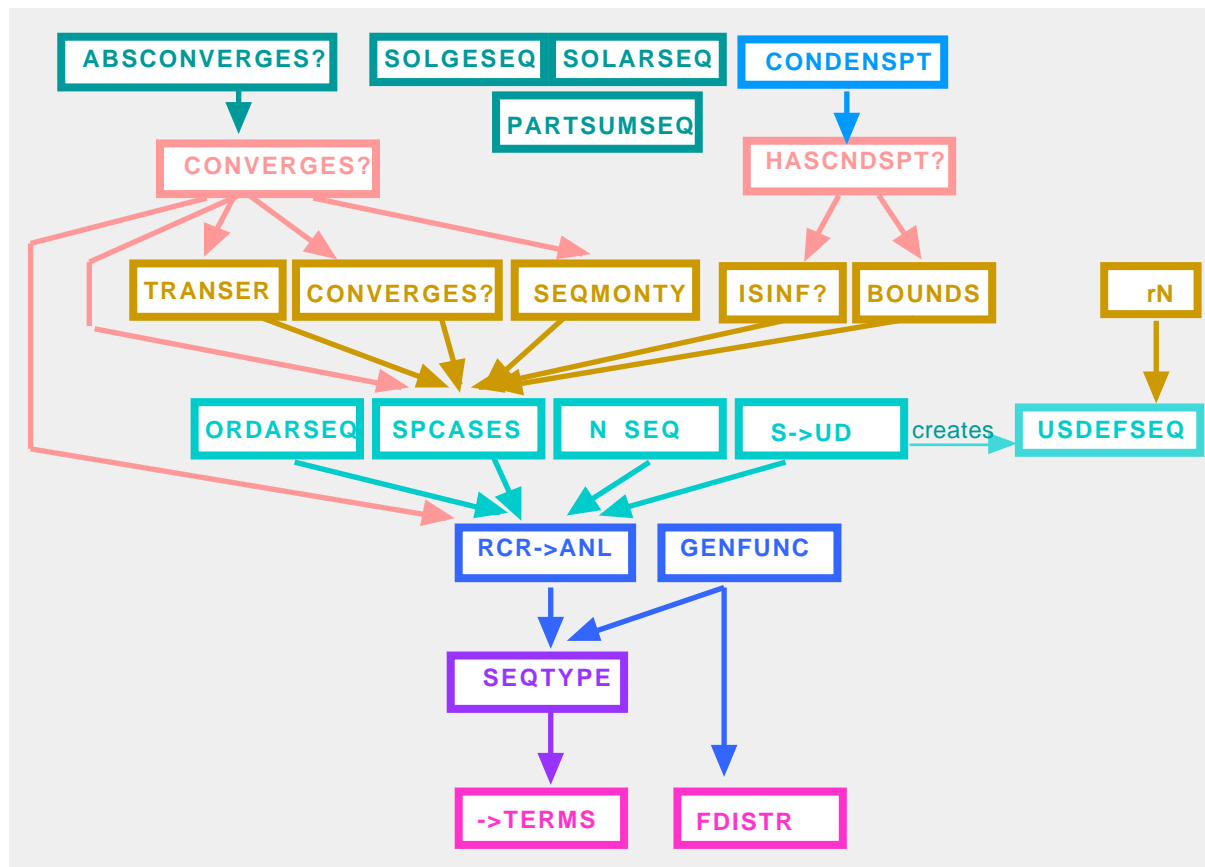
Enter N = 1E100 and press [SUBST], [EXPAND] to get 1.03972077075. Press [ENTER] to make a copy of the result. If you now use XQ to convert the result to something "more algebraic", you get  $\frac{25469}{24496}$ . Doesn't look like a candidate for a limit, so let's try something different. Drop the ratio and think about the limit of the

# Sequences, series and limits with the HP49G - Part 5

original series, that was  $\text{LN}(2)$ . Should we suspect that this  $\text{LN}(2)$  is a factor of the limit of the rearranged series? Let's see. Enter 2. and press [LN]. Press [/]. Aha! The result is 1.49999999987 which can be converted to  $\frac{3}{2}$  with XQ and which shows that the rearranged series converges to  $\frac{3}{2} \text{LN}(2)$ . In fact you could achieve *any* limit or *no limit at all* by just doing the proper rearrangement. Such series, the convergence of which depend on rearrangements are *conditionally convergent series*, and their convergence behaviour offers pathologically beautiful examples of convergence investigations. Series can be either absolutely or conditionally convergent but not both.

That was it for this part. We'll continue in the sixth and last part with limits and some other properties of functions, like for example how we can find if some function is continuous at some point. Let's hope that no ancient greek sophist will puzzle us with unnecessary problems and that no ancient arab game inventor will drive us bankrupt ;-). As always I include the table of our program collection so far.

$\sum_{n=1}^{\infty} \frac{1}{n}$  GREETINGS(n),  
Nick.

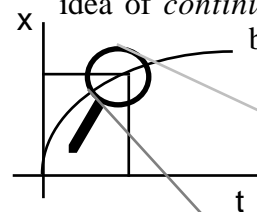
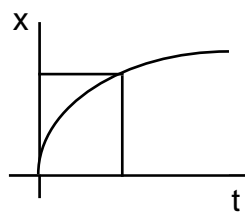


# Sequences, series and limits with the HP49G - Part 6

Hi again!

After a long and sometimes exhausting journey, we come to the 6<sup>th</sup> and last route of this marathon, in which we'll take a look at limits of functions. This will be of great help for the next marathon, the calculus marathon, which I think, is one of the most interesting for many people.

Let's start with a crossover to physics, and consider some time dependent function, like for example the function that describes the one-dimensional motion of a body. The *model* that we use when we talk about such functions in the classical physics approximation is the very basic idea of *continuity*. The x-coordinate of the

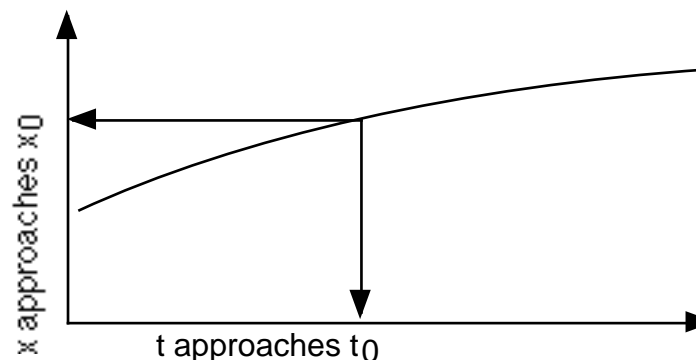


body is a *continuous function* of time, which itself is also continuous. That means, very easily but also very unprecisely spoken, that if you take a loupe and consider the ideally drawn motion

graph, you will always get a *curve*, and not some *discrete* sequence of points, no matter how in. Notice that this is *assume* without any nobody on this world can the x-coordinate of some this remind you of some numbers?) Now, the first this meaning in precise what we have seen already,

strongly you zoom something that we experimental proof, since really take measurements of body for *all* times. (Does basic property of real question is, how can we put words? First, remember when we were considering condensation points of discrete sequences, and try to imagine two *continuous* sequences of points, of which one represents time, and the other represents the x-coordinate in space. If some condensation point of the first sequence corresponds to some condensation point of the second, we talk about a

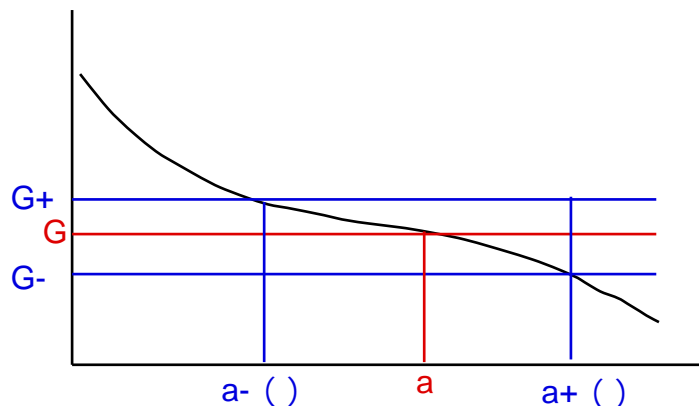
limit of the function. The x-coordinate is in a tiny neighbourhood of the limit, when the independent variable  $t$  is in a tiny neighbourhood of some condensation point. (The condensation points don't have to be



themselves members of the two continuous sequences.) If this limit about which we talk is equal to the function value at this point, then the function is continuous at this point. *The continuous model of motion demands this to be valid at any point of the considered motion.* Many of the usual functions are indeed continuous, but investigation exactly the cases in which this isn't true, is what will give us some more insight.

We see that the meaning of continuity has to do with limits. And this is reason enough to take a closer look at limits and the capabilities of the HP49G when working with limits. In general, when we say that some function  $y = f(x)$  has the limit  $G$  when the independent variable  $x$  approaches  $a$ , we mean that the quantity  $|f(x) - G|$  approaches 0, when  $x$  approaches  $a$ . Let's put this in precise math language. We consider some function  $y = f(x)$  and the quantity  $|f(x) - G|$ . We take an arbitrary small positive quantity  $\epsilon > 0$  and compare this to the

# Sequences, series and limits with the HP49G - Part 6



quantity  $|f(x) - G|$  while  $x$  approaches  $a$ . If the inequality  $|f(x) - G| < \epsilon$  holds for any  $x$  for which the inequality  $0 < |x - a| < \delta$  holds, then the function  $y = f(x)$  has the limit  $G$  when the independent variable approaches the value  $a$ . We denote that with  $\lim_{x \rightarrow a} f(x) = G$ . The quantity  $\delta$  is not some known function, but it depends on the choice of the tiny quantity  $\epsilon$ . The above can also be understood as the search for some tiny neighbourhood of  $x = a$ , inside of which the function  $y = f(x)$  creates values of  $y$  which all are inside a tiny neighbourhood of  $G$ . For example, consider the function  $y = x^2$  at the point  $x = 0$ . The limit  $G$  of this function, when  $x$  approaches  $0$ , is equal to  $0$ . If we demand that the quantity  $|f(x) - G| < \epsilon$ , with  $\epsilon = 1E-6$ , then we have to satisfy the inequality  $|x^2 - 0| < 1E-6$  for some distinct values of  $x$ . Do such values exist? Yes, they exist and they all satisfy the inequality  $|x - a| < \delta$ , which in this particular case translates to  $|x - 0| < 1E-3$ , or  $|x| < 1E-3$ . If we choose  $x$  to be "at most"  $1E-3$  away from  $x = a = 0$ , then we satisfy the demand that  $y$  is "at most"  $1E-6$  away from  $y = G = 0$ .

The HP49G has extended capabilities for finding limits of functions.

We start with simple examples and proceed to more complex ones. So let's try to find  $\lim_{x \rightarrow 0} x^2$ . Go to the EQW and press [blue-shift], [4] to get the menu CALCULUS. Press the menu key [LIMIT] and then the menu key [lim]. The unfinished expression  $\lim( \quad , \quad )$  appears. The cursor blinks over the first argument of the function lim. Enter  $X^2$  and press [arrow-right]. Enter  $X = 0$ , to supply the part  $x \rightarrow 0$ . Press [ENTER]. Now you have  $\lim_{x \rightarrow 0} X^2$  on stack level 1, the "pretty print" version of  $\lim(X^2, X = 0)$ . Press [EXPAND]. The HP49G returns  $0$ , the limit of  $X^2$  for  $X \rightarrow 0$ . Press [red-shift], [HIST] to undo the last result and get the previous stack. We're going to see something strange (again;-). Press [arrow-down] to bring the expression  $\lim_{x \rightarrow 0} X^2$  into the EQW. First thing to notice: now the pretty printed expression appears in the EQW, while previously we had  $\lim(X^2, X = 0)$ . You can select and edit all parts of the pretty printed form. But if you select the whole expression and press [red-shift], [VAR] to copy it, then suddenly the EQW contains...  $(X = 0 \quad X^2)$ , a rather unusual expression. If you now press [ENTER] you will see the expression  $\lim_{x^2} X = 0$  on stack level 1, which is  $\lim_{x \rightarrow 0} X^2$  with its arguments interchanged. Press [arrow-down] again to bring the expression  $\lim_{x^2} X = 0$  into the EQW. Press [red-shift],[VAR] to copy it again, and notice that the selected expression changes again to  $(X^2 \quad X \rightarrow 0)$ . Press [ENTER] and you will see the expression  $\lim_{x \rightarrow 0} X^2$  on stack level 1. The second exchange of arguments has brought us back the original form. The moral of the story is that you should avoid copying the whole expression for a limit, or find out how this curiosity of exchanged arguments could be used for some purposes.



# Sequences, series and limits with the HP49G - Part 6

The function `lim` can of course be used also in stack syntax. Enter  $\frac{1}{X}$ , then  $X = a$  and then enter `LIMIT` to get the result  $\frac{1}{a}$ . The function `lim` was introduced with the last ROM version as a pretty printed alternative of the function `LIMIT`, which is in all previous ROM versions. `LIMIT` is still in the current ROM version, but you won't find it in the command catalogue. If you want to use it with the current ROM version, you have to type it in. If you go to the EQW, type `LIMIT` and then press [arrow-right] to start entering the arguments, then the expression is converted automatically to `lim(■)`, the template of the new pretty printed version. If you try entering '`LIMIT(X2, X = 0)`' from the command line, then the command line parser will mark the equals sign and complain about invalid syntax. Entering '`lim(X2, X = 0)`' brings the same error again. So if you have entered some expression with limits in the EQW, you just can't use the last command recovery to re-enter it from the command line. You have to re-enter the expression from the EQW. What a mystery!

Let's examine that a bit further. Go to the EQW and enter  $\lim_{X \rightarrow 0} X^2$ . Press [ENTER] to put that on the stack, and press [ENTER] two times to make two copies of the expression. Press [->LST] to turn the algebraic object to its list RPL equivalent. The result of this operation is  $\{X \rightarrow 2 \lim_{X \rightarrow 0} (\wedge)\}$ . Either this is much to high mathematics for me, or the pretty print version is totally..., well, let's say "unusual" ;-). Press [OBJ->] to explode the list. On stack level 3 you can see now the object ■ Invalid Expression. The list item count on stack level 1 is 7., which shows that there were invisible items in the list (and also in the algebraic object). Stack level 2 is occupied by an object which doesn't show up. If you drop the 7. and press [TYPE], you get 18. the object type of analytic functions, like SIN, COS and so on. Drop all objects until  $\lim_{X \rightarrow 0} X^2$  is on stack level 1 again. Press [->LST] and

then [->ALG] to transform the algebraic object to a list and then back to algebraic. You get  $\lim_{X \rightarrow 0} X^2$ , as it should be. But if you press [COMP->] and then [->ALG], then you will get the very mysterious result ■ Invalid Expression' which shows again that pretty print has totally messed up the inner structure of the function `LIMIT`. It will work and find the right limits, but you have to be veeeery careful if you use programs that explode and re-build expressions that contain the function `lim`.

After the obligatory cry out, let's take a look at the strength of `lim`. We start with easier examples and proceed to more complicated things.

Enter  $\frac{X^2 - 4}{X - 2}$ , then  $X = 2$ , and then press [`lim`]. The HP49G returns

the result 4, the limit of  $\frac{X^2 - 4}{X - 2}$  for  $X \rightarrow 2$ . The HP49G knows how to find limits in cases where the function itself is undefined, like in the above example. The mathematic part of `lim` is very well done.

From the EQW enter  $\lim(a^X, X = 0)$  and expand. The HP49G returns the result 1.

Enter  $\lim 1 + \frac{1}{X^X}, X =$  and expand. Again, after some seconds you get the result  $e$ .

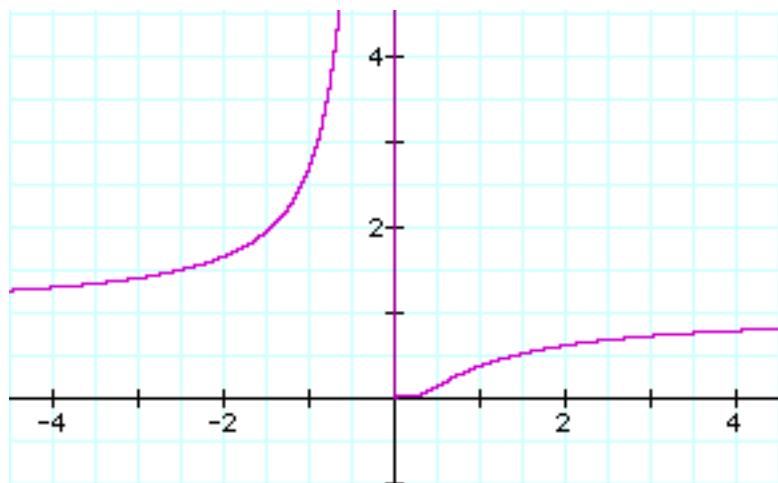
Enter  $\lim (1 + X)^{\frac{1}{X}}, X = 0$  and expand. You get the limit  $e$  after some seconds.

Enter  $\lim \frac{\ln(1 + X)}{X}, X = 0$  and expand. The result is 1.

# Sequences, series and limits with the HP49G - Part 6

Enter  $\lim \frac{\text{SIN}(X)}{X}, X = 0$  and expand to get the limit, 1.

In many cases it is important to know if the independent variable approaches some value from the left or from the right. Consider for example the function  $e^{-\frac{1}{x}}$ . If you plot it then you can see that if  $x$



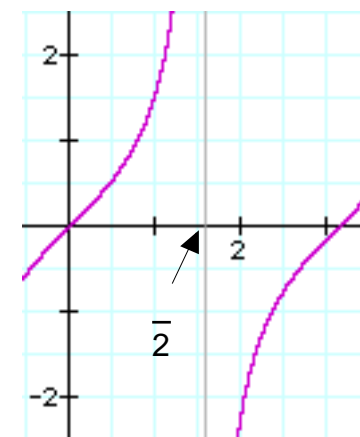
approaches 0 from the left, that is if it comes from values less than 0, then  $e^{-\frac{1}{x}}$  goes up to  $+$ . But if  $x$  approaches 0 from the right, that is if it comes from values greater than 0, then  $e^{-\frac{1}{x}}$  goes to 0. If you go to the EQW, enter  $\lim e^{-\frac{1}{x}}, X = 0$  and expand, then the HP49G errors out with "Unsigned inf". In such cases it can help to specify the direction from which the limit is approached. In this case, if we want to specify that  $X$  approaches 0 from the left, then we enter  $X = 0 - 0$  for the value that  $X$  is going to. if we want to specify that  $X$  approaches 0 from the right, then we enter  $X = 0 + 0$ . Here we

encounter yet another strange thing. Go to the EQW, enter  $\lim e^{-\frac{1}{x}}, X = 0 - 0$  and press [ENTER] to put the expression on

stack level 1. Now you see  $\lim_{x \rightarrow 0-0} e^{-\frac{1}{x}}$ . Expanding this we get... the error "Unsigned inf" again. But if we use stack syntax, that is entering  $e^{-\frac{1}{x}}$ , then  $X = 0 - 0$  and pressing [lim], we get  $+$  ! Go figure out why the algebraic syntax fails, while stack syntax works. Let's try the opposite direction. Enter  $\lim e^{-\frac{1}{x}}, X = 0 + 0$  and expand. Again the

HP49G errors out. But entering  $e^{-\frac{1}{x}}$ , then  $X = 0 + 0$  and pressing [lim] we get the result  $+0$ , which means that the limit is 0 and that it is approached from positive values. So it seems that the "pretty print" of the command lim is, let's say, not very recommendable. ;-)

Let's consider another example, the function  $\tan(x)$ . If  $x$  approaches  $\frac{\pi}{2}$  from the left, that is if it comes from values less than  $\frac{\pi}{2}$ , then  $\tan(x)$  goes up to  $+$ . But if  $x$  approaches  $\frac{\pi}{2}$  from the right, that is if it comes from values greater than  $\frac{\pi}{2}$ , then  $\tan(x)$  goes down to  $-$ . First we try without specifying a direction from which  $x$  approaches  $\frac{\pi}{2}$ . Go to the EQW and enter



# Sequences, series and limits with the HP49G - Part 6

$\lim \text{TAN}(X), X = \frac{1}{2}$ . Press [EXPAND] to get  $\infty$ , an *unsigned*

infinity, an infinity without sign. Now let's try to specify the direction from which we approach the limit. If we want to specify that  $X$  approaches  $\frac{1}{2}$  from the left, then we enter  $\lim \text{TAN}(X), X = \frac{1}{2} - 0$ .

Expanding the last expression we get again  $\infty$  and not  $+$ , which shows that in this particular case the command `lim` has a shortcoming.

Same for the other direction. We specify that  $X$  approaches  $\frac{1}{2}$  from the right, that is we enter  $\lim \text{TAN}(X), X = \frac{1}{2} + 0$ , and we expand.

But though we specified the direction from where we approach  $\frac{1}{2}$ , we get  $\infty$  and not  $+$ . Specifying the direction from which we approach the limit, doesn't work in this case. Even using the stack syntax of `lim` wouldn't help. But we can use the command `SERIES` instead, which is more powerful (and in general slower) when finding limits. Enter  $\text{TAN}(X), X = \frac{1}{2} - 0$ , and 1, the order of the series expansion of

$\text{TAN}(X)$  for  $X = \frac{1}{2}$  coming from the left. Press [SERIES]. The

HP49G returns a list at stack level 2 and  $h = \frac{1}{2} - X$  at stack level 1.

Press [SUBST] to substitute  $\frac{1}{2} - X$  for  $h$  in all expressions of the list, and then press [HEAD] to get the first element of the list, the limit of  $\text{TAN}(X)$  for  $X = \frac{1}{2}$  coming from the left. The result is

Limit:  $(+)$ . Let's try the same for  $X = \frac{1}{2}$  coming from the right.

Enter  $\text{TAN}(X), X = \frac{1}{2} + 0$ , and 1. Enter SERIES SUBST HEAD to get the result Limit:  $(-)$ . We see two things here. First thing, *we can specify one sided limits*. Second thing, the command `SERIES` seems to be more powerful than the command `lim`. If `lim` fails, don't give up but try again using `SERIES`.

We continue with some additional examples of finding limits with the HP49G.

Enter  $\lim \frac{\text{LN}(X)}{X - 1}, X = 1$  and expand. The HP49G returns 1.

Enter  $\lim \frac{X^3}{2e^X - X^2 - 2}, X = 0$  and expand. The HP49G finds the limit, 3.

Enter  $\lim \frac{\text{COS}(X) - 1}{X^2}, X = 0$  and expand to get the result  $-\frac{1}{2}$ .

Enter  $\lim \frac{\text{SIN}(2X)}{\text{COS}(X)^2}, X = \frac{1}{2}$  and expand to get  $\infty$ . In this example

we must specify from which direction  $X$  approaches  $\frac{1}{2}$ . But trying to

expand  $\lim \frac{\text{SIN}(2X)}{\text{COS}(X)^2}, X = \frac{1}{2} + 0$  or  $\lim \frac{\text{SIN}(2X)}{\text{COS}(X)^2}, X = \frac{1}{2} - 0$

still returns  $\infty$ , the unsigned infinity. So we try to find the one sided

limits using `SERIES`. Enter  $\frac{\text{SIN}(2X)}{\text{COS}(X)^2}, X = \frac{1}{2} - 0$ , 1, and then

`SERIES SUBST HEAD`. Now you have the result Limit:  $(+)$  which

# Sequences, series and limits with the HP49G - Part 6

is correct for  $X$  approaching  $\frac{2}{2}$  from the left. For the right sided limit enter  $\frac{\sin(2-X)}{\cos(X)^2}$ ,  $X = \frac{2}{2} + 0$ , 1, and then SERIES SUBST HEAD to get Limit:  $(-)$ .

Enter  $\lim((1-X) \ln(X-1), X = 1+0)$  and expand. The HP49G errors out with "Unable to find sign". Again the pretty print version has problems in algebraic syntax. Enter  $(1-X) \ln(X-1)$ , then  $X = 1+0$  and press [lim]. Now the result is  $+:0$ .

Enter  $\lim \frac{\tan(3-X)}{\tan(X)}$ ,  $X = \frac{2}{2}$  and expand. The HP49G needs some seconds to return the limit,  $\frac{1}{3}$ .

Enter  $\lim \frac{X^4}{e^X}$ ,  $X =$  and expand. The HP49G errors out with "Non algebraic in expression". Using stack syntax, that is entering  $\frac{X^4}{e^X}$ , then  $X =$  and pressing [lim], returns the limit,  $+:0$ .

Enter  $\lim \frac{\ln(X)}{X^n}$ ,  $X =$  and expand. The HP49G needs several seconds to error out with "Unsigned inf error". Entering  $\frac{\ln(X)}{X^n}$ , then  $X =$  and pressing [lim], doesn't do any better in this case. So we try SERIES. Enter  $\frac{\ln(X)}{X^n}$ , then  $X =$  and then 1. Press [SERIES] and let the HP49G do the work, until it errors out with "Unsigned inf error" again. What can we do? Well, let's try to assume something for

$n$ . Enter  $n = 0$  ASSUME. If you now enter  $\lim \frac{\ln(X)}{X^n}$ ,  $X =$  and expand, the HP49G errors with "Non algebraic in expression". If you enter  $\frac{\ln(X)}{X^n}$ , then  $X =$  and press [lim], the HP49G needs several seconds to return the limit,  $+:0$ . Enter  $n$  UNASSUME  $n = 0$  ASSUME and then enter  $\frac{\ln(X)}{X^n}$ , then  $X =$  and press [lim] to try to find the limit for  $n = 0$ . The HP49G returns  $+$ .

Enter  $\lim X \cdot \text{ATAN} \frac{1}{X}$ ,  $X =$  and expand. The HP49G returns 1.

Enter  $\lim \frac{1}{\sin(X)} - \frac{1}{X+X^2}$ ,  $X = 0$  and expand to get 1.

Enter  $\lim(X^X, X = 0+0)$  and expand. The HP49G errors out with "Unable to find sign". But if you enter  $X^X$ , then  $X = 0+0$ , and press [lim], the HP49G returns the limit 1.

Enter  $\lim(\sqrt[n]{X}, X =)$  and expand to get 1.

From the above examples we see: If lim in algebraic "pretty print" syntax doesn't work, then try lim with stack syntax. If this still doesn't work, then try SERIES. And if it still doesn't work, then try to make assumptions for additional variables that appear in the expression for which you want to find the limit.

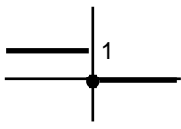
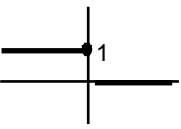
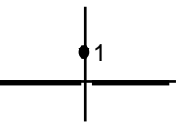
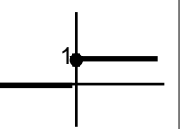
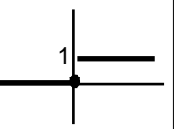
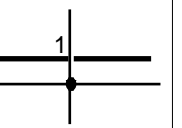
What about piece wise defined functions? Can the HP49G find limits of such expressions? Let's see. First experiment, enter  $\text{IFTE}(X < 0, 1, 0)$ , then  $X = 0$ , and press [lim]. The HP49G returns 0, which is wrong! It should return 1. Let's try the one sided limit.

# Sequences, series and limits with the HP49G - Part 6

Enter IFTE( $X < 0, 1, 0$ ), then  $X = 0 - 0$ , and press [lim]. You get 0, the wrong result again. In the following table we have a summary of all cases where the HP49G finds wrong limits of a piece wise defined function. The top row contains the functions that were used. The left column contains the value that  $X$  approaches. The second row

$X = a \pm 1E-499$ , and substitute this in the function that contains IFTE. This causes the condition to be evaluated to 1 for true or 0 for false, but fortunately doesn't do anything else, so that we can MATCH  $X$  for  $0 \pm 1E-499$  and EVAL to get the right part of the piece wise defined function and find its limit using the command SERIES. This of

course limits the usefulness of our program for finding limits of functions that don't contain  $1E-499$ , but I think that this limitation is not very bad, since such functions are not likely to be used.

	IFTE( $X < 0, 1, 0$ )	IFTE( $X = 0, 1, 0$ )	IFTE( $X = 0, 1, 0$ )	IFTE( $X > 0, 1, 0$ )	IFTE( $X > 0, 1, 0$ )	IFTE( $X > 0, 1, 0$ )
						
$X=0$	0	1	1	1	0	0
$X=0-0$	0	1	1	1	0	0
$X=0+0$	0	1	1	1	0	0

contains a mini graph of the used function. The red cells are the cases where the HP49G returns the wrong limit. As we can see the HP49G rather returns the *value* of the function at the considered point but not the limit of the function. When it finds the correct limit, it does so because of the *false* reason. Especially for the cases where no direction was specified, like the limit for  $X = 0$  in the above table, the HP49G should return something that tells you that a direction is needed, or error out. So you shouldn't try to find such limits by simply using the piece wise defined function. Let's try to make a program that enhances the built-in lim for such cases in which we want to find the limit of a piece wise defined function at the point of the "jump". Our program could first check to see if the function contains IFTE. If it doesn't then we simply use SERIES to find the limit. We don't use lim, because SERIES is more powerful. For example, the

command lim finds both limits of  $\frac{1}{X}$  for  $X = 0 + 0$  and for  $X = 0 - 0$

to be  $+$ , while SERIES finds  $+$  for  $X = 0 + 0$  and  $-$  for  $X = 0 - 0$ . If IFTE is in the function, we check again if a one sided limit was specified. If no direction was specified, then we return a  $?$ , which signals us that a direction is needed. If a direction was specified from which  $X$  approaches  $a$ , then we turn  $X = a \pm 0$  to

```
<<
PUSH RCLVX
-> func xeq vx
<<
IF                                     @If IFTE is in the function
  func ->LST
  { IFTE } HEAD POS
THEN                                  @then
  IF
    xeq ->LST                         @If the right or left
    3 5 SUB                          @limit is wanted
    DUP {0 + =} SAME
    SWAP {0 - =} SAME
    OR
  THEN
    func xeq ->LST                   @Built up X=a±1E-499
    3 1E-499 PUT ->ALG
    SUBST                             @Substitute X=a±1E-499
    POP PUSH                          @POP and PUSH (to avoid
                                     @mode changes)
```

# Sequences, series and limits with the HP49G - Part 6

```

xeq ->LST TAIL      @prepare list for match
1 3 SUB
2 1E-499 PUT
->ALG
xeq ->LST HEAD
2 ->LIST

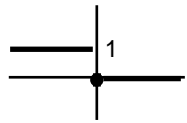
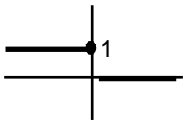
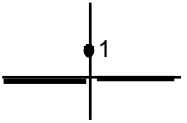
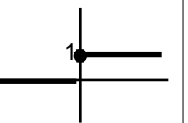
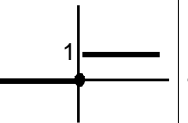
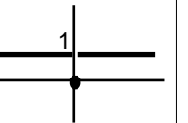
MATCH DROP          @Match a±1E-499 to X
EVAL                 @Find limit of the right part
xeq 2 SERIES          @of the definition using SERIES
SUBST HEAD
EXPAND

ELSE                 @If no direction was specified
?                     @return ?
END

ELSE                 @else (no IFTE in function)
func xeq 2 SERIES @just find the limit
SUBST HEAD EXPAND @using SERIES
END
vx STOVX             @Restore VX
>>
POP                  @Restore settings
>>

```

This is the program is LIM from the programs that come along with this document. Let's check it. We use first exactly the same piece wise defined functions like that in the table on the previous page. The results are summarised on the table below. As we can see the program returns the correct limits. We do some additional tests. Enter

	IFTE(X <0,1,0)	IFTE(X 0,1,0)	IFTE(X ==0,1,0)	IFTE(X 0,1,0)	IFTE(X >0,1,0)	IFTE(X 0,1,0)
						
X=0	?	?	?	?	?	?
X=0-0	1	1	0	0	0	1
X=0+0	0	0	0	1	1	1

IFTE  $X < 0, 0, \frac{\sin(X)}{X}$ ,  $X = 0 + 0$  and press [LIM] to get 1, the

correct limit. Enter IFTE  $X < 0, \frac{\sin(X)}{X}, e^{-\frac{1}{X}}$ ,  $X = 0 - 0$  and press

[LIM] to get 1, the correct limit again. Enter

IFTE  $X < 0, \frac{\sin(X)}{X}, e^{-\frac{1}{X}}$ ,  $X = 0 + 0$  and press [LIM] to get +:0,

which again is correct.

The program can be made better. For example it runs the same or nearly the same code more than once. The portions of code which are similar or even identical could be run only once. Another thing to make better would be to return both the left and the right limit in case no direction is specified. And another thing for improvement is that the program will work only for an argument on stack level 1, that has the form  $X = a$ , or  $X = a + 0$ , or  $X = a - 0$ , where  $a$  is a single term. For example if this argument is  $X = 1 + a + 0$ , that is if we want the limit for  $X = 1 + a$  the program will not work properly.

We return to continuous functions. Using limits we can make a definition of continuity that allows us to implement a program that tests if some function is continuous at a given point. If a function  $f(x)$  is defined at  $x =$  and also defined in the neighbourhood of  $x =$ , then we can say that it is continuous at  $x =$ , if  $\lim_{x \rightarrow} f(x) = f( )$ . This

means for our program that it has to calculate  $\lim_{x \rightarrow} f(x)$  and  $f( )$  and then check if the two quantities are equal to each other. Since we might have some piece wise defined function, we use the program LIM and not the built-in function lim, so that we are also able to find if some piece wise

# Sequences, series and limits with the HP49G - Part 6

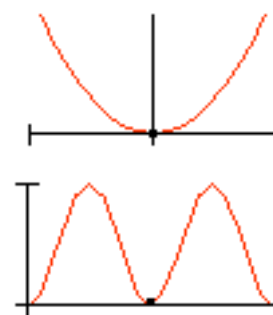
defined function is continuous at some point or not. The program should take a function and an equation of the form  $x =$  and return a 1 if the function is continuous at or a 0 otherwise.

```
<<
  PUSH RCLVX
  -> func xeq vx
  <<
    func
    xeq EQ-> {0 +} +      @Construct x = +0
    ->ALG =
    LIM                  @Find limit
    func
    xeq EQ-> {0 -} +      @Construct x = -0
    ->ALG =
    LIM                  @Find limit
    IF                  @If one or both limits aren't
      DUP2 2 ->LIST      @defined
      ? POS
    THEN                @then return 0 (discont.)
      DROP2 0
    ELSE                @but if the limits are defined
      func xeq SUBST      @Find f( )
      EXPAND
      -> rlim llim fval
    <<
      CASE
        NEG             @If both limits are and/or
        2 ->LIST         @- then we have a pole
        rlim POS         @and so...
        NEG
        2 ->LIST
        llim POS
        AND
      THEN               @...return -1
        -1
      END
  <<
  CASE
    NEG             @If both limits are and/or
    2 ->LIST         @- then we have a pole
    rlim POS         @and so...
    NEG
    2 ->LIST
    llim POS
    AND
  THEN               @...return -1
    -1
  END
```

```
fval ?      @If function is undefined
SAME        @but left and right limits
rlim llim   @exist and are equal to
== AND      @each other
THEN        @then we have a removable
0 xeq EQ->   @discontinuity. Build up
== rlim     @replacement function
func IFTE
2 ->LIST
END
llim rlim   @If both limits and the function
== rlim     @value are equal, then
fval ==     @continuous
AND fval
? SAME
NOT AND
THEN
1
END
0           @Else discontinuous
END
>>
END
vx STOVX    @Restore VX
>>
POP         @Restore settings
>>
```

This is the program ISCONT? that comes along with this document. Let's test it. Enter  $X^2$ ,  $X = 0$  and press [ISCONT?] to check the function  $X^2$  for continuity at  $X = 0$ . The result is 1, which tells you that  $X^2$  is continuous at  $X = 0$ .

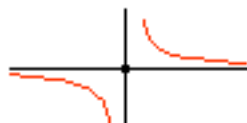
Enter  $1 - \cos(X)^2$ ,  $X =$  , and press



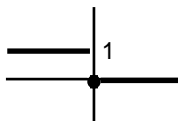
# Sequences, series and limits with the HP49G - Part 6

[ISCONT?]. Again the result is 1, the function is continuous at  $X = 0$ .

Enter  $\frac{1}{X}$ ,  $X = 0$ , and press [ISCONT?]. The result is -1, which means that the function is discontinuous and has a pole at  $X = 0$ .



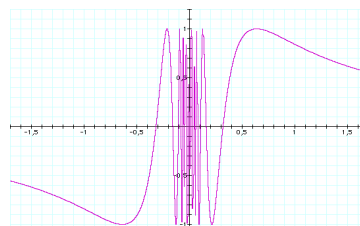
Let's test the program with piece wise defined functions. Enter IFTE( $X < 0, 1, 0$ ),  $X = 0$  and press [ISCONT?]. The HP49G returns 0, the function is discontinuous at  $X = 0$ .



Enter IFTE  $X < 0, \frac{\sin(X)}{X}, e^{-\frac{1}{X}}$ ,  $X = 0$  and



press [ISCONT?]. The HP49G returns 0 because the function is discontinuous at  $X = 0$ .



Enter  $\sin \frac{1}{X}$ ,  $X = 0$  and press

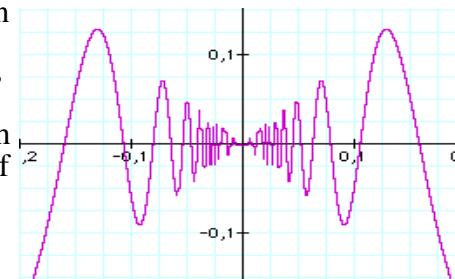
[ISCONT?]. The HP49G flashes a message "Bounded var error" and displays a menu with "OK" (key [F6]). Press the menu key [OK]. The same message flashes once more. Press the menu key [OK] again. The result is 0. We have the same message like when we want to get the limit of  $\sin(X)$  for  $X$ . The function oscillates between -1 and 1, it is bounded between these two values but doesn't approach some special value.

Enter  $X \sin \frac{1}{X}$ ,  $X = 0$  and press [ISCONT?]. The result is

0 IFTE  $X == 0, 0, X \sin \frac{1}{X}$ . This means that the function is discontinuous at  $X = 0$ , but we have to do with a *removable*

discontinuity. The function

IFTE  $X == 0, 0, X \sin \frac{1}{X}$  is a replacement function which retains all properties of  $X \sin \frac{1}{X}$  but is continuous at  $X = 0$ .



For better understanding of the *removable* discontinuity consider the

function  $\frac{X^2 - 2}{X - 1}$ . This function isn't defined at  $X = 1$ . Enter  $\frac{X^2 - 2}{X - 1}$ , then  $X = 1$  and press [SUBST] and [EXPAND] to get

a ?, which shows that the function isn't defined at this point. But both the right and the left limits are -1 at  $X = 1$ . Enter  $\frac{X^2 - 2}{X - 1}$ , then

$X = 1 + 0$  (for the right limit) and press [lim] to get -1. Enter again

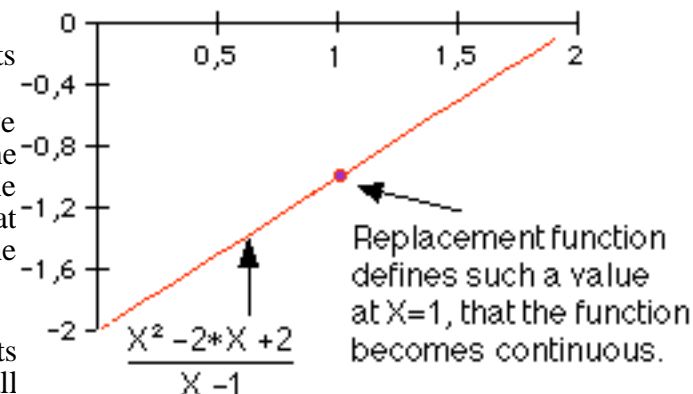
$\frac{X^2 - 2}{X - 1}$ , then  $X = 1 - 0$  (for the left limit) and press [lim] to get

-1 again. Since the original function isn't defined and thus discontinuous at

$X = 1$ , and

since both limits are -1 at the same point, we can change the definition of the function, so that it is equal to the limit -1 at

$X = 1$  and retains its definition at all





# Sequences, series and limits with the HP49G - Part 6

other points. This is exactly what the program ISCONT? does in such cases. Enter  $\frac{X^2 - 2}{X - 1}$ , then  $X = 1$  and press [ISCONT?]. The

result is the list 0 IFTE  $X == 1, -1, \frac{X^2 - 2}{X - 1}$ , which tells

you that the function is discontinuous at  $X = 1$ , but the discontinuity is removable through the replacement function

$$\text{IFTE } X == 1, -1, \frac{X^2 - 2}{X - 1}.$$

Perhaps you remember that in the trigonometry marathon we calculated

the sum  $\sum_{n=0}^N \sin(n \cdot X)$  and that we found the result

$$\frac{\sin(X \cdot N + X) - (\sin(X \cdot N) + \sin(X))}{2 \cos(X) - 2}.$$

The question there was,

what is going on when for example  $X = 2$ . The sum is a finite

quantity but the quantity  $\frac{\sin(X \cdot N + X) - (\sin(X \cdot N) + \sin(X))}{2 \cos(X) - 2}$  has

the denominator  $2 \cos(X) - 2$ , which is 0 when  $X = 2$ . In the trigonometry marathon we have taken the limit of the expression  $\frac{\sin(X \cdot N + X) - (\sin(X \cdot N) + \sin(X))}{2 \cos(X) - 2}$  for  $X = 2$  and  $N = 5$

and we found it to be equal to 0. Here we can go a bit further to understand better what is going on. Enter

$$\frac{\sin(X \cdot N + X) - (\sin(X \cdot N) + \sin(X))}{2 \cos(X) - 2},$$

and then  $N = 5$ . Press

[SUBST] and enter  $X = 2$ . Press [ISCONT?]. The HP49G needs several seconds to return:

$$0 \text{ IFTE } X == 2, 0, \frac{\sin(X \cdot N + X) - (\sin(X \cdot N) + \sin(X))}{2 \cos(X) - 2}.$$

The function is discontinuous at  $X = 2$ , but the discontinuity is removable and the function becomes continuous if we replace it with:

$$\text{IFTE } X == 2, 0, \frac{\sin(X \cdot N + X) - (\sin(X \cdot N) + \sin(X))}{2 \cos(X) - 2}.$$

Exactly the same way we can always replace such functions with removable discontinuities by functions that are continuous at the considered points and retain the original definition of the function in all other points. We will meet such functions again at the differential equations marathon.

A more difficult problem is to find out if a function is continuous in some given interval of values of the independent variable. This would require to find out if  $|f(x) - f(y)| < \epsilon$  for  $|x - y| < \delta$  for all values of

that are in the interval, where  $\epsilon$  and  $\delta$  are arbitrary small positive quantities. For example, if we wanted to check if  $f(X) = X^2$  is continuous for  $X$  taking values from  $-1$  to  $1$ , we would have to solve inequalities of the form  $|(X + \delta)^2 - X^2| < \epsilon$ , which the HP49G can't solve. So we are going to fake it. (Shame on us! ;-))

You of course have noticed that ratios play a major role when it comes to discontinuities. Most (all?) of the time discontinuities appear because some denominator is equal to 0. And so we make a program that takes some function and tries to find if roots of the denominator exist in the interval in which we examine the continuity of the function.

```
<<
RCLVX { } { }
-> func inter vx rat disc
<<
PUSH 1 CF
func TAN2SC
```

@Turn TAN to SIN/COS

# Sequences, series and limits with the HP49G - Part 6

```

EXPAND
IF                                @If the result of EXPAND
    DUP TYPE 9                    @isn't algebraic
THEN
    1 ->ALG                      @then turn it ti algebraic
END
->LST                            @Turn alg. to list
1                                @Do to all objects in list
<<
    IF                            @If object is / then
        { / ) OVER POS           @evaluate and add ratio to
    THEN                          @local variable 'rat'
        EVAL DUP
        'rat' STO+
    ELSE                          @else simply evaluate
        EVAL
    END
>> DOSUBS DROP
IF                                @If list rat isn't empty
    rat { }
THEN
    rat 1                        @Take denominator of each
    <<                          @ratio
        FXND NIP
    >> DOSUBS
    inter HEAD SOLVE             @Find roots of each denom.
    1
    <<                          @Set flag 1 if roots
        IF                      @were found
            DUP { }
        THEN
            1 SF
    >> DOSUBS
IF                                @If roots were found
    1 FS?
THEN
    1
    <<

```

```

IF                                @If roots were returned in
    DUP TYPE 5 ==                @a list then explode it.
THEN
    OBJ-> DROP
END
>> DOSUBS
1
<<                                @Test if solutions are in
    EQ-> NIP DUP                  @interval.
    inter 2 GET
    OVER inter 3
    GET AND
    IF                            @If test can't be evaluated
        DUP TYPE 9 ==            @because of variables like
    THEN                          @'n1' in the solution
        NIP                      @then return uneval. test
        'disc' STO+              @add result to list 'disc'
    ELSE                          @else
        IF                      @if root not in interval
            NOT
        THEN                      @then drop it.
            DROP
        ELSE                      @else
            'disc' STO+          @add result to list 'disc'
        END
    END
    >> DOSUBS
ELSE                                @If no roots were found
    DROP                          @drop empty list of roots.
END
disc vx STOVX
POP
>>
>>

```

This is the program INTERCONT? that comes along with this document. It takes a function and a list from the stack. The list contains

# Sequences, series and limits with the HP49G - Part 6

the independent variable and the low and high values that defined the interval in which we want to check the function.

Enter for example  $X^2$  and  $\{X \ -1 \ 1\}$  and press [INTERCONT?]. The HP49G returns an empty list which means that the program didn't find any discontinuities in the specified interval.

Enter  $\frac{\text{SIN}(X)}{X}$ ,  $\{X \ - \ \}$  and press [INTERCONT?]. The

HP49G returns  $\{0\}$  which means that the program found one discontinuity at  $X = 0$ . (You can use this value to further examine the discontinuity using the program ISCONT?.)

Enter  $\frac{1}{\text{SIN}(X)}$ ,  $\{X \ - \ \}$  and press [INTERCONT?]. Now the HP49G needs a bit longer and it returns a list that looks different:

$$2 \quad n1 - \quad \text{AND} \quad 2 \quad n1 \\ -(2 \quad n1 - ) - \quad \text{AND} \quad -(2 \quad n1 - )$$

We take a closer look at the first expression in the list. The values for  $X$  that possibly result in a discontinuity of the function are of the form  $2 \quad n1$ , where  $n1$  is some integer. All values of  $n1$  that satisfy  $2 \quad n1 - \quad \text{AND} \quad 2 \quad n1$  lead to in values of  $X$  that result in discontinuities of  $\frac{1}{\text{SIN}(X)}$  in the interval from  $-$  to  $.$  The

HP49G can't solve the inequalities for  $n1$  and so we have to find out these values ourselves. From the first inequality we have:

$$2 \quad n1 - \quad 2 \quad n1 - 1 \quad n1 - \frac{1}{2}$$

At the same time the second inequality must be satisfied too and so:

$$2 \quad n1 \quad 2 \quad n1 - 1 \quad n1 - \frac{1}{2}$$

That means that  $n1$  must satisfy  $n1 - \frac{1}{2} \quad \text{AND} \quad n1 - \frac{1}{2}$  and at the same time must be integer. So the only possible value of  $n1$  is  $0$ , and the corresponding value for  $X$  is  $X = 2 \quad n1 = 2 \quad 0 = 0$ . That means that  $\frac{1}{\text{SIN}(X)}$  has a discontinuity at  $X = 0$ .

The second element of the returned list of possible discontinuity points,  $-(2 \quad n1 - ) - \quad \text{AND} \quad -(2 \quad n1 - )$ , can be examined the same way.

The program will not find some removable discontinuities, like for example the discontinuity for  $X = 1$  of the function  $\frac{X^2 - 2 \quad X + 2}{X - 1}$ .

This is because we expand the function and doing this the HP49G turns it to  $X - 2$ .

In the program we use the command TAN2SC to turn any occurrence of TAN to SIN/COS, so that we can handle it like the other ratios.

The command FXND, that we also use in the program, takes an expression and returns its numerator and denominator separately. If the expression doesn't have a denominator, that is is the denominator is 1, then a 1 is returned as denominator.

One purpose that the program INRECONT? can be used for is of course to find if some function is continuous in a specified interval. Another purpose is to enhance the capabilities of the HP49G when it evaluates inequalities. It may sound strange, but remember that mathematics is a land with many, many connections between its parts. Let's start with an example. Remove all assumptions for  $Z$ , enter  $e^{-Z} \quad \text{SIN}(Z) \quad 0$  and expand. The result is the unevaluated expression, because the HP49G can't determine if it is true or false.

# Sequences, series and limits with the HP49G - Part 6

Actually this is correct since we didn't tell it what values  $Z$  can have. So let's do that. Enter  $Z = 0$  and use ASSUME, then press [DROP], then enter  $Z$  and ASSUME and DROP again. Now the HP49G knows that  $0 < Z$ . Having  $Z$  in this interval, the expression  $e^{-Z} \sin(Z)$  is nonnegative, and so expanding  $e^{-Z} \sin(Z) > 0$  should return 1 now. But it doesn't. The HP49G still can't figure out that  $e^{-Z} \sin(Z)$  is greater than or equal to 0. But here we can use another way. A function that is continuous in a specified interval and positive at some point of this interval, is positive over the whole interval. Similarly, a function that is continuous in a specified interval and negative at some point of this interval, is negative over the whole interval. For our example that means:

- 1) Find the sign of the function  $e^{-Z} \sin(Z)$  in an arbitrary point  $Z_0$  between 0 and  $\pi$ .
- 2) Find if the function  $e^{-Z} \sin(Z)$  is continuous from  $Z = 0$  to  $Z = \pi$ . If it isn't continuous, then we can't say anything about  $e^{-Z} \sin(Z) > 0$ . But if it is...
- 3) Check if  $e^{-Z} \sin(Z)$  has roots between  $Z = 0$  and  $Z = \pi$ . If it does, then  $e^{-Z} \sin(Z) > 0$  we again can't say if  $e^{-Z} \sin(Z) > 0$  is true or false. But if it doesn't then  $e^{-Z} \sin(Z)$  has the sign that we found in step 1 over the whole interval. That means, we can evaluate  $\text{SIGN}(e^{-Z_0} \sin(Z_0)) > 0$  instead of the original  $e^{-Z} \sin(Z) > 0$ .

These are the basic operations of the following program, EVACOMP, that takes an inequality from stack level 2 and a list from stack level 1, that contains the independent variable and the low and high values that define the interval. It returns 1 if the inequality holds in the interval, or 0 if it doesn't.

```
<<
PUSH
SWAP OBJ-> NIP          @Transform lhs rhs to lhs-rhs 0
UNROT - 0 ROT           @and lhs rhs to lhs-rhs 0 and
EVAL                    @evaluate inequality.
SWAP OBJ->              @Evaluate low and high of
DROP EVAL SWAP          @interval. (Why?)
EVAL SWAP 3
->LIST
{ HOME CASDIR REALASSUME }
RCL RCLVX 0
-> cmpexpr inter assmlst vx hs
<<
inter OBJ-> DROP        @Remove all assumptions for
ROT UNASSUME            @independend variable.
DUP 4 ROLL              @Assume indep low
ASSUME DROP
SWAP ASSUME             @Assume indep high
DROP
cmpexpr OBJ->           @Find sign of lhs-rhs at
3 DROPN                @ (low+high)/2
inter OBJ-> DROP        @ (centre of interval).
+ 2 / = SUBST
EXPAND ->NUM SIGN
IF                      @If sign 0.
DUP
THEN                   @then...
'hs' STO
cmpexpr OBJ->           @Find eventual discontinuities
3 DROPN inter          @of lhs-rhs in interval
INTERCONT?
IF                      @If no discontinuities found
{ } SAME
THEN
cmpexpr OBJ->           @Try to find root of lhs-rhs
3 DROPN               @in interval starting at
inter OBJ->            @ (low+high)/2
DROP SWAP
```

# Sequences, series and limits with the HP49G - Part 6

```

->NUM SWAP ->NUM
DUP2 + 2. /
3 ->LIST
IFERR      @If error occurs during
  ROOT      @searching for root
  inter HEAD
  PURGE
THEN        @then assume that no root
  cmpexpr   @exists in interval and
  OBJ-> NIP  @evaluate sign 0 or sign 0
  hs 3 UNPICK
  EVAL
ELSE        @else if root was found
  IF         @check if it is in interval.
    DUP inter @If in interval...
    2 GET >
    SWAP inter
    3 GET >
    AND
  THEN      @then return ? (dunno)
    ?
  ELSE      @else if root out of interval
    cmpexpr @Return result of
    OBJ-> NIP @sign 0 or sign 0
    hs 3
    UNPICK
    EVAL
  END
END
ELSE        @else if discontinuities found
  ?
  @return ? (dunno).
END
ELSE        @else if sign=0 then we have
  DROP ?
  @root in interval, so return ?
  @ (dunno again).
END
vx STOVX
{ HONE CASDIR }
EVAL assmlst

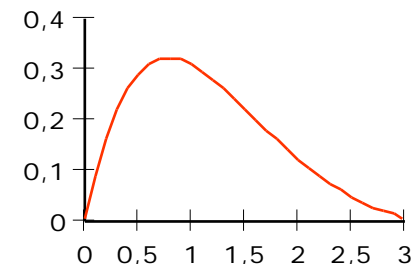
```

```

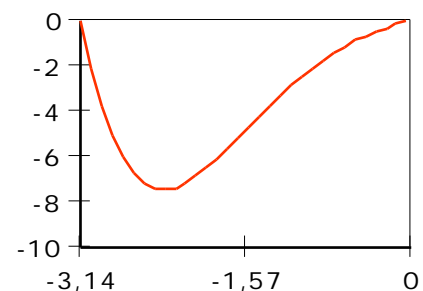
'REALASSUME' STO
>>
POP
>>

```

Let's test the program with some inequalities that the HP49G can't evaluate out of the box. Enter the inequality  $e^{-Z} \sin(Z) \geq 0$ , then the interval  $\{Z \geq 0\}$  and then press [EVACOMP]. The HP49G returns 1, that is the inequality holds in the specified interval.



Let's see what happens with the same inequality in the interval from  $Z = -3.14$  to  $Z = 0$ . Enter  $e^{-Z} \sin(Z) \geq 0$ ,  $\{Z \leq 0\}$  and press [EVACOMP]. Now the HP49G returns 0, because the inequality doesn't hold in the specified interval.



This program is a rather dangerous experiment. It relies blindly on the hope that if a root exists in the specified interval, then ROOT will find it if we feed it with a guess value that lies inside the interval. ROOT is the programmable analogon of the numeric equation solver. It takes three arguments: The equation to be solved at stack level 3, the variable to solve for at stack level 2, and the guess (starting) value at stack level 1. This guess vale can have 3 formats. Either it is a single real number that represents our guess. Or it is a list with two real numbers, which specify an interval inside of which we expect (or want to find) a solution. Or it is a list with three real numbers, the first two

# Sequences, series and limits with the HP49G - Part 6

representing the interval in which we search the root, the third representing the guess value inside that interval. Though giving ROOT a list with three real numbers allows some control of the root that it finds, we can't control it rigourously. Exactly this makes the program dangerous. The fact that ROOT doesn't find a root in the specified interval doesn't mean that there are really no roots in that interval (and thus that the sign of the function is the same with the sign at  $\frac{\text{low} + \text{high}}{2}$ ). It rather means that ROOT just found a root outside the interval and nothing more. So the program should be used with caution and perhaps in combination with a graph of the function.

We move on to rules for rearrangements of limits. If  $\lim_{x \rightarrow a} f(x) = F$  and  $\lim_{x \rightarrow a} g(x) = G$  then:

$$\lim_{x \rightarrow a} (f(x) \pm g(x)) = \lim_{x \rightarrow a} f(x) \pm \lim_{x \rightarrow a} g(x)$$

$$\lim_{x \rightarrow a} (f(x) \cdot g(x)) = \lim_{x \rightarrow a} f(x) \cdot \lim_{x \rightarrow a} g(x)$$

$$\lim_{x \rightarrow a} \frac{f(x)}{g(x)} = \frac{\lim_{x \rightarrow a} f(x)}{\lim_{x \rightarrow a} g(x)} \text{ in case } \lim_{x \rightarrow a} g(x) = G \neq 0$$

The HP49G has no built-in commands for converting limits according to these rules. So we write the program LIMEXPAND that applies them.

```
<<
  DUP OBJ-> 3 DROPN      @Full distribution of *, / over
  FDISTRIB              @over +, -.
  SWAP ->LST SIZE 3 -    @Convert lim(f(x),x=a) to
  DUP 3 + SUB + ->ALG    @lim(fdistrib(f(x)),x=a)
  WHILE                 @While matching
    { 'lim(&A+&B,&C)' 'lim(&A,&C)+lim(&B,&C)'}
    MATCH SWAP
    { 'lim(&A*&B,&C)' 'lim(&A,&C)*lim(&B,&C)'}
    MATCH ROT OR
    SWAP
    { 'lim(-&A,&C)' '-lim(&A,&C)'}
    MATCH ROT OR
  REPEAT
  END                    @Until no match occurs
->TERMS                  @Split to summands
1
<<
```

```
    IF                    @If limit of ratio
      DUP
      { '-lim(&A/&B,&C)' '&B)'}
      MATCH SWAP
      { 'lim(&A/&B,&C)' '&B)'}
      MATCH ROT OR
    THEN                  @then take denominator
    OVER                  @and find its limit.
    { '-lim(&A/&B,&C)' '&C)'}
    MATCH DROP
    { 'lim(&A/&B,&C)' '&C)'}
    MATCH DROP
    lim 0
    SWAP DUP
    { 'lim(&A/&B,&C)' 'lim(&A,&C)/lim(&B,&C)'}
    MATCH DROP ROT
    IF                    @If limit was expression
      DUP TYPE 9 ==
    THEN                  @then construct
      UNROT SWAP          @IFTE(limit 0,lim(A)/lim(B),
      IFTE                 @    limit(A/B))
    ELSE
      IF                  @else if it wasn't expression
      THEN                @then if it is 0
      NIP                 @then lim(A)/lim(B)
```

# Sequences, series and limits with the HP49G - Part 6

```

ELSE          @else if it is = 0
  DROP        @then lim(A/B)
END
END
ELSE          @else if no limit of ratio
  DROP        @drop expression
END
>> DOSUBS
0 + LIST      @Build-up sum of limits
>>

```

Let's see the program in action. Enter  $\lim_x (\sin(x) (\cos(x) - x))$  and press [LIMEXPAND]. You get the expanded form:

$$\lim_x (\cos(x)) \lim_x (\sin(x)) - \lim_x (x) \lim_x (\sin(x))$$

Enter  $\lim_{x \rightarrow 0} \frac{x^2 - 1}{x - 1}$  and press [LIMEXPAND]. The limit of the denominator  $x - 1$  for  $x \rightarrow 0$  is not 0 and so you get the expanded form  $\frac{\lim_{x \rightarrow 0} (x^2)}{\lim_{x \rightarrow 0} (x - 1)} - \frac{\lim_{x \rightarrow 0} (1)}{\lim_{x \rightarrow 0} (x - 1)}$ .

But if you enter  $\lim_{x \rightarrow 1} \frac{x^2 - 1}{x - 1}$  and press [LIMEXPAND] then the HP49G finds out that the limit of the denominator  $x - 1$  for  $x \rightarrow 1$  is equal to 0 and so you get the expanded form  $\lim_{x \rightarrow 0} \frac{x^2}{x - 1} - \lim_{x \rightarrow 0} \frac{1}{x - 1}$ .

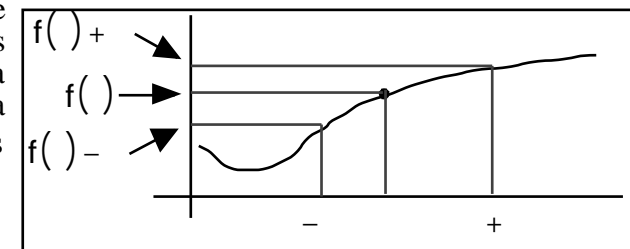
Notice however that though the HP49G will find the correct limit of  $\lim_{x \rightarrow 0} \frac{x^2}{x - 1} - \frac{1}{x - 1}$  which is 2, it will not find the correct limit when

you expand the form  $\lim_{x \rightarrow 0} \frac{x^2}{x - 1} - \lim_{x \rightarrow 0} \frac{1}{x - 1}$ .

How can the program be changed, so that it also expands the denominators according to the rules on the previous page?

And now for something completely different, as Monty Trabythons says. Let's try to visualise how  $\lim$  and  $( )$  relate to each other for a given function  $f(x)$  at a given point  $x = a$ . As we already have seen, if a function  $f(x)$  is defined at  $x = a$  and if it is also defined in the neighbourhood of  $x = a$ , then we can say that it is continuous at  $x = a$ , if  $\lim_{x \rightarrow a} f(x) = f(a)$ . This is equivalent to the following: If for an arbitrary tiny  $\epsilon > 0$  we can find a corresponding  $\delta > 0$ , such that  $|f(x) - f(a)| < \epsilon$  for all values of  $x$  that satisfy  $|x - a| < \delta$ , then the function  $f(x)$  is continuous at the point  $x = a$ . This sentence sounds much more complicated than it is. (Trying not to be misunderstood mathematics statements often sound ununderstandable ;-)). We can take away much of the "mystery" of the above statement, if we visualise it.

What the above statement says is best seen in a picture. We have a function  $f(x)$  and its value  $f(a)$  at the point  $x = a$ . Now, we take a range of

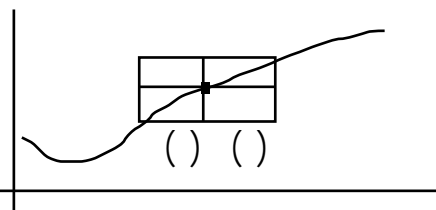


values of  $f(x)$  around  $f(a)$ , from  $f(a) - \epsilon$  to  $f(a) + \epsilon$ . These two values are the values of  $f(x)$  at some points near  $x = a$ . So we have  $f(x_1) = f(a) - \epsilon$  and  $f(x_2) = f(a) + \epsilon$ . The maximum of the absolute

# Sequences, series and limits with the HP49G - Part 6

value of the difference of  $f(x)$  and  $f(x_0) - \epsilon$  on the one hand, and  $f(x)$  and  $f(x_0) + \epsilon$  on the other hand, is  $\epsilon$ . This is what the formula  $|f(x) - f(x_0)| < \epsilon$  says. Now, we search for a band of values of the independent variable  $x$ , such that  $|f(x) - f(x_0)| < \epsilon$  is satisfied. These values of  $x$  are around  $x = x_0$ . The maximum difference between  $x_0$  and these values is  $\delta$ , a tiny positive, that depends somehow on the value of  $\epsilon$ . That is what formula  $|x - x_0| < \delta$  says. If such a  $\delta$  exists, then the function is continuous at  $x = x_0$ . In other words, we search for an interval of values of  $x$ , such that the values of  $f(x)$  stay within  $f(x_0) - \epsilon$  and  $f(x_0) + \epsilon$ .

We make a program that takes a continuous function at  $x = x_0$ , its independent variable, the point  $x_0$  and the tiny positive from the stack, and draws the function along with a box, which has the width  $2\delta$  and



the height  $2\epsilon$ , and which is centred at the point  $(x_0, f(x_0))$ . Then, using tracing we can move the graphics cursor along the function and as long as we stay in the box, we can view the coordinates, which satisfy  $|x - x_0| < \delta$  and  $|f(x) - f(x_0)| < \epsilon$ .

```
<<
->NUM SWAP ->NUM SWAP @Turn    and    to numbers.
-> func var point
<<
  PUSH
  #131d #64d PDIM      @Make PICT 131×64 pixels
  FUNCTION             @Set plot type function
  var INDEP            @Set independent variable
  func STEQ            @Store function in EQ
```

```
"Solving for    "
1 DISP              @Display message.
func DUP            @Build-up |f(x)-f(x0)|. Using ABS
var point =         @with algebraics that contain
SUBST ->NUM          @numeric values, sets approx.
- ABS POP           @mode, so we restore modes.
=                  @Build-up |f(x)-f(x0)| = ε.
var point +         @We will try to find    in the
var point -         @interval from    -    to    + .
point 3 ->LIST       @We use    as help for guessing
ROOT               @a starting value of    .
var PURGE          @Purge created variable.
point - ABS         @Store abs. value of difference
' ' STO            @between solution and    .

"Determining X-range"
1 DISP              @Display message.
point 1.5 * -       @Set horiz. view range such that
point 1.5 * +       @the plot has width 3* δ.
XRNG

"Autoscaling Y-range"
1 DISP              @Display message.
AUTO               @Autoscale.
var PURGE          @Purge var. created by AUTO.
PPAR 1 2 SUB        @Get lower left and upper right
                    @coordinates of the plot.
LIST HEAD IM       @Find height in user coord.
.046875 *           @Enlarge the height of the plot
MAX                @either 3 pixels, or user
DUP NEG SWAP       @units, whichever is bigger.
2 ->LIST PPAR       @.046875 is the result of 3/64,
1 2 SUB IM ADD     @the ratio of 3 pixels to the
OBJ-> DROP YRNG     @corresp. user units.
ERASE DRAX DRAW    @Draw function with axes.
point point -      @Calculate coords. of box
point +            @corners and box centre.
func var point =
```



# Sequences, series and limits with the HP49G - Part 6

```

SUBST ->NUM
DUPDUP -
SWAP +
-> x ly ux y ly uy
<<
  lx ly R->C          @Draw the box.
  ux uy R->C
  BOX
  lx y R->C           @Draw lines in box.
  ux y R->C
  LINE
  x ly R->C
  x uy R->C
  LINE
  PICT                @Calculate coordinates of low
  lx ly y + 2 /
  R->C C->PX
  { #4d #3d } -
  " " 1 ->GROB        @Make " " to GROB using mini
  REPL               @font, and put it in PICT.
  PICT               @Calculate coordinates of high
  lx uy y + 2 /
  R->C C->PX
  { #4d #3d } -
  " " 1 ->GROB        @Make " " to GROB using mini
  REPL               @font, and put it in PICT.
  PICT               @Calculate coordinates of left
  lx x + 2 / ly
  R->C C->PX
  { #0d #2d } ADD
  " " 1 ->GROB        @Make " " to GROB using mini
  REPL               @font, and put it in PICT.
  PICT               @Calculate coordinates of right
  ux x + 2 / ly
  R->C C->PX
  { #0d #2d } ADD
  " " 1 ->GROB        @Make " " to GROB using mini
  REPL               @font, and put it in PICT.

```

```

PICTURE                @Activate plotting environment.
func "function"        @Return labelled results.
->TAG
x "point" ->TAG
" " ->TAG
" " ->TAG
>>
>>
>>

```

This is the program GRAPH that comes along with this document. Let's try it. Enter the function  $\text{SIN}(X)$ , the independent variable  $X$ ,

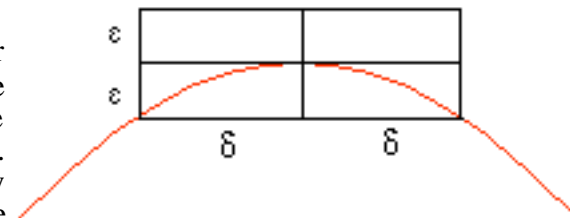
the point  $\frac{\pi}{2}$  where we want to test for continuity, and .00001 as the tiny positive  $\epsilon$ . Press

[GRAPH]. After

some seconds the HP49G draws the picture on the right. Press the menu key [TRACE] to trace the function and then the

menu key [(X,Y)] to display cursor coordinates. Move the cursor along the function by pressing [arrow-left] or [arrow-right]. As long as you the cursor remains in the box, the inequalities  $|x - \frac{\pi}{2}| < \delta$  and  $|f(x) - f(\frac{\pi}{2})| < \epsilon$  are both satisfied. Press [ON] to leave the plotting environment. The program quits and you have the results on the stack.

You see that if  $\text{SIN}(X)$  is allowed to vary from  $\text{SIN}(1.5707963268) + 0.0001$  to  $\text{SIN}(1.5707963268) - 0.0001$ , then  $X$  must vary between  $1.5707963268 + 0.00447213974$  and  $1.5707963268 - 0.00447213974$ . Try with different input data and look at the results. One thing to be aware of: Don't enter values for that are smaller than  $1E-11$ . The program will crash. Nothing else will happen, but you garbage will be left on the stack. This comes

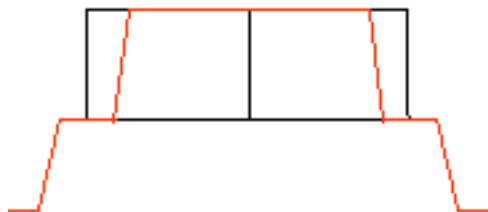


# Sequences, series and limits with the HP49G - Part 6

because the HP49G can handle numbers within  $1E-499$  of 0 but only within  $1E-11$  of 1.

If you enter for example

$\text{SIN}(X)$ ,  $X$ ,  $\frac{1}{2}$  and  $1E-12$ ,



then the HP49G will plot the picture on the left and then crash. The crash comes because the Y-coordinate in pixels of the first " " can't be calculated correctly. The strange "stair" is the function  $\text{SIN}(X)$  drawn with an accuracy that exceeds the available accuracy of the HP49G. The highest step is at  $Y = 1$ , the middle step is at

$Y = .999999999999$ , and the lowest step is at

$Y = .999999999998$ . For the HP49G there is no number between 1 and  $.999999999999$  and no number between  $.999999999999$  and  $.999999999998$ . If you enter 1,  $1E-12$  and then press [+],

then the result will be 1 and not  $1.000000000001$  because this result exceeds the available 12 digits with which the HP49G expresses numbers. As you can see, the smallest difference between the values of Y in plots is  $1E-12$ . The HP49G can't "see" differences of Y values that are even smaller, and thus the "quantised" stair plot. (Nonetheless the plot can still be traced!)

Anyway, what exactly are precision and accuracy? Are there any definitions for these things? And how does the HP49G do numeric calculations? What is its precision and accuracy? Let's take a closer look.

When doing numeric calculations we have to be aware of the fact that precision loss may occur. (We are talking here about what numbers the HP49G shows us out of the box, and not about how it carries out its numeric operations internally.) The numeric precision of the result will eventually be lower than the precision of the operands that we carry operations with. Consider for example the operation (in RPL syntax)  $1. \ 1.E12. \ +$  which on the HP49G results in 1. This result is only

an approximation of the correct result,  $1.000000000001$ . This correct result can't be expressed with the available 12 digits that the HP49G uses to represent real numbers. Or consider the operation (RPL again)  $1. \ 3. \ / \ 3.$ , which results in  $.333333333333$  instead of  $\frac{1}{3}$ . Such problems are probably the strongest reasons for designing the HP49G in such a way that it tries (more or less) to carry operations as exact as possible and only give numeric results when you tell it to do so. It's "standard" mode is exact, which makes exact calculations easier. Of course you can turn approximate mode on and use it like you would use its ancestor, the HP48, but this doesn't seem to be the way the HP49G was thought to do its work. It rather seems that the "standard" method should be to carry on all operations in exact mode, getting exact intermediate results, and at the end, to use  $\rightarrow\text{NUM}$  or  $\text{XNUM}$  to get the "numbers". The second example of above, carried out with exact numbers (integers) would be  $1 \ 3 \ / \ 3$ , which results in  $\frac{1}{3} \ 3$ . If you use  $\rightarrow\text{NUM}$  on this result you will still get

$.333333333333$  and not 1. But if you first expand the expression  $\frac{1}{3} \ 3$  and then use  $\rightarrow\text{NUM}$ , then the result is 1. The first example is a bit more complicated to be calculated exactly, but it is possible. Consider the operation  $1. \ 1.E12. \ R \rightarrow I \ 1.E-12 \ 1.E12.$

$R \rightarrow I \ + \ 1E12 \ R \rightarrow I \ /$ , which results in  $\frac{1000000000001}{1000000000000}$ . If you use  $\rightarrow\text{NUM}$  here, you get again 1. instead of the correct result,  $1.000000000001$ , so you may think what was the use of the exact calculation. In this case we didn't win anything because the precision loss was less than the 12 digits that the HP49G uses. But in more complicated calculations the precision loss may become bigger and our results worse.

The same problems occur when we carry out symbolic operations with symbolic arguments. Consider the classical example  $\text{SIN}$ , which returns a fat 0, while  $\rightarrow\text{NUM} \text{ SIN}$  returns  $-2.0676153735E-13$ , a result very close to but not exactly 0. So

# Sequences, series and limits with the HP49G - Part 6

the first moral of the story is that we should carry out all operations exact, and at the end, when we want the "damned numbers", we just use  $\rightarrow$ NUM, or XNUM, or we switch to numeric mode (by pressing at the same time [red-shift] and [ENTER]) and pressing [EVAL].

Let's now get a closer look to the definitions of precision and accuracy. Suppose that you have to do with a number  $x$ , which is not known exactly but only with an error of (positive)  $\epsilon$ . The number will then be somewhere between  $x - \epsilon$  and  $x + \epsilon$ . Does this  $\epsilon$  has to do something with the HP49G? Suppose that you have a number, say 1.23456 of which you know that it has an error of .01. Then the number "is meant to be" is somewhere between the numbers 1.22456

and 1.24456. The precision is defined as  $-\text{LOG} \frac{\epsilon}{x}$ . So the

precision of the above number is  $-\text{LOG} \frac{.01}{1.23456} = 2.09151220163$ .

The accuracy is defined as  $-\text{LOG}(\epsilon)$ , which for the above number is  $-\text{LOG}(.01) = 2$ . The following table contains some examples of numbers, errors, precisions and accuracies. (The small "e" denotes the "E" of HP49G numbers, so  $1e-12$  means  $1E-12$ .)

x		Precision	Accuracy
3,14	0,001	3,4969296481	3
0,0076	0,00033	1,3622996524	3,4814860601
100	0,002	4,6989700043	2,6989700043
6,023e+23	100000000000	13,779812863	-10
1,23456	1e-12	12,091512202	12

We take a look at the last example on the table. We have the number, 1.23456 and know that it has an error of  $1E-12$ . The number is somewhere between  $1.23456 - 1E-12$  and  $1.23456 + 1E-12$ . But for the HP49G all numbers between  $1.23456 - 1E-12$  and  $1.23456 + 1E-12$  are the same. Enter 1.23456,  $1E-12$ , and press [+] or [-]. You get 1.23456 in both cases. You get something

different than 1.23456, only when the magnitude of the second operand is greater than  $5E-12$ . It seems that  $\epsilon$  depends on the number that we have to do with. There is a minimal value for  $\epsilon$  which makes the quantities  $x - \epsilon$  and  $x + \epsilon$  to be distinguishable from each other and from  $x$ . If  $\epsilon$  is less than this minimal value, then no distinction is possible for the HP49G. We try to find out what this minimal  $\epsilon$  is. A real number on the HP49G can be represented using at most 12 digits (for us, normal users). If we add some positive  $\epsilon$  to a number  $x$ , then the result will be different than  $x$ , when  $\epsilon$  is at least so big that it "changes something" at the last (right most) digit of the number  $x$ . Since the number  $x$  can have up to 12 digits, up to 12 different powers of 10 are present in the number. For example in the number 12345.5678901 the powers 5,4,...0,-1,-2,...-7 of 10 are present, as you see by writing the number in its representation with powers of 10:

$$1 \cdot 10^5 + 2 \cdot 10^4 + \dots + 5 \cdot 10^0 + 5 \cdot 10^{-1} + 2 \cdot 10^{-2}, \dots + 1 \cdot 10^{-7}$$

The difference between 5, the highest power of 10 in the number, and 12, the number of digits, is  $5 - 12 = -7$ , which is equal to the lowest power of 10 in the number. In case the number  $x$  doesn't have 12 digits (like for example 12345.567), think it in the 12 digits approximation of the HP49G, that is as 12345.5670000. The number  $\epsilon$  that added to 12345.5678901 returns something different than 12345.5678901, must then be at least about  $1E-7$ , which has an exponent of -7 (base 10). Actually,  $\epsilon$  can be at least  $5.00000000001E-8$ , which has the next lower exponent of 10 and a mantissa of 5.00000000001, which is the next possible number bigger than 5 on the HP49G. The exponent of 12345.5678901 can be found by using the command XPON (on the second page of menu MTH/REAL). So the exponent of  $\epsilon$  is equal to  $\text{XPON}(x) - 12$ . And itself is equal to  $5.00000000001 \cdot \text{ALOG}(\text{XPON}(x) - 12)$ . For example in the case of the number 12345.5678901 of above, we have:

# Sequences, series and limits with the HP49G - Part 6

= 5.000000000001 ALOG(XPON(12345,67890) - 12) =  
5.000000000001E - 8

Any positive less than 5.000000000001E - 8 added to 12345.5678901 will produce the same number, 12345.5678901, on the HP49G. The above way to calculate will work only for  $\text{XPON}(x) - 12 \geq -499$ , that is for  $\text{XPON}(x) \geq -487$ . If the exponent of  $x$  is even smaller, then  $\text{XPON}(x) - 12 < -499$ . For example with  $x = 1\text{E} - 499$  we have  $\text{XPON}(x) - 12 = -511$ , and  $\text{ALOG}(-511) = 0$ , that is we would calculate

= 5.000000000001 0 = 0, which is wrong, since we can add  $1\text{E} - 499$  to  $x = 1\text{E} - 499$  and get the result  $2\text{E} - 499$ . So we refine our calculation of . If the number has an exponent of less than -487, we simply return  $1\text{E} - 499$ . Else we use the above formula for . But there is still a problem. Since the greatest possible number that the HP49G can express, is 9.999999999999E499 there is no positive that we can add to that number and still get something different than 9.999999999999E499. The biggest number for which the formula for works is (for the HP49G) 9.999999999998E499. Above this number there is no that we can use. So we make the last refinement. If the number that we have is 9.999999999998E499, then we return ?, else if the number has an exponent of less than -487, we return  $1\text{E} - 499$ , else we use the above formula for . We make a user defined function that does all the above. Go to the EQW and type in:

$$(x) = \text{IFTE}(x > 9.999999999998\text{E}499, ?, \text{IFTE}(\text{XPON}(x) < -487, 1\text{E} - 499, 5.000000000001 \text{ALOG}(\text{XPON}(x) - 12)))$$

Press [ENTER], then [blue-shift], [2] to DEFINE the function that calculates the smallest possible value for that can be added to the number  $x$  and return a result different than  $x$ . Let's try some examples. The results are on the table on the right. You just enter the number  $x$  and press the menu key [ ] in the variables menu.

x	
3.14	5.000000000001E-12
1.E78	5.000000000001E66
100.	5.000000000001E10

Now that we have we can use it to define the greatest possible precision of given number on the HP49G. Go to the EQW again and

type in:  $\text{PRCSN}(x) = -\text{LOG} \frac{(x)}{x}$ . Press [ENTER] to put the

equation on stack level 1 and DEFINE the function. The function uses the already defined function . It calls and passes the argument  $x$  to it. Similarly we can define a function for the greatest possible accuracy of a given number. In the EQW type in  $\text{ACRCY}(x) = -\text{LOG}((x))$  and then define the function. We make a small table with results of all three functions for some numbers.

x		PRCSN(x)	ACRCY(x)
3.14	5.000000000001E-12	11.7979596437	11.3010299957
1.E78	5.000000000001E66	11.3010299957	-66.6989700043
100.	5.000000000001E10	11.3010299957	9.30102999566
4.57E15	5000.000000001	11.9609461957	-3.69897000434
9.34E-20	5.000000000001E-32	12.2713768719	31.3010299957

We see that though precision remains almost constant at about 12, accuracy varies very strongly and can even become negative! A negative accuracy may sound bad, but it doesn't have to be bad at all. Look at its definition to understand why.

We come now to some stuff about limits that has been asked quite often in the group. Can the HP49G get limits of more than one variables? If it does, then how can tell it to do so?

The good news is that the HP49G can handle such limits. But you

# Sequences, series and limits with the HP49G - Part 6

have to provide some additional information. Let's start with an example. We take a function of 2 variables,  $\frac{\sin(X - Y)}{X}$ . What is the limit of this function when  $X \rightarrow 0$  and at the same time  $Y \rightarrow 0$ ? Well, this is not enough information for finding the limit. We also need something else, namely *how the variable Y approaches 0, compared to how variable X approaches 0*. Let's suppose that  $Y = X$ , that means variable Y goes to 0 exactly the way variable X goes to 0. The limit of the function is then...0! That might sound strange since we know that  $\lim_{x \rightarrow 0} (\sin(x)) = 1$ . But it is true. Go to the

EQW and enter  $\lim \lim \frac{\sin(X - Y)}{X}, Y = X, X = 0$ . Press

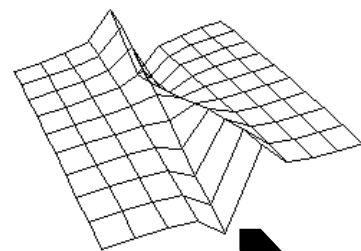
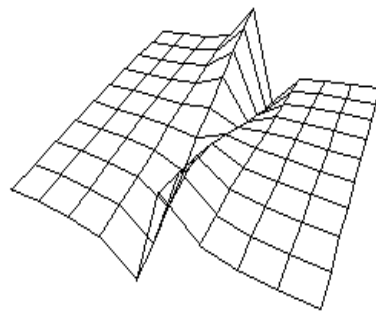
[ENTER] to put the expression as  $\lim_{X \rightarrow 0} \lim_{Y \rightarrow X} \frac{\sin(X - Y)}{X}$  on stack

level 1. Expand to get 0. Perhaps we should do a plot to understand this better. First we do a Fast3D surface plot of the function  $\frac{\sin(X - Y)}{X}$ , just to get an idea of how the surface looks. Press and

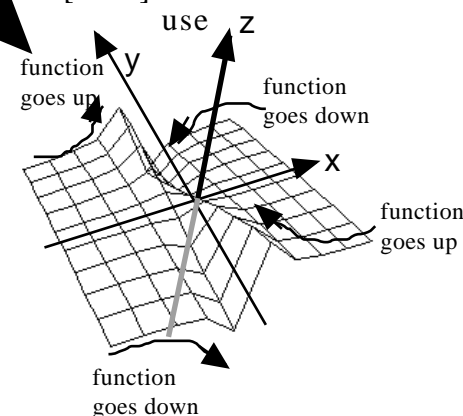
hold down [blue-shift], and press [F4]. Release both keys and you come at the "PLOT SETUP" screen. Choose plot type "Fast3D". Go to the input field "EQ:". Press [EQW] to go to the equation writer for easy formula typing. Type in:

$\frac{\sin(X - Y)}{X}$ . Press [ENTER] to put

the function in the input field "EQ:". Enter 'X' for independent and 'Y' for dependent variable. Now, press and hold down [blue-shift], and press [F2]. Release both keys and you come at the "PLOT WINDOW - FAST3D" screen. Enter -1 to 1 for "X-Left:" and "X-right:", -1 to 1 for "Y-near:" and "Y-Far:" and -4



rotate the plot and you can use the keys [+] and [-] to zoom in and out. Rotate and zoom in and out until you find some orientation and viewing distance that let you understand the plot better. When you see the plot almost like on the picture on the left, then the orientation of the axes and the function itself in space is as annotated in the picture below. At  $X = 0$  extraordinary things happen.

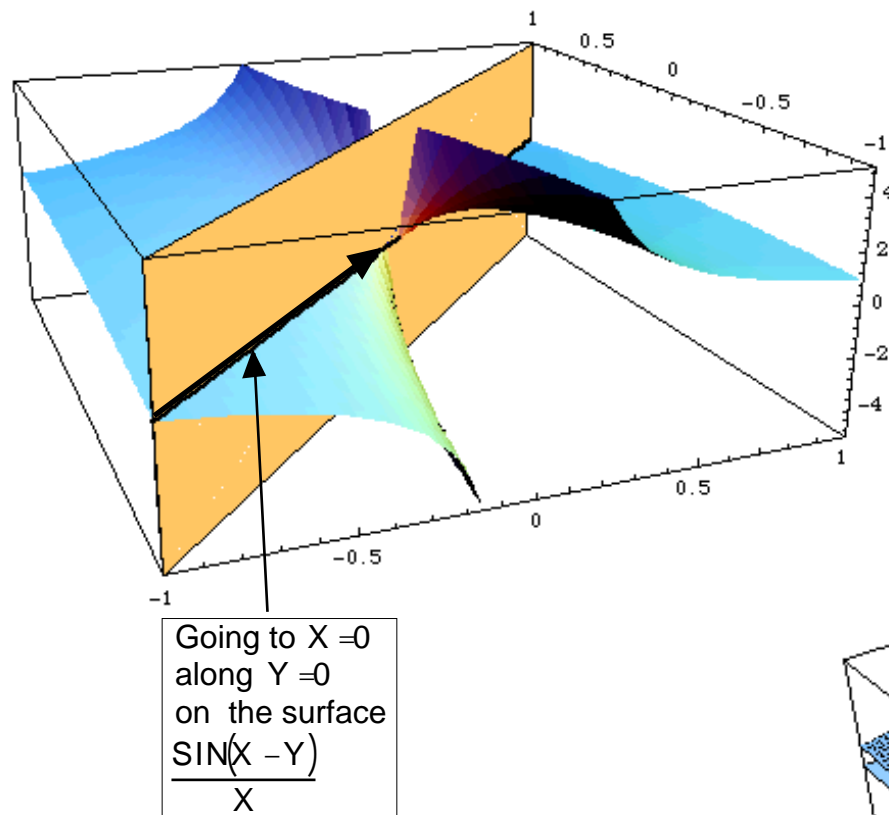


As you may have noticed, we have some kind of gap here. The function seems to jump from - to + and vice versa. Actually it does, but the limit of the function  $\frac{\sin(X - Y)}{X}$  when  $X \rightarrow 0$ , depends on how Y alters with altering X. In our case, we had the limit

$\lim_{X \rightarrow 0} \lim_{Y \rightarrow X} \frac{\sin(X - Y)}{X}$ , which implies that  $Y = X$ . Let's follow the surface along the line that is defines by  $Y = X$ . That means that we choose a path that lies on the surface  $\frac{\sin(X - Y)}{X}$ , and which in addition has always the property that the coordinate Y is equal to the

to 4 for "Z-Low:" and "Z-High:". Enter 10 for both steps of the independent and the dependent variable. Press the menu key [ERASE] and then the menu key [DRAW]. The HP49G draws the plot which is not very understandable at the first look. But you can use the arrow keys, the key [TOOL] and the key [NXT] to

# Sequences, series and limits with the HP49G - Part 6

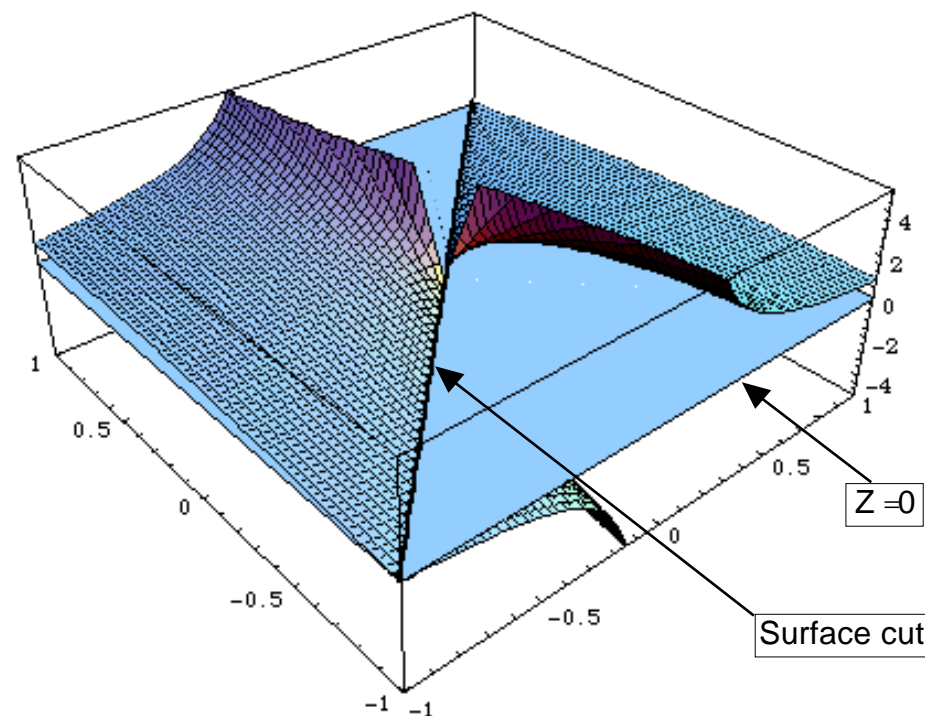


coordinate  $X$ , which is exactly what the equation  $Y = X$  says. The picture above demonstrates this. (Unfortunately the picture was not made with the HP49G. But who can say what the future might bring. Making the substitution the future = Rcobo in the last sentence, we derive the sentence: who knows what Rcobo might bring ;-)) As you can see, walking on the (blue) surface along the path  $Y = X$  (thick line in yellow box) towards  $X = 0$ , let's us stay on the same  $Z$ -value, namely 0, no matter how close to  $X = 0$ .

Since we now know that all values of  $\frac{\text{SIN}(X-Y)}{X}$  along  $Y = X$  are equal to 0, we can also plot the same function together with the plane  $Z=0$  and take a look at the curve on the cut of the two surfaces. This is what the picture below shows. The light blue plane is the surface of  $Z = 0$ . The cut with the surface  $\frac{\text{SIN}(X-Y)}{X}$  is the thick straight line, which for  $X \neq 0$  remains

at the height  $Z = 0$ . Note that the plots on this page are only a demonstration and by no means a rigid proof of the fact that

$$\lim_{X \rightarrow 0} \lim_{Y \rightarrow X} \frac{\text{SIN}(X-Y)}{X} = 0. \text{ They are only a help for}$$





# Sequences, series and limits with the HP49G - Part 6

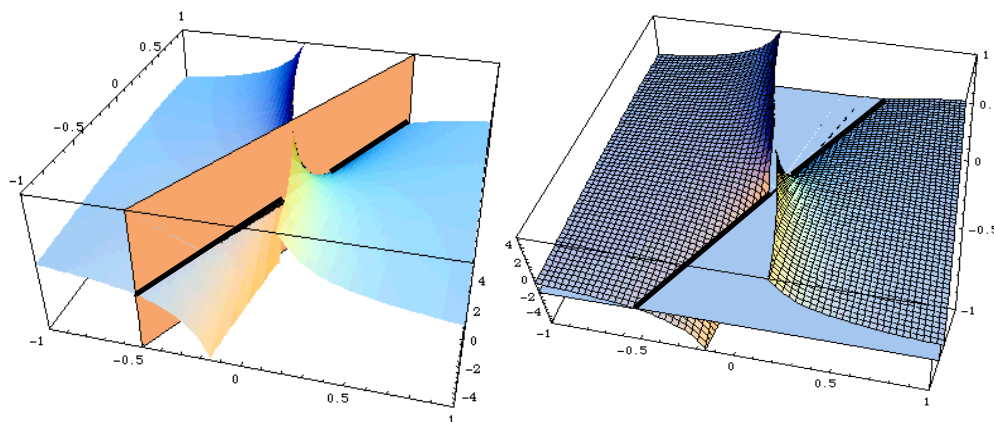
understanding how the both function and the limit behave.

Now, there are many other (infinite) ways in which  $Y$  can dependent on  $X$ . What happens for example when  $Y = 2X$ ? Can we find

$\lim_{X \rightarrow 0} \lim_{Y \rightarrow 2X} \frac{\sin(X - Y)}{X}$  ? You of course guessed that we can. We

enter  $\lim \lim \frac{\sin(X - Y)}{X}, Y = 2X, X = 0$ , we press [EXPAND],

and soon we get  $-1$ . Now, I guess that you want to see that on a plot again, right? So here you are, the same plots as on the previous page, but this time for  $Y = 2X$  and. As we can see, all points on the thick



black curve are at  $Z = -1$ . The curve only *looks* like a straight line that cuts the surface along  $Y = 2X$  at  $Z = -1$ , but we'll soon see that

this is not true. And as we can also see, in both limits of  $\frac{\sin(X - Y)}{X}$

both variables  $X$  and  $Y$  approach  $0$ . The first is taken for  $X \rightarrow 0$ , and  $Y = X$ , which means that also  $Y \rightarrow 0$ . The second is taken for  $X \rightarrow 0$ , and  $Y = 2X$ , which means that also  $Y \rightarrow 0$ . Nonetheless the first limit was found to be  $0$ , while the second was  $-1$ , which shows that in such limits not only the value is important, which  $X$  and

$Y$  go to, but also how  $Y$  depends on  $X$ . The expression

$\lim_{X \rightarrow 0} \left( \lim_{Y \rightarrow 0} (Z(X, Y)) \right)$ , without the information about how  $Y$  depends on  $X$ , is in the most cases not sufficient for limit calculation.

Curves (which don't have to be straight lines) on surfaces  $Z(X, Y)$ , that satisfy  $Z(X, Y) = \text{constant}$ , are named with the help of a composed word that has two parts. The first part is equi- (or aqui-), or iso<sup>10</sup>-. The second part depends on what the surface represents. If for example we plotted the potential, then the curves are named "isopotential", or equipotential, and so on. There is a special kind of plot, the contour plot, that draws such curves for given constant values of  $Z(X, Y)$ . (If you remember page 2-23 of the complex numbers marathon, then the picture at the lower part of the right column was the aqui-height contour plot of Mt-Pilatus.)

Of course the HP49G can do contour plots. But they are often not of good quality. Let's do such a plot for the function

$\frac{\sin(X - Y)}{X}$ . Press and hold down [blue-shift], and press

[F4]. Release both keys and you come at the "PLOT SETUP" screen. Choose plot type "Ps-Contour". Go to the input field

"EQ:" and enter  $\frac{\sin(X - Y)}{X}$ . Enter 'X' for independent and

'Y' for dependent variable. Press and hold down [blue-shift], and press [F2]. Release both keys and you come at the "PLOT

WINDOW - PS-CONTOUR" screen. Enter  $-2$  to  $2$  for "X-Left:" and "X-right:" and  $-1$  to  $1$  for "Y-near:". Enter  $10$  for both steps of the independent and the dependent variable. Press the menu key [ERASE] and then the menu key [DRAW]. The HP49G starts drawing small line segments that are parts of the contour curves (aqui-whatever curves). It needs a loooong time (almost 6.5 minutes) to finish the plot and it doesn't draw the curves as a whole, but chopped in small segments, like the picture on the next page. (This picture is still much better than

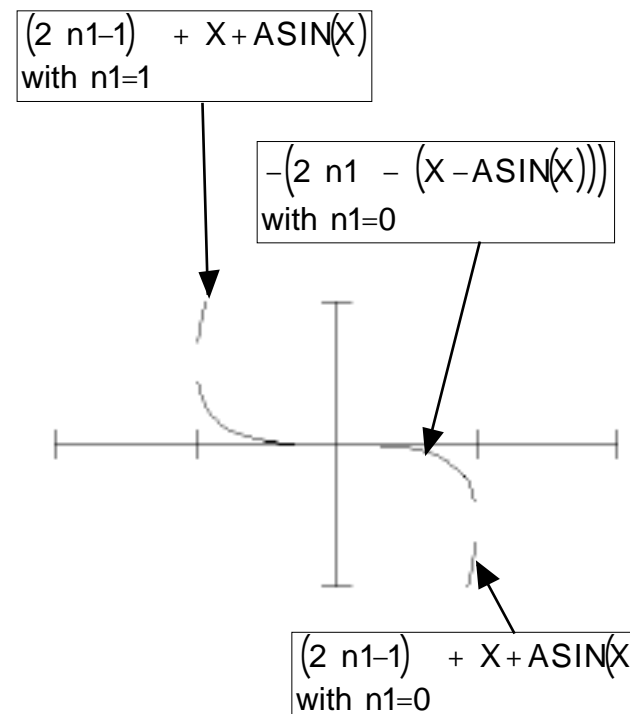
<sup>10</sup> From the greek word "Ison" for "equal". The character "i" is pronounced like the "ee" in the word "tree".

# Sequences, series and limits with the HP49G - Part 6

what the calculator draws!) When the plot is ready you can trace it. Press the menu key [TRACE] and then the menu key [(X,Y)]. The cursor goes to the upper left line segment and on the lower part of the display you see "INPUT: {-1.8 .9}". These are the values of X and Y. If you press [F2], then you see "OUTPUT: .237433266797". This is the value of Z at this point. Press [F2] once again to remove display of coordinates and then once more to show the values of X and Y again. Let's find a curve that belongs to an approximately constant value of Z. Using the arrow keys go to the top of the screen, at third line segment starting from the left. Display the value of Z, it is about .95. Press [arrow-down] once. The Z value is now about .99, approximately the same. Press [arrow-down] once again. The Z value is again about 1., again approximately the same. Another press of [arrow-down] moves to a line segment with Z=.96. Now press [arrow-down] and then [arrow-right] to come to Z=1.07. You may already have the impression that the curve with Z=1. goes like an "S" from the upper left part to the centre of the screen. The rest of this curve continues from the centre to the lower right part of the screen. The same way you can find (approximately) the contours with Z=0 and Z=-1, of the previous pages. But what kind of curve is the contour with Z=1? It is definitely not a straight line, but what kind of curve does it represent? Well, to find that we can simply solve the equation  $\frac{\sin(X-Y)}{X} = 1$  for the variable Y. Enter the equation, enter Y and press [ZEROS] (second page of menu S.SLV). The result is the list of solutions



$\{(2^{n1-1}) + X + \text{ASIN}(X) - (2^{n1} - (X - \text{ASIN}(X)))\}$ . We have two solutions, or better two families of solutions, which both are functions of X and depend also on the arbitrary integer n1. It seems that there are either many curves that belong to each family, or that each integer value of n1 constructs a different segment of one curve. If we plot these functions for several different integer values of n1, we can see what the curve is, that represents the cut between  $Z = \frac{\sin(X-Y)}{X}$  and Z=1. With the list of solutions on stack level 1, enter STEQ to store the list in EQ. We will plot first for n1=-1, so enter -1 'n1' STO. Press and hold down [blue-shift], and press [F4] to go to the "PLOT SETUP" screen. Choose the plot type "Function". Leave 'X' as independent variable. Press and hold down [blue-shift], and press [F2] to go to the "PLOT WINDOW" screen. Enter -2 to 2 for "H-View:" and -1 to 1 for "V-View:". Enter 1 for "Step:" of the independent variable and check the option \_Pixels. Press the menu key [ERASE] and then the menu key [DRAW]. The HP49G doesn't plot anything. Let's change the value of n1 to 0. Exit the plotter and enter 0. Press [blue-shift] and

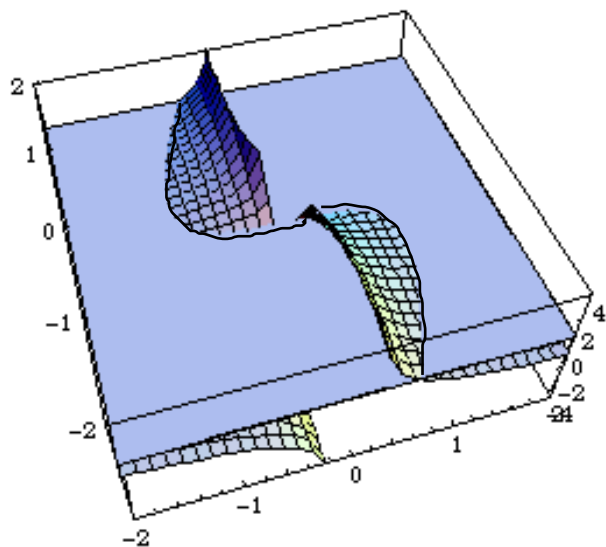
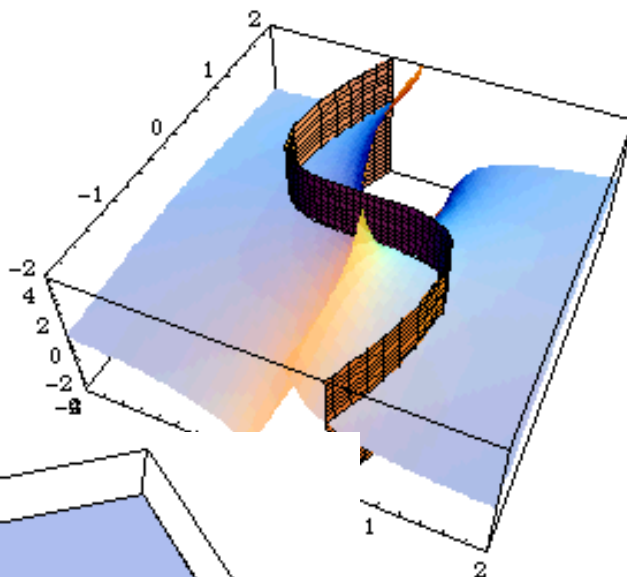




# Sequences, series and limits with the HP49G - Part 6

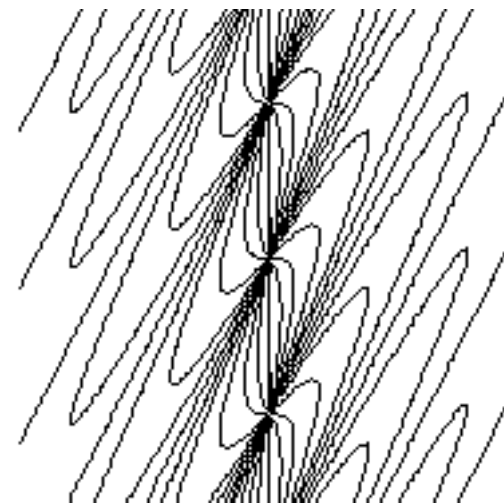
then the menu key [n1] in the variables menu, to store 0 in n1. Enter DRAW. Now the HP49G draws two curves. Store 1 in n1 and DRAW again. The plot now looks like the "S"-curve that we suspected at the contour plot. As we can see, the whole curve is not a function but a relation. Each segment of the curve forms a function that depends on X and contains also the parameter n1.

Let's take a look at the same contour in three dimensions. The second picture is especially for seeing how the curve of the cut of  $Z = 1$  with  $Z = \frac{\sin(X - Y)}{X}$  forms.



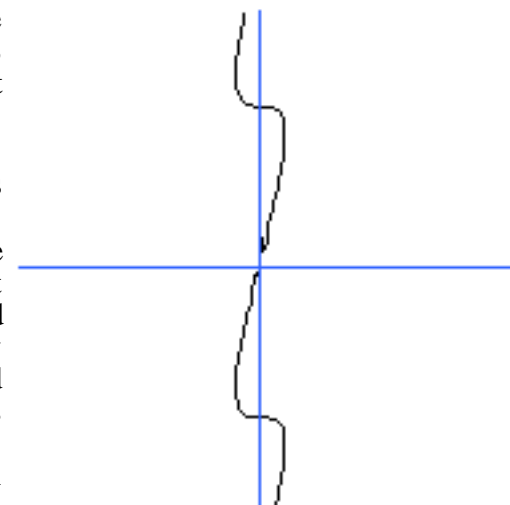
We already have examined the case  $Z = -1$  and we saw that it is almost a straight line at the vicinity of  $X = 0$ ,  $Y = 0$ . Now, we just make the view range larger and we take a closer look at this case. Make a

contour plot first. Let X go from  $-10$  to  $10$ , and Y from  $-5$  to  $5$ . Adjust the number of steps to 20 in horizontal and 10 in vertical direction. When after a looong time the HP49G is ready, the plot looks like a very chopped and ziggy-zaggy version of the picture above. If the resolution were better, then you could find through tracing that the contours that belong to  $Z = -1$  are not straight lines. You can see these contours on the picture at the bottom of this column.



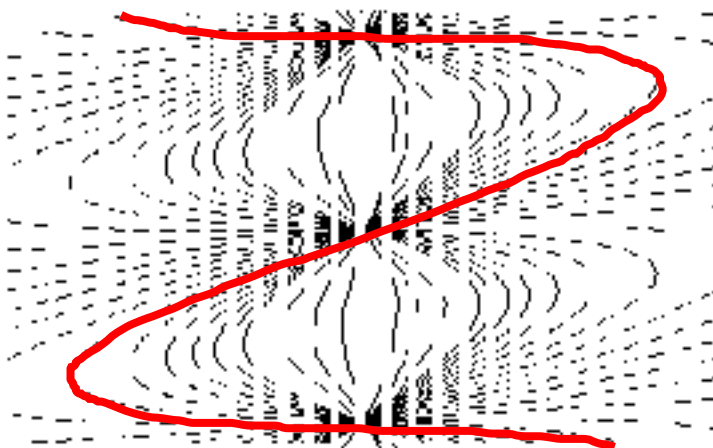
Of course we can't change the physical resolution of the screen, but we can do something else to make the quality of the contour plot better. We can plot exactly the region of interest and specify a big number of steps in x- and y-direction. Go to the PLOT WINDOW - PS CONTOUR screen and set X go from  $-1.5$  to  $1.5$ , and Y from  $-3.5$  to  $3.5$ .

Adjust the number of steps to 30 in horizontal and 15 in vertical direction. Before you draw let's take a look at yet another unexpected thing. Press the menu key [OK] to leave the screen and return to the stack. Press [VAR] to go to the variables menu and press [NXT] until



# Sequences, series and limits with the HP49G - Part 6

you see **VPAR** in the menu. Press the corresponding soft menu key to put the variable on the stack. Now press arrow down to edit it. Go to the last two numbers of the list which are the number of steps in x- and y-direction. You see what happened? While the number of steps in y-direction was correctly set to what we entered (15), the number of steps in x direction is not 30 but 20! The graphic user interface thought that 20 steps in x-direction are enough, and silently ignored that we entered 30 and used 20 instead, without finding it necessary to inform us. This is of course not so good. ;-) So change the second last number to 30, press enter to put the edited list on the stack and then press [blue-shift] and the menu key [ **VPAR** ] to store the edited list in VPAR. Now, don't use any graphics setup screen for the plot but simply press or enter DRAX and DRAW and go for a coffee while the HP49G is plotting. As we have  $30 \times 15 = 450$  line segments to plot, each of which must first be calculated, it takes a long time until the plot is finished. When it is ready you can find through tracing that the contour for  $Z = -1$  is approximately the thick red line on the picture to the right. Despite the big number of line segments the plot is still not very satisfactory. It is better than before but still it lets you only imagine what the contour curves look like, and that only on some parts of the plot, while on other parts the curves are nearly unrecognisable.



Until now we have examined limits of functions  $Z(X, Y)$ , in which  $Y$  behaves in such a way, that  $Z$  remains the same over the path that is defined by  $X$  and  $Y$ . This doesn't have to be this way. For example

the limit of  $\frac{\text{SIN}(X - Y)}{X}$  for  $X \rightarrow 0$ ,  $Y \rightarrow 0$  and  $Y = X^2$  will not give an aqu-height curve of the surface. We will examine such case a later on, but for now we try to find a way to draw the aqu-height curves a bit better. Yes, we are going to enhance the contour plots on the HP49G.

Let's stay at  $\frac{\text{SIN}(X - Y)}{X}$ . Suppose that we want to draw the contour

at  $Z = -1$ . That means, we must somehow turn on all pixels that correspond to coordinates  $(X, Y)$ , for which the equation holds:

$$\frac{\text{SIN}(X - Y)}{X} = -1. \text{ This equations can be understood as an implicit}$$

definition of  $Y$ , which can be made explicit by simply

solving for  $Y$ . Enter  $\frac{\text{SIN}(X - Y)}{X} = -1$  and  $Y$ , and press

[SOLVE]. The result is the solutions list:

$$Y = -(2 \cdot n1 - (X + \text{ASIN}(X)))$$

$$Y = (2 \cdot n1 - 1) + X - \text{ASIN}(X)$$

. As we already have seen, we can plot these two curves for some different integer values of  $n1$ , and get the curves that we want to see. Exactly this will be what a program should do, that draws

contour plots. We give it the equation  $Z = \frac{\text{SIN}(X - Y)}{X}$  and

a list of  $Z$  values, for which we want to see the contours.

Then the program puts all  $Z$  values in  $Z = \frac{\text{SIN}(X - Y)}{X}$ ,

solves for  $Y$  (or  $X$ ) and draws the resulting functions. An additional problem is that the solutions may contain additional variables, like  $n1$ , which don't appear in the original function. In this case it should ask the user to supply values for such variables. A very interesting problem that appears here is that we may have more than one additional variables, and the user can supply several values for each of them.

# Sequences, series and limits with the HP49G - Part 6

The program must then construct all possible combinations of values for these variables. We will see in a few moments a very unorthodox way to do that. So, here is the program listing - we will describe the program after the listing.

```
<<
PUSH
"CONTOUR PLOT"          @Title for INFORM
{
  { "Z(X,Y):"           @Start input fields definit.
    "Enter two variables function"
    9.}
  {}                    @Empty field
  { "Var. X:"           @Input field X-variable
    "Enter X-variable"
    6.}
  { "X-Range:"          @Input field X-range
    "Enter X-variable range"
    5.}
  { "Var. Y:"           @Input field Y-variable
    "Enter Y-variable"
    6.}
  { "Y-Range:"          @Input field Y-range
    "Enter Y-variable range"
    5.}
  { "Var. Z:"           @Input field Z-variable
    "Enter Z-variable"
    6.}
  { "Contours:"         @Input Z-contours
    "Enter Z-contours"
    5.}
}                        @End input fields definit.
{ 2. 0.}                @INFORM Format
{}                        @Default inputs
{}                        @Reset inputs
IF                        @If user pressed OK
THEN
  IF                      @If inputs missing
    DUP NOVAL POS
    THEN
      "Missing Data" DOERR @Abort
    ELSE
      OBJ-> DROP {}
      -> func xvar xrng
      yvar yrng zvar zcnt
      newvars
      <<
        IF                @If the first elem. of
          zcnt HEAD TYPE  @contour values list is
          2. ==           @a string
        THEN              @then the user wants
          "Calculating Z-values
of contours..." 1 DISP  @n contours from zmin to
          zcnt 3 GETI UNROT @to zmax. Calculate them.
          GET DUP2 - NEG
          zcnt 2 GET 1 - /
          -> zmin zmax step
          <<
            {} zmin zmax
            FOR I
              I + step
            STEP
          >>
          'zcnt' STO
        END
        xrng 1             @Convert evtl. algebraics
        << ->NUM >>        @to numbers for all
        DOSUBS             @appropriate inputs
        'xrng' STO
        yrng 1
        << ->NUM >>
        DOSUBS
        'yrng' STO
        zcnt 1
        << ->NUM >>
        DOSUBS
      >>
    >>
  >>

```

# Sequences, series and limits with the HP49G - Part 6

```

'zcnt' STO
IF
  func LNAME AXL
  zvar POS NOT
THEN
  zvar =
END
'func' STO
CASE
  1 CF
  "Solving for "
  yvar + "
" + 1 DISP xvar INDEP
  yvar DEPND
  xrng OBJ-> DROP
  DUP2 XRNG XVOL
  yrng OBJ-> DROP
  DUP2 YRNG YVOL
  func yvar
  IFERR
    ZEROS
  THEN
    DROP 1 SF
  END
  1 FC?C
THEN
  END
  yrng OBJ-> DROP
  DUP2 XRNG XVOL
  xrng OBJ-> DROP
  DUP2 YRNG YVOL
  yvar INDEP xvar
  DEPND "Solving for "
  xvar + "
" + 1 DISP yvar
  IFERR
    ZEROS
  THEN
    DROP 1 SF
  END
  1 FC?C
THEN
  END
  1 FS?
  OVER {} SAME
  OR
  THEN
    "No symbolic solution.
Numeric not implem." DOERR
  ELSE
    DUP STEQ
    "Filtering variables..."
    1 DISP
    IF
      DUP TYPE 5
    THEN
      1 ->LIST
    END
    1
    <<
    LNAME AXL DUP
    1 OVER xvar POS
    1 - SUB SWAP DUP
    xvar POS 1 +
    OVER SIZE SUB +
    DUP 1 OVER yvar
    POS 1 - SUB SWAP
    DUP yvar POS 1 +
    OVER SIZE SUB +
    DUP 1 OVER zvar
    POS 1 - SUB SWAP
    DUP zvar POS 1 +
    OVER SIZE SUB +
    @Set flag 1 (solving failed)
    END
    1 FC?C
    THEN
    END
    1 SF
    END
    IF
    1 FS?
    OVER {} SAME
    OR
    THEN
    "No symbolic solution.
Numeric not implem." DOERR
    ELSE
    DUP STEQ
    "Filtering variables..."
    1 DISP
    IF
    DUP TYPE 5
    THEN
    1 ->LIST
    END
    1
    <<
    LNAME AXL DUP
    1 OVER xvar POS
    1 - SUB SWAP DUP
    xvar POS 1 +
    OVER SIZE SUB +
    DUP 1 OVER yvar
    POS 1 - SUB SWAP
    DUP yvar POS 1 +
    OVER SIZE SUB +
    DUP 1 OVER zvar
    POS 1 - SUB SWAP
    DUP zvar POS 1 +
    OVER SIZE SUB +
    @If user entered f(x,y)
    @convert to z=f(x,y)
    @In case
    @Prepare plotting env.
    @adjusting all parameters
    @according to inputs
    @If solving z(x,y) for y
    @errors out
    @then indicate failure
    @through setting flag 1
    @If flag 1 clear? clear
    @Do nothing!!!
    @In case solving for y
    @failed (flag 1 was set)
    @adjust plotting paramet.
    @and try to solve for x.
    @Else store sols. in EQ
    "Filtering variables..."
    @Find additional variables
    @If sol. wasn't list
    @then convert it to list
    @Start of outer DOSUBS proc.
  
```

# Sequences, series and limits with the HP49G - Part 6

```

IF
  DUP {}
THEN
  1
  <<          @Start of inner DOSUBS
    IF        @proc
      newvars
      OVER POS
    THEN
      DROP
    ELSE
      newvars
      +
      'newvars'
      STO
    END
  >>          @End of inner DOSUBS proc
  DOSUBS
  ELSE
    DROP
  END
>>          @End of outer DOSUBS proc
DOSUBS
DROP
END
IF          @If we have additional vars
  newvars {}
THEN
  CLLCD LCD-> @Prepare and show informat.
  {#0d #0d}   @screen about new variables.
  "Solutions contain new variables.
You must specify their values in
the next input form. If you don't
enter any, then {0} will be used.

Press OK to continue."
  1 ->GROB REPL
  ->LCD

```

```

{{{}}}{}}{}{"OK" CONT}}
TMENU 3 FREEZE @Menu with "OK" to continue
HALT          @Wait for user response
"NEW VARIABLES" @Title for INFORM
newvars 1      @Construct input field
<<            @specification for each
  ":" OVER SWAP @new variable.
  + "Enter values for "
  ROT + 5 3 ->LIST
>>
DOSUBS
newvars SIZE 4 / @Construct format spec.
IP 1 + 1 2 ->LIST
{} {} INFORM
IF          @If user pressed [OK]
THEN
  1          @If values list wasn't
  <<        @input, then use {0.}
    IF
      DUP NOVAL
      SAME
    THEN
      DROP {0.}
    END
  >>
  DOSUBS
ELSE        @Else (user pressed [CANCL])
  {} 1 newvars
  SIZE
  START      @Use {0.} for each new var.
  {{0.}} +
  NEXT
END
1          @Convert user input to num.
<< -NUM >>
DOSUBS      @Store list
newvars SWAP 2 @{{{newvars}}{values}} in
->LIST 'newvars'@newvars

```

# Sequences, series and limits with the HP49G - Part 6

```

STO
END
FUNCTION #0d RES    @Setup, and show PICT    << "
{#0d #0d} PVIEW
DRAX LABEL
zcnt 1
<<                @Do for each contour
  zvar STO          @Store in global Z-var
  IF
    newvars {}      @If we have new variables
  THEN
    newvars OBJ->
    DROP
    ->nvars nvals @Store in locals
    <<            @Do for each new variable
      "<< "        @Start constructing prog.
      1 nvars SIZE
      FOR N
        "1 " + nvals
        N GET ->STR
        + " SIZE FOR I

      +
      IF
        N 1 ==
      THEN
        "{} -> vals

      << "

      +
      END
      " vals " +
      nvals N GET
      ->STR +
      " I GET + "
      +
      IF
        N nvars SIZE
        THEN
          " -> vals
          +
          ELSE
            nvars ->STR
            +
            " STO DRAW "
            +
            END
          NEXT
          1 nvars SIZE
          FOR N
            IF
              N nvars SIZE
            THEN
              " NEXT >> "
              +
              ELSE
                " >> NEXT >> "
                +
                END
              NEXT          @Convert string prog. to
              OBJ-> EVAL    @program and run it.
            >>
            ELSE            @else (no new vars)
              DRAW          @simply draw
            END
            IF              @If we have new variables
              newvars {}
            THEN            @Then take var. names
              HEAD
            END
            zvar + PURGE    @Purge Z-var and new vars.
            >>
            DOSUBS
            RCEQ func EQ->

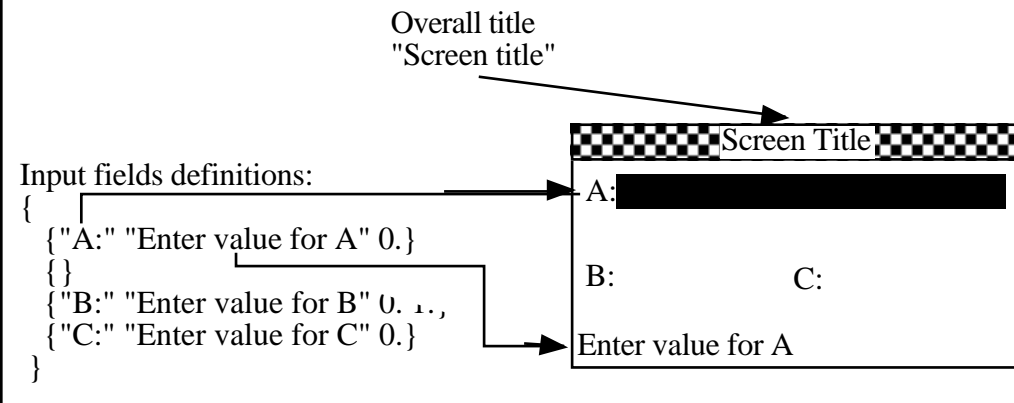
```

# Sequences, series and limits with the HP49G - Part 6

```

IF          @get function out of
  DUP TYPE 6 == @z=f(x,y) or f(x,y)=z
THEN
  DROP
ELSE
  NIP
END
STEQ 10 NUMX @Prepare for built-in
10 NUMY PCONTOUR @contour plot
PICTURE
>>
END
END
-1 MENU POP @Restore last menu and flags
>>

```

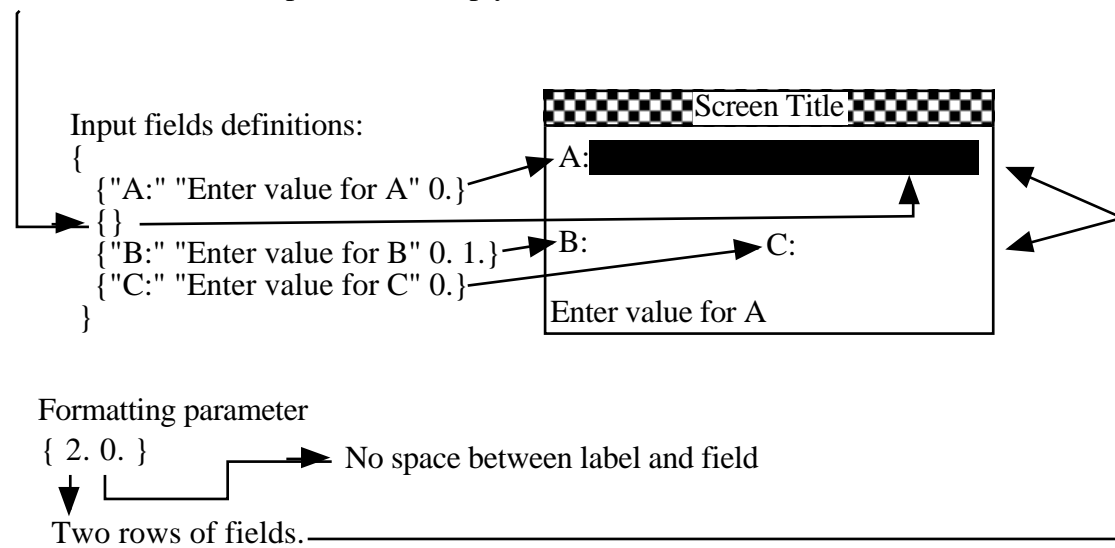


This is the program CONTOURPLOT that comes along with this document. I know that you are eager to test it but let's describe some programming techniques used in the program first.

First of all the program put the arguments for INFORM on the stack. So, let's see how this command works. It takes 5 parameters from the stack and shows an input screen. From stack level 5 the command takes the overall label, a string, which appears at the top of the of the input screen. From stack level 4 the command takes the definitions of the input fields. This is a list which consists of sub lists, one sub list for each field. The first element of each sub list is a string, which appears in front of the corresponding input field. The second element is a string which appears as a help for the user at the bottom of the screen, when the corresponding input field is selected. The rest of the elements are object types of the valid objects for the corresponding input field. For example, the input field "A:" of the screen at the top right of this page will accept only real

numbers. The object types of the valid object types must be real numbers, that is with decimal point. An empty list specifies an empty input field that can be used for formatting the screen. From stack level 3 the command takes the formatting parameter which is also a list. It

Empty input field. The previous input field "A:" covers the space of the empty field.



# Sequences, series and limits with the HP49G - Part 6

contains two real numbers the first of which specifies in how many rows the input fields will be shown. The second specifies how much space will be left free between the label of the input field and the field itself. The next parameter is a list containing the inputs that appear in the fields when we reset the screen. When the input screen appears, the menu line contains a menu with three items. "EDIT" lets you edit the object in the selected input field in the command line. "CANCL" cancels the dialogue and returns a 0 on stack level 1. This can be also done by pressing the key [ON]. "OK" exits the dialogue. It returns all inputs in a list on stack level 2 and a 1 on stack level 1. This can be also done by pressing the key [ENTER]. If some input field is empty when you press the menu key [OK] (or the key [ENTER]) then the list with the inputs will contain a NOVAL for the input field without input. As you can see, the command provides all necessary output for distinguishing between "CANCEL" or "OK", and for finding out which inputs were left empty by the user. If you press the key [NXT] then the second page of the menu appears. The first item, "RESET", lets you reset the selected or all input fields to their reset objects which you provided as a list on stack level 2. It shows a popup menu which allows to select between resetting the selected or all input fields. The next and last parameter is a list that contains the default inputs for the fields. These appear when the screen is initially

Screen Title	
A:	
B:	C:
Enter value for A	
EDIT	CANCL OK

Screen Title	
A:	
B:	C:
Enter value for A	
RESET	CALC TYPES CANCL OK

Screen Title	
A:	Reset value Reset all
B:	C:
Enter value for A	
RESET	CALC TYPES CANCL OK

shown. The second menu item, "CALC", allows you to temporarily leave the input screen and go to the stack. But this stack is not the same like that which you worked until the input screen appeared. This is a new separate stack, which only contains the object of the selected input field at the time when you pressed "CALC". On the top of the screen the title of the input screen is still shown. Also the help string that normally is shown at the bottom of the screen, now is shown under the title, so that you still now what the calculator expects you to enter. If you press the menu key [CANCL], then this separate stack is abandoned and you return to the input screen without changing the object of the selected field. But if you press [OK] then the object at stack level 1 replaces the object of the selected input field. Pressing the menu key [STS] changes the header of the stack display to the normal header, that is information about the current path and other indicators are displayed. If you press some key that activates another menu while you are in the separate stack, then the header changes and displays again the title of the input screen and directly under the title it displays a message, which tells you how to return to the menu of the input screen. That means, press [blue-shift] and then [ON] to get the menu that allows you to return to the input screen. In the separate version of the stack you can do anything that can be done in the normal stack, that is calculations, run programs etc. If while in the input screen you press the key [HIST], then you land at the well known interactive stack, which contains anything that was on the stack before the input field was shown. You can move up and down using the keys [arrow-up] and [arrow-down], and copy any object to the command line of the input screen by pressing the menu key [ECHO]. You can leave history

Screen Title	
Enter value for A	
5:	
4:	
3:	
2:	
1:	3.456789
STS	CANCL OK

Screen Title	
Press [CONT] for menu	
5:	
4:	
3:	
2:	
1:	3.456789
Some other menu	



# Sequences, series and limits with the HP49G - Part 6

by pressing either [ON] or [ENTER]. Finally the menu key [TYPES] of the input screen, displays a message box that contains all valid object types for the selected input field.

Let's go on with the description of the program. After data input the program first examines what the first object of the list is, which you entered in the input field "Contours:". If it is a string, then the program assumes that the user wants  $N$  contours from  $Z = Z_{\min}$  to  $Z = Z_{\max}$ . That means that the list will be interpreted as  $\{\text{string } N \ Z_{\min} \ Z_{\max}\}$ . Not a very elegant method for choosing this option, but what else can we do when we have no check-field available in the built-in version of INFORM? If the first item of the list in "Contours:" is not a string, then the program assumes that the user wants exactly those contours, whose  $Z$ -values are given in the list. Note the usage of the command GETI, which is quite similar to GET, but it also automates the index incrementing when getting elements out of a list or matrix. While the command GET only returns the  $i^{\text{th}}$  element of a list, the command GETI returns the  $i^{\text{th}}$  element at stack level 1, the index  $i+1$  at stack level 2, and the list at stack level 3. If the index was at the last element, then execution of GETI wraps the index back to 1 and sets the flag -64 to indicate that we arrived at the end of the list. The following two code snippets return identical results applied to a list is on stack level 1:

```
1 OVER SIZE
FOR I
  DUP I GET
  SWAP
END

-64 CF
1
WHILE
  -64 FC?
REPEAT
  GETI UNROT
END
DROP
```

The program does some conversions, like for example converting functions that were input as  $f(X,Y)$  to  $Z = f(X,Y)$ . Notice that not all error checking is done. For example if you enter a list like  $\{1 \ 2 \ "2"\}$  in the input field "Contours:" then the program will crash. But you may add your own error checking code of course.

The next thing that the program does is, trying to solve  $Z = f(X,Y)$  for the variable  $Y$  (or any other name that you enter in the input field "Var. Y:"). If it finds solutions then it proceeds, else it tries to solve  $Z = f(X,Y)$  for variable  $X$ . If it finds solutions it proceeds, else it errors out with "No symbolic solution. Numeric not. implem.". That means, the program tries to convert the equation to  $Y = g(X,Z)$  or to  $X = h(Y,Z)$ , in order to plot it later as a function for several values of  $Z$ . These  $Z$ -values are those given in the input field "Contours:". It is interesting to examine how the CASE THEN END clause works, that is used at this point. We have two things to examine when we check if solutions were found. The one thing is to check if the returned solution was an empty list. The other thing is to check if the HP49G errored out while trying to solve for  $Y$  or  $X$ . The normal usage of the CASE THEN END clause is:

```
CASE
  Test1
  THEN
    Action1
  END
  Test2
  THEN
    Action2
  END
  ..... further tests and actions
  Default action
END
```

But we use the clause in a very different style:

# Sequences, series and limits with the HP49G - Part 6

```
CASE
  Several actions and attempt to solve for Y and set
  flag 1 if appropriate (indicator for no solution)
  Test flag 1
THEN
  Do nothing
END
  Several actions and attempt to solve for X and set
  flag 1 if appropriate (indicator for no solution)
  Test flag 1
THEN
  Do nothing
END
  Set flag 1
END
```

The clause is used as a kind of switch or sieve that exploits the fact that the CASE clause will keep on checking each case until some case is true, no matter if an action is performed after the check. We must only provide a test (or the result of a test) for the THENs of the clause. This is what FC?C is used for. If the two attempts for solving  $Z = f(X,Y)$  didn't work, then we have to use numeric methods. Perhaps at this point you would like the program to simply plot a built-in contours plot. Or use a numeric method. Or do something else. It is up to you to decide.

The next thing the the program does is checking if the solutions contain new additional variables, which don't appear in the original function. This can happen when for example we use SOLVE with trigonometric functions. Variables like  $n1$ ,  $n2$ , and so on might appear in the solutions, which would make the plot impossible, if no values for them are specified. If the program discovers such variables, it displays a message for information and waits for you to press the menu key [OK]. Notice that the word "OK" of this message is written in inverse in the program, in order to imitate the looking of the menu key. This can be done by pressing the key [TOOLS] while editing some object, then pressing [NXT] twice, then pressing the menu key [Style] and selecting some style. The characters that you type after

pressing the menu key of a style, will appear in this style. Pressing the key for the same style again just deactivates that style.

If there were new additional variables the program constructs the arguments for an additional INFORM screen, where you can specify values for those variables.

And then, oh! then comes another unorthodox part, which shows how flexible this machine is. But let's see first what the problem is and why the unorthodox solution. Suppose that you have an arbitrary number of variables  $n1, n2, \dots$ , each of which may have an arbitrary number of distinct values. How can you construct a list containing every distinct combination of values of those variables? This is a non-trivial problem. It can be solved using recursive programming but because we already have seen many examples of recursive programming, we use another method here. We let the program... construct another program according to the number of the variables and their values and let it run to built-up those combinations of values. The program just concatenates strings (code snippets in blue bold in the program listing) which form a new program. When ready, it uses OBJ-> to compile the string and generate the runnable program, and then runs it using EVAL. This resembles macro programming and inline include of macro code in the SAS System.

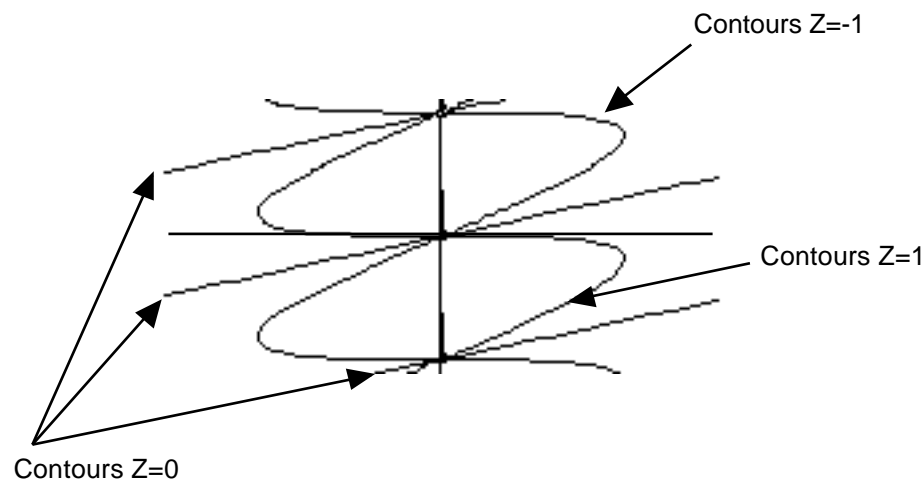
Then the program draws the curves of the contours for all Z-values that we specified and activates the PICT environment so that you can observe the curves and how they behave.

When you exit the PICT environment, the program returns the found solutions  $Y = g(X,Z)$  or  $X = h(Y,Z)$ , stores the original equation in EQ and sets all parameters for plotting the corresponding built-in contours plot. That means, in case you want it, you can superimpose a built-in contour plot with the curves that the program plotted, by simply entering DRAW.

Enough explanations, now we test! Enter ERASE to erase any old plots. In the menu VAR press [CONTO] to run the program CONTOURSPLOT. We will plot the contours of our old friend

# Sequences, series and limits with the HP49G - Part 6

$\frac{\sin(X-Y)}{Y}$  for  $Z = 0$ ,  $Z = 1$ , and  $Z = -1$ . When the INFORM screen appears, press [EQW], enter  $\frac{\sin(X-Y)}{Y}$  and press [ENTER]. The expression is put in the input field "Z(X,Y):" and the next field "Var. X:" is selected. Enter X. Now the next field is selected. Enter the list  $\{-1.5 \ 1.5\}$  as range for X. Then enter Y for "Var. Y:",  $\{-3.5 \ 3.5\}$  for "Y-range:", Z for "Var. Z:" and  $\{-1 \ 0 \ 1\}$  for "Contours:". Press the menu key [OK]. The program shows some messages and then the text that informs you about the existence of new variables. At that point press the menu key [OK]. A new INFORM screen appears, where you can enter the values for the variable n1. Enter the list  $\{-1 \ 0 \ 1\}$ , which means that the contours will be drawn for  $n1 = -1$ ,  $n1 = 0$ , and  $n1 = 1$ . Press the menu key [OK]. The program needs some time to plot the curves of the three contours. When ready you should see the picture below. When you finish looking at the plot, press the menu key [CANCL]. The program puts



the solutions found  $Y = -(2 \ n1 - (X + \text{ASIN}(X)))$  on stack  
 $Y = (2 \ n1 - 1) + X - \text{ASIN}(X)$

level 1, prepares all necessary things for a subsequent built-in contour plot. If you wish, you can use the command DRAW to plot that on the already plotted curves.

On with limits of functions with two variables. We examine the limit of  $\frac{\sin(X-Y)}{Y}$  for  $X \rightarrow 0$  and  $Y \rightarrow 0$  for the case (path)  $Y = X^2$ . Go

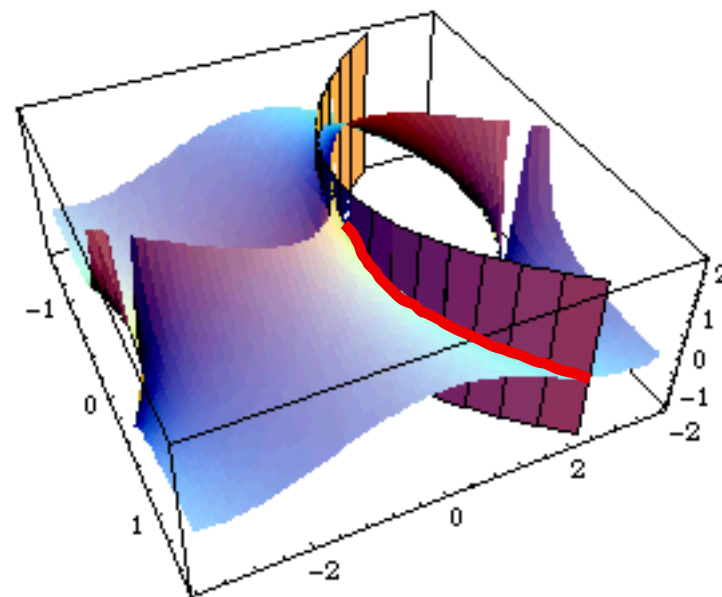
to the EQW and enter  $\lim \lim \frac{\sin(X-Y)}{X}, Y = X^2, X = 0$ . Press

[ENTER] to put that on the stack and press [EXPAND] to get a 1 as result. Is that correct? Well, we can visualise this with Mathematica. In the picture below we see the path that we are on when we follow the

coordinates  $X, Y = X^2, Z = \frac{\sin(X-Y)}{X} = \frac{\sin(X-X^2)}{X}$  (the red

thick curve). It shows that such paths don't have to be at a constant value for Z.

If we put the plane  $Z = 1$  on the same



# Sequences, series and limits with the HP49G - Part 6

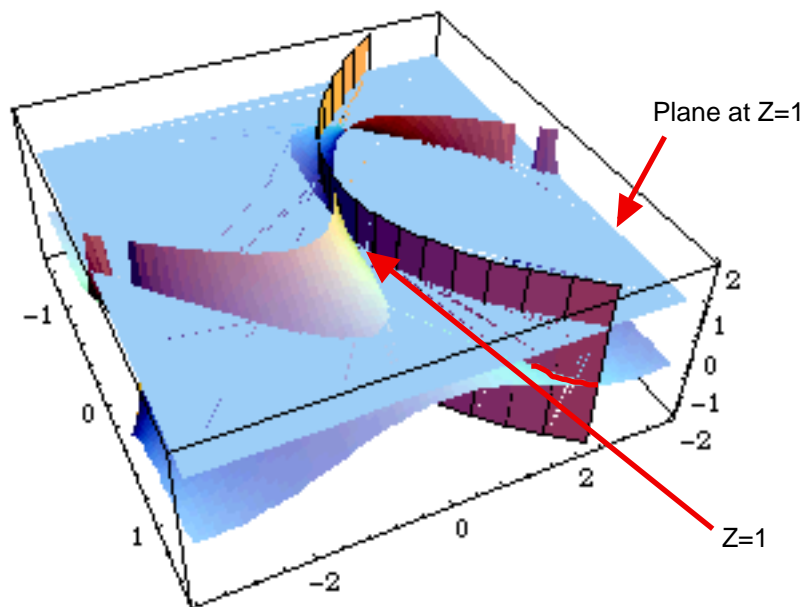
plot, then we can see that the limit that the HP49G returned was correct. So we are able to calculate such limits, but what about visualising them? Let's see. We have the function

$$Z = \frac{\sin(X - Y)}{X}$$

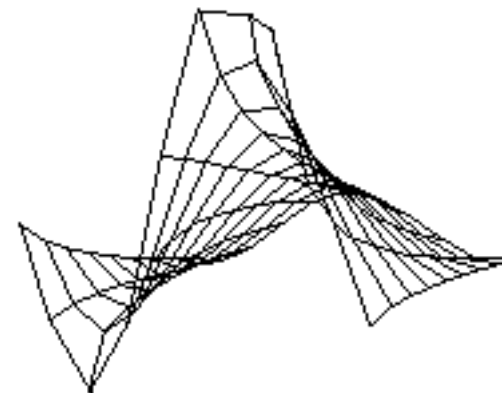
which we can plot using the plot type Fast3D or Wireframe. We would like to superimpose it with some additional graphics that make the stuff more understandable. So we can't use Fast3D as this plot type doesn't allow more than one graphics to be superimposed. We are going to use Wireframe. We will plot first  $\frac{\sin(X - Y)}{X}$ . Let's do that. Go to the PLOT-SETUP screen and chose

plot type Wireframe. Enter  $Z = \frac{\sin(X - Y)}{X}$  as the function to plot and

X for independent, Y for dependent variable. Go to the PLOT WINDOW - WIREFRAME. Set "X-Left:" to .1 (avoiding X = 0 to prevent the plot from going off screen, "X-Right:" to 1.5, "Y-Near:" to -3.5, "Y-Far:" to 3.5, "Z-Low:" to -4, "Z-High:" to 4. Set the coordinates of the point from which you observe the plot, "XE:" to -7, "YE:" to -20 and "ZE:" to 1. Set "Step Indep:" and "Depnd:" to 10. Press the menu key [ERASE] and then [DRAW]. The HP49G plots the wireframe which then looks like the picture at the top of the right of



this  
page.  
Press



[CANCL] to exit the PICT and go to the PLOT SETUP screen again. Now we will add the path X,

$$Y = X^2, Z = \frac{\sin(X - Y)}{X} = \frac{\sin(X - X^2)}{X}$$

as a three dimensional parametric curve on the same plot. The HP49G has no plot type for three dimensional parametric curves. But it has parametric surface plot type. So we will fake it. (Again, shame on us! ;-))

Chose plot type Pr-Surface. This plot type expects a list of three expressions in EQ, which define X, Y, and Z. Enter the list

$$X \quad X^2 \quad \frac{\sin(X - X^2)}{X}$$

in the input field "EQ:". Leave X and Y for independent and dependent variables. As you can already see we have the two variables X and Y but our parametric expressions depend only on X (curve). This is the first step of our faking. Go to the PLOT WINDOW - PR-SURFACE screen. Press the menu key [XXYY]. Enter for "XXLeft:" .1, for "XXRight:" 1.5. Leave all other fields as they are except the number of steps for "Depnd:", which you must change to 1. (This is the second step of the faking.) Press the menu key [DRAW] and wait until the plot is ready. The HP49G draws an additional curve on the previous plot. You should trace this curve

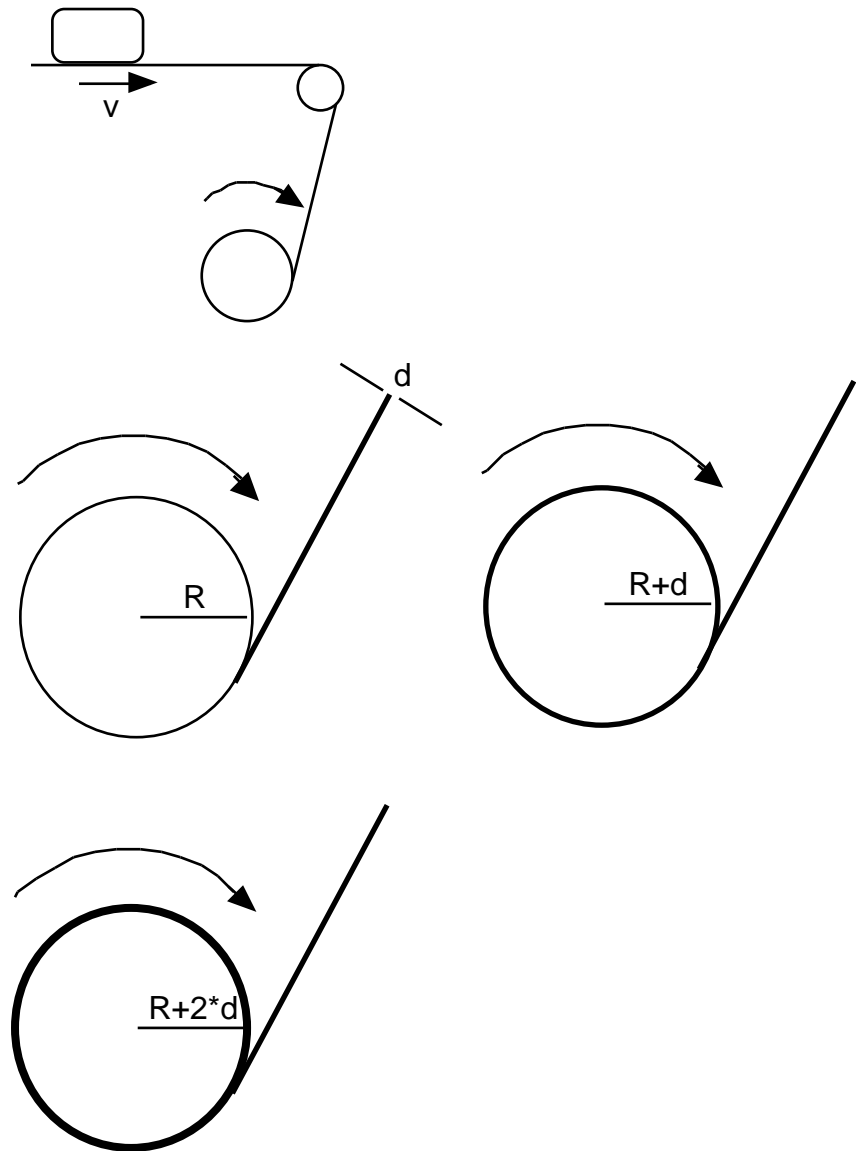
# Sequences, series and limits with the HP49G - Part 6

because it is not very visible. Press [TRACE]. A list with two input coordinates appears on the bottom of the screen. From these two input coordinates only the first (X) is interesting for us. (Remember? We fake it ;-)) Press [arrow-right] and follow the graphics cursor until it you see "INPUT: {.100000000008 " at the bottom of the screen. Press the menu key [F2] again to show the output coordinates. Unfortunately you can't see the value of the third coordinate

$Z = \frac{\sin(X - X^2)}{X}$ , but if you press [ENTER] the HP49G puts a list with the three output coordinates on the stack. Exit the plot, go to the stack and press [DTAG] and then [OBJ->]. Drop the list element count. You see that the third coordinate is .898785491972. It is approaches 1, and if you re-plot the Wireframe with "X-Left:" set to .01 and the Pr-Surface with "XXLeft:" set to .01, the returned number for  $Z = \frac{\sin(X - X^2)}{X}$  will be even closer to 1 as the plot is made for values of X that are even closer to 0. Of course you can also plot a three dimensional parametric curve alone in exactly the same way. There is no need to superimpose it with some other plot. This was done here only for demonstrating the behaviour of the limit.

Let's have another example of such a limit. Go to the EQW and enter  $\lim_{Y \rightarrow X} \frac{\sin(X - Y)}{Y}$ ,  $Y = X$  ( $X = 1$ ),  $X = 0$ . Put that on the stack and press [EXPAND]. After some seconds you get -2, the correct limit.

As a small example of a sequence/series problem we are going to examine... a music cassette. (Yes, I know that in modern times thus prehistorical sound carrier should have been replaced by a DVD ;-)) We are going to calculate the length of the tape with the help of a series. If the radius of the coiling role is R and the thickness of the tape is d, then each time the role roles 2 a new layer of tape is coiled over it, making the coiling radius a little bit thicker. The coiling radius starts at R, after 2 it is R + d, after another 2 it is



# Sequences, series and limits with the HP49G - Part 6

$R + 2d$ , and so on. The length of the tape that coils round the role in each rotation is respectively  $2R$ ,  $2(R + d)$ ,  $2(R + 2d)$ , and so on. The length of the the whole tape must

be  $\sum_{n=0}^N 2(R + nd) + L_0$ , where  $N$  is the number of rotations that the role has done at the end of the concert,  $n$  is the number of rotations so far, and  $L_0$  is the length of the tape that remains uncoiled. Let's

calculate that. Enter  $L = \sum_{n=0}^N 2(R + nd) + L_0$  and press

EXPAND to get  $L = (dN^2 + (2R + d)N + 2R) + L_0$ . This is the length of the tape.

The tape runs with a constant linear speed  $v$ , which means that we can calculate how long it will take until it ends. Since  $v = \frac{L_1}{te}$   $te = \frac{L_1}{v}$ ,

we have  $te = \frac{(dN^2 + (2R + d)N + 2R)}{v}$ , where  $te$  is the time when the tape ends, and  $L_1$  is the length of the tape without the part  $L_0$ , which remains uncoiled. Enter  $v = \frac{L_1}{te}$ , then enter  $te$  and

then press [SOLVE] to get  $te = \frac{L_1}{v}$ . Press [OVER] to copy the equation of stack level 2 to stack level 1. Press [arrow down] to get the equation to the EQW. In the EQW press [arrow down] once to select the left hand side. Then press [backspace] to put the cursor right after  $L$ , and press [1]. Press [arrow up] to select the left hand side, which now is  $L_1$ . Press [arrow right] to select the right hand side. Here we must delete the term  $L_0$ . Press [arrow down] to select the first term and then [arrow right] to select the second term. Press [red-shift] and then [backspace] to delete  $L_0$ , and then again [backspace] to delete the remaining  $+$ . Press [ENTER] to put the edited equation,

$L_1 = (dN^2 + (2R + d)N + 2R)$  on the stack. Then press

[SUBST] to get  $te = \frac{(dN^2 + (2R + d)N + 2R)}{v}$ .

While the linear speed of the tape is constant, the angular frequency changes while more and more tape coils around the role. Since

$v = R \cdot \omega = \frac{v}{R}$ , we have  $\omega_n = \frac{v}{R + nd}$ . This is a quantity that changes. It is a monotonically decreasing sequence. Here we see that decreases in discrete steps, as  $n$  increases. But because  $d$ , the thickness of the tape is so small, the sequence behaves "almost" like a function. (Actually the model that we used is much too simple, but it goes in the right direction for explaining why/how the angular speed of the role decreases with time.)

Before we end this last part of the rather lengthy marathon, let's look again at our table of programs on the next side. The new programs are rather stand alone without much connections with the rest. But anyway it looks like crazy, thus it must be good ;-)

This marathon has been so long and exhaustive mainly because of two reasons. The first reason is that the HP49G doesn't have much built-in functionality for sequence. So we had to program much. And because a program for symbolic mathematics is a "black box" when given without explanations, we had to explain much of the underlying theory. The other reason is... that the HP49G doesn't have INTEGERASSUME. We already have talked much about that, so leave it be. ;-)

But I think that it had the advantage of demonstrating some programming techniques and also of demonstrating how self contradicting and at the same time wonderful the HP49G is. Next time we will continue with the long awaited Calculus Marathon. May the force be with us.

Greetings,  
Nick.

# Sequences, series and limits with the HP49G - Part 6

